A

DLS 26-SEP-74 15:22 31098

3rd Test Run of JOVIAL Equations and Tables

(J31098) 26-SEP-74 15:22;;;; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RN2([INFO-ONLY]) RJC([INFO-ONLY]) ;
Sub-Collections: RADC; Clerk: DLS;

DLS 26-SEP-74 15:22 31098

3rd Test Run of JOVIAL Equations and Tables

This contains altered tab setttings to make the equations and tables line up better. They have noooot been edited for mispellings or errors of ommision.

3rd Test Run of JOVIAL Equations and Tables

**********EQUATIONS**********

Appendix A

SYNTAX EQUATIONS

The following pages contain the complete syntactic description of JOVIAL (J73). The metalinguistic equations are in alphabetical order of the metalinguistic terms being defined. In general, each defining equation is individually boxed. The boxes are nubered sequentially in the upper left hand corner by a number in italics followed by a colon. Following the colon is a list of the box numbers in which the current term is used as a part of the definitions of other terms. The metalinguistic symbology is explained in section 1.4.

Equations 94 and 95 are in one box. These are both valid and necessary definitions for format:list. Equation 144 is the definition for mark. In the same box, opposite each mark is a metalinguistic term (or two). These marks constitute the definitions of these terms. Equation 172 defines pattern:digit. In the same box is information giving the bit patter corresponding to each pattern:digit, depending on the order of the pattern:constant. In the box with equation 190, the definition of relational:operator, is a list of the meanings of the relational:operators. Box 234 contains a definition for system:dependent:character, but the definition is a prose description; a metalinguisitic equation is not feasible. Equations 247 and 248 are in one box. They are both valid and necessary definitions for variable.

1: 233

abbreviation ::= letter

2: 63

abnormal:directive ::= !ABNORMAL data:name ;

3: 130

absolute:function:call ::= ABS (numeric:formula)

4: 58, 170

actual:define:parameter ::=
definition
" definition "

5: 101, 170, 180, 191

STOP
alternate:entrance:name
RETURN
procedure:name
TEST control:variable

EXIT statement:name
actual:input:parameter ::=
statement:name
procedure:name
formula
table:name
data:block:name
variable
@ pointer:formula

6: 170, 180, 191

actual:output:parameter ::= variable

7: 166,217

allocation:increment ::= number

8: 9, 49, 166, 182, 205, 217

allocation:specifier ::= @ pointer:formula

9: 75, 184

alternate:entrance:declaration ::=

ENTER alternate:entrance:name

(formal:input:parameter

: formal:output:parameter)

environmental:specifier item:description
allocation:specifier

packing:specifier [bit:number]

= + constant ;
-

10: 130

alternate:entrance:function:call ::= ALT (procedure:name)

11: 5, 9, 53, 101, 122, 138, 180, 187, 193

alternate:entrance:name ::= name

```
12:    159

          +
          =
          *
   arithmetic:operator  ::=
          /
          \
          **


13:

   assignment:operator  ::=    =


14:    207

   assignment:statement  ::=
                  formula
          variable                  indexed:variable:range  ;
          indexed:variable:range =
                  format:function:call

                  formula
      format:variable    =                 ;
                  indexed:variable:range


15:    159

   attribute:association  ::=  @@  [  description:attibute  ]
```

16: 63

begin:directive ::= !BEGIN reference ;

17: 18

bit:form ::= form

18: 18, 29, 97, 159, 196

pattern:constant
entry:variable
comparison
chain:comparison
bit:string:function:call
shift:function:call
bit:formula ::= bit:form
bit:formula logical:operator bit:formula
NOT bit:formula
bit:formula & bit:formula
(bit:formula)
numeric:formula
character:formula

19: 9, 182, 205, 217, 218

bit:number ::= number

20: 18, 130

bit:string:function:call ::=

BIT (formula , numeric:formula ,
numeric:formula)

21: 247

bit:variable ::=

entry:variable
BIT (named:variable , numeric:formula ,
numeric:formula)

22: 217

bits:per:entry ::= number

23: 130

byte:string:function:call ::=

BYTE (character:formula , numeric:formula ,
numeric:formula)

24: 18

chain:comparison ::= comparison relation:operator
formula

25: 26, 32, 46, 62, 100, 112, 120, 137, 213, 234, 240

sign
character ::=
system:dependent:character

```
26:    29, 39
   character:constant  ::=    count    '  character      '

27:    29
   character:form  ::=  form

28:    29
   character:format  ::=      count    C

29:    18, 23, 93, 94, 97
         character:constant
         character:variable
         character:form
   character:formula  ::=        character:function:call
         character:formula  &  character:formula
         (  character:formula  )
         bit:formula

30:    29
   character:function:call  :=  function:call
```

31: 29, 96, 247

```
        named:character:variable
character:variable ::=      BYTE (
named:character:variable , numeric:formula

                         , numeric:formula    )
```

32: 233

```
comment ::= " character      "
```

33: 18, 24

```
comparison ::= formula relational:operator formula
```

34: 63

```
compool:directive ::=
                    name
          compool:name
  !COMPOOL              ( name )      ;

                      ( compool:name )
```

35: 34

```
compool:name ::= name
```

36: 219

declaration
compound:statement ::= BEGIN END ;
statement

37: 38, 97, 238, 241

conditional:formula ::= formula

38: 207

conditional:statement ::=

IF conditional:formula ; controlled:statement

statement:name ; ELSE
controlled:statement

39: 9, 42, 182, 205, 233

numeric:constant
constant ::= pattern:constant
character:constant

40: 97

constant:formula ::= (formula)

41: 166, 167, 217, 218

```
constant:list ::=

      [ index ]
    [ index ]    constant:list:element
   constant:list:element
            ,
```

42: 41, 42

```
          ,     +     constant     ,
constant:list:element ::=                -
      count ( constant:list:element )
      constant:list:element
```

43: 136, 148

```
      increment:phrase        terminator:phrase
      replacement:phrase
control:clause ::=      initial:phase
            increment:phrase
      terminator:phrase       replacement:phrase
```

44: 5, 239

```
      named:variable
control:variable ::=
      letter:control:variable
```

45: 38, 141

```
controlled:statement ::= statement
```

46: 63

copy:directive ::= COPY character ;

47: 26, 28, 42, 72, 78, 95, 98, 103, 120, 124, 126, 171, 173, 199

count ::= number

48: 182

data:allocator:specifier ::= @

49: 49, 51, 75

```
data:block:declaration ::=

                environmental:specifier
   BLOCK  data:block:name                    ;
                allocation:specifier

                simple:item:declaration
                table:declaration
      BEGIN                        END     ;
                data:block:declaration
                independent:overlay:declaration
```

50: 5, 49, 52, 90, 109, 138, 209

data:block:name ::= name

```
51:    54

          item:declaration
   data:declaration  ::= table:declaration
          data:block:declaration
          overlay:declaration


52:    2, 129, 174, 195, 244

          item:name
   data:name  ::=        table:name
          data:block:name


53:    130

          procedure:name
   data:size:function:call  ::=  DSIZE  (              )
          alternate:entrance:name


54:    36, 54, 112, 179

          status:list:declaration
          form:declaration
          data:declaration
          null:declaration
   declaration  ::=      define:declaration
          name declaration
          processing:declaration
          external:declaration
          BEGIN declaration      END    ;


55:    54

   define:declaration  ::=
```

```
DEFINE define:name    (  formal:define:parameter    )
"  definition  ";
```

```
56:    55, 58

   define:name  ::=  name
```

```
57:    4, 55

   definition  ::=  sign
```

```
58:

   definition:invocation  ::=  define:name    (
   actual:define:parameter      )
```

```
59:    185

   dependent:program:declaration  ::=  procedure:declaration
```

```
60:    15, 69

        item:name
   description:attribute  ::=
        item:description
```

(equ61)

61: 166, 217

dimension:list ::= [lower:bound : upper:bound
]

62: 207

direct:statement ::= DIRECT character JOVIAL ;

63:

compool:directive
skip:directive
begin:directive
end:directive
trace:directive
copy:directive
abnormal:directive
sets:directive
directive ::= uses:directive
pointer:directive
order:directive
recursive:directive
time:directive
space:directive
linkage:directive
interference:directive
frequency:directive

64: 233

```
        !COMPOOL
        !SKIP
        !BEGIN
        !END
        !TRACE
        !COPY
        !ABNORMAL
        !SETS
   directive:key  ::=    !USES
        !POINTER
        !ORDER
        !RECURSIVE
        !TIME
        !SPACE
        !LINKAGE
        !INTERFERENCE
        !FREQUENCY
```

65: 63

```
   end:directive  ::=  !END ;
```

66: 217

```
   entries:per:word  ::=  number
```

67: 18, 21, 117, 252

```
   entry:variable  ::=  table:name    (    index    )    @
   pointer:formula
```

68: 9, 49, 166, 182, 205, 217

```
                    program:name
          IN        procedure:name
environmental:specifier  ::=        RESERVE

     RESERVE
```

69: 159

```
evaluation:control  ::=  @  [  description:attribute  ]
```

70: 207

```
exchange:statement  ::=  variable  ==  variable  ;
```

71: 207

```
exit:statement  ::=  EXIT  statement:name  ;
```

72: 82

```
                  count    D      count    Z

        +
exrad   ::=               count    Z      count    D
        -

                  count    Z      *
```

73: 130

```
exrad:function:call ::= XRAD ( numeric:formula )
```

74: 132

```
exrad:specifier ::= number
```

75: 54

```
external:declaration ::=
                  simple:item:declaration
                  table:declaration
                  data:block:declaration
                  name:declaration
   DEF                  procedure:declaration
                  alternate:entrance:declaration
   REF                     simple:item:declaration
                  table:declaration
      BEGIN       data:block:declaration  END    ;
                  name:declaration
                  procedure:declaration
                  alternate:entrance:declaration
```

76: 87

```
field:width ::= number
```

77: 157

```
fixed:constant ::=

   number    ,
                        +              +
```

```
                    E                     scale     A
      scale
                              -                     -
      number     .    number



78:    158

                              *         count    D
                    integer:part
                              .         fraction:part

          +
   fixed:format  ::=                    integer:part
   count     *   R
          -

                    count     *
                                        fraction:part
                             .



79:    160

   fixed:function:call  ::=  function:call



80:    162

   fixed:variable  ::=  named:variable



81:    157

   floating:constant  ::=
                    +
         number     E    scale
                    -
        number      .                          M    +
      scale
```

```
                  +
         E                 scale
                  -
    number   .   number
```

82: 158

floating:format ::= significand E exrad R

83: 162

floating:function:call ::= function:call

84: 162

floating:variable ::= named:variable

85: 141

for:clause ::= FOR loop:control ;

86: 17, 27

form ::= form:name (formula)

87: 54

B

```
   form:declaration  ::=  FORM  form:name
   field:width       ;
         C



88:    86, 87

   form:name  ::=  name



89:    55, 170
   formal:define:parameter  ::=  letter



90:    9, 170, 182

         statement:name
         simple:item:name
   formal:input:parameter  ::=   procedure:name
         table:name
         data:block:name



91:    9, 170, 182

   formal:output:parameter  ::=  simple:item:name



92:    95

         null:format
         insert:format
   format  ::=   skip:format
         character:format
         pattern:format
         numeric:format
```

93: 14, 130

format:function:call ::=

FORMAT (character:formula , format:list
, procedure:name)

94: 93, 95, 96

format:list ::= character:formula

95: 93, 95, 96

format
format:list ::=
count (format:list)

96: 14, 102, 247

format:variable ::=

FORMAT (character:variable , format:list
, procedure:name)

```
97:    5, 14, 20, 24, 33, 37, 40, 86, 119, 159, 192, 203,
209, 242, 245

          pointer:formula
          numeric:formula
          bit:formula
   formula  ::=  conditional:formula
          character:formula
          value:formula
          numeric:formula
          constant:formula



98:    78

          count    D          count    Z

   fraction:part  ::=

          count    D



99:    130

   fraction:part:function:call  ::=  FRAC  (  numeric:formula
   )



100:   65

   frequency:directive  ::=  !FREQUENCY  character      ;
```

```
101:  30, 79, 83, 125

        intrinsic:function:call

        procedure:name
   function:call  ::=          @ pointer:formula
        alternate:entrance:name

              (    actual:input parameter       )



102:  248

        format:variable

        BYTE  (  named:character:variable  ,
   numeric:formula

   functional:variable  ::=          , numeric:formula
   )

        BIT  (  named:variable  ,  numeric:formula

              ,  numeric:formula    )



103:  158

   generalized:numeric:formula  ::=     count   N       R



104:  207

   go:to:statement  ::=   GOTO   statement:name   [
   index    ]   ;
```

105: 115

high:point ::= numeric:formula

106: 233

```
        +
        -
        *
        /
        **
        \
        &
        =
        ==
        <
        >
        <=
        >=
ideogram ::= <>
        .
        ,
        :
        ;
        !
        "
        #
        (
        )
        [
        ]
        @
        @@
```

107: 43

```
        numeric:formula
increment:phrase ::= BY
        numeric:value:formula
```

```
108:   49, 168

   independent:overlay:declaration  ::=

         [ number ]
      OVERLAY            independent:overlay:expression  ;
         [ pattern:constant ]



109:   111

         spacer
         simple:item:name
   independent:overlay:element  ::=      table:name
         data:block:name
         ( independent:overlay:expression )



110:   108, 109

   independent:overlay:expression  ::=

      independent:overlay:string    |
      independent:overlay:string



111:   110

   independent:overlay:string  ::=
   independent:overlay:element



112:   185

   independent:program:declaration  ::=
         statement
      PROGRAM program:name    ( character    )    ;
         declaration
```

113: 41, 67, 104, 237

index ::= index:component

114: 113, 116

index:component ::= numeric:formula

115: 116

index:component:range ::= low:point : high:point

116: 118, 155

index:component:range
index:range ::=
index:component

117: 150

table:variable
indexed:variable ::=
entry:variable

```
118:  14
   indexed:variable:range  ::=
                item:name
                                [ index ]      @ pointer:formula
                table:name
                     item:name
           ALL  (                      @  pointer:formula    )
                     table:name


119:  43
   initial:phrase  ::=  formula


120:  92
          count     S
                     numeral
   insert:format     ::=     count     /
                     letter
          count     "  character       "


121:  182
   instruction:allocation:specifier  ::=  pointer:formula
```

122: 130

```
      procedure:name
instruction:size:function:call  ::=  ISIZE  (        )
      alternate:entrance:name
```

123: 157, 175

```
integer:constant  ::=  number
```

124: 158

```
                count    Z          count    D
        +
integer:format  ::=                   R
        -
                count    D          count    Z
```

125: 160

```
integer:function:call  ::=  function:call
```

126: 78

```
      count    Z          count    D
integer:part  ::=
      count    D
```

127: 130

integer:part:function:call ::= INT (numeric:formula)

128: 162

named:variable
integer:variable ::=
letter:control:variable

129: 63

interference:directive ::= !INTERFERENCE data:name !
data:name ;

130: 101

format:function:call
byte:string:function:call
bit:string:function:call
alternate:entrance:function:call
number:of:entries:function:call
location:function:call
shift:function:call
absolute:function:call
words:per:entry:function:call
intrinsic:function:call ::= exrad:function:call
significand:function:call
signed:function:call
signum:function:call
size:function:call
type:function:call
fraction:part:function:call
integer:part:function:call
instruction:size:function:call
data:size:function:call

```
131:  51
        simple:item:declaration
   item:declaration  ::= ordinary:table:item:declaration
        specified:table:item:declaration


132:  9, 60, 166, 167, 182, 205, 217, 218

   item:description  ::=

      C    size:specifier

      F , R      significand:specifier      ,
      exrad:specifier

                 status:list
      S          status:list:name
        , R    size:specifier          +
      U          ,                precision:specifier
                 =


133:  52, 60, 118, 167, 187, 205, 218, 229, 237,

   item:name  ::=  name
```

134: 1, 89, 120, 135, 136, 145, 197, 221

letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

135: 44, 128, 233, 248

letter:control:variable ::= letter

136: 140

letter:loop:control ::= letter (control:clause)

```
137:   63
   linkage:directive  ::=  !LINKAGE    character      !


138:   130
         statement:name
         named:variable
   location:function:call  ::=  LOC  (   table:name      )
         data:block:name
         procedure:name
         alternate:entrance:name


139:   130
         AND
         OR
   logical:operator  ::=
         EQV
         XOR


140:   85
         named:loop:control
   loop:control  ::=
         letter:loop:control


141:   207
   loop:statement  ::=  for:clause  contolled:statement
```

142: 115

```
low:point ::= numeric:formula
```

143: 61

```
      number
lower:bound ::=
      simple:item:name
```

144: 197

```
        +       plus:sign
        -       minus:sign
        *       asterisk
        /       slash
        \       back:slash
        &       ampersand
        >       greater:than:sign
        <       less:than:sign
        =       equals:sign
        @       at:sign
mark    ::=     .       decimal:point
        :       colon
        ,       comma
        ;       semicolon
                space
        (       left:parenthesis, parenthesis
        )       right:parenthesis, parenthesis
        [       left:bracket, bracket
        ]       right:bracket, bracket
        '       prime
        "       quotation:mark
        $       dollar:sign
        !       exclamation:point
```

145: 11, 34, 35, 50, 56, 88, 133, 183, 186, 206, 220, 221, 225, 233, 236, 241

```
                  letter
        letter    numeral
name    ::=
          $         $
                    #
```

146: 54, 75

```
        statement:name
name:declaration  ::=  NAME              ;
        procedure:name
```

147: 31, 102

```
named:character:variable  ::=  named:variable
```

148: 140

```
named:loop:control  ::=  named:variable  (  control:clause
)
```

149: 219

```
named:statement  ::=  statement:name  :  statement
```

150: 21, 44, 80, 84, 102, 128, 138, 147, 148

```
        simple:variable
named:variable ::=
        indexed:variable
```

151: 54, 164, 215

```
        NULL    ;
null:declaration ::=
        BEGIN    END    ;
```

152: 92

```
null:format ::=
```

153: 219

```
        NULL    ;
null:declaration ::=
        BEGIN    END    ;
```

154: 7, 19, 22, 47, 66, 74, 76, 77, 81, 108, 123, 143, 169, 177, 189, 194, 201, 210, 214, 223, 232, 233, 243, 249, 250

```
number ::= numeral
```

```
155:  130

   number:of:entries:function:call  ::=  NENT  (  table:name
   [  index:range  ]    )



156:  120, 145, 154, 197

          0
          1
          2
          3
   numeral  ::=  4
          5
          6
          7
          8
          9



157:  39, 159

          integer:constant
          fixed:constant
   numeric:constant  ::= floating:constant
          status:constant
          qualified:status:constant



158:  92

          generalized:numeric:format
          integer:format
   numeric:format  ::=
          fixed:format
          floating:format
```

159: 3, 18, 20, 21, 23, 31, 73, 97, 99, 102, 105, 107, 114, 127, 142, 159, 161, 175, 196, 198, 200, 202, 232

```
        numeric:constant
        numeric:variable
        numeric:function:call
           +
               numeric:formula
           -

   numeric:formula  ::=  numeric:formula  arithmetic:operator

                  evaluation:control    numeric:formula
                   evaluation:control
        formula
                   attribute:association
        (  numeric:formula  )
        bit:formula
```

160: 159

```
        integer:function:call
   numeric:function:call  ::=    fixed:function:call
        floating:function:call
```

161: 97, 107

```
   numeric:value:formula  ::=  [  numeric:formula  ]
```

162: 159, 176, 247

```
        integer:variable
   numeric:variable  ::= fixed:variable
        floating:variable
```

```
163:   63

   order:directive  ::=  !ORDER  ;



164:   165

         null:declaration
         ordinary:table:item:declaration

   ordinary:table:body  ::=
                  ordinary:table:item:declaration
         BEGIN              END    ;
                  subordinate:overlay:declaration



165:   235

   ordinary:table:declaration  ::=  ordinary:table:heading
   ordinary:table:body



166:   165

   ordinary:table:heading  ::=
         environmental:specifier
      TABLE  table:name
         allocation:specifier

         :  allocation:increment    dimension:list

         structure:specifier        packing:specifier

         item:description        =  constant:list    ;
```

167: 131, 164

```
ordinary:table:item:declaration ::=

    ITEM item:name    item:description
    packing:specifier      = constant:list    ;
```

168: 51

```
        independent:overlay:declaration
overlay:declaration ::=
        subordinate:overlay:declaration
```

169: 9, 166, 167, 182, 205, 217, 218

```
                N
packing:specifier ::=       M    number
                D
```

170:

```
        actual:define:parameter
        formal:define:parameter
parameter ::=         actual:input:parameter
        formal:input:parameter
        actual:output:parameter
        formal:output:parameter
```

```
171:   18, 39, 108

          1
          2
   pattern:constant  ::= 3    B    count    '  pattern:digit
          4
          5



172:   171

          pattern pattern:digit   order

       0 0 0 0 0          0
       0 0 0 0 1          1              1
       0 0 0 1 0          2
       0 0 0 1 1          3              2
       0 0 1 0 0          4
       0 0 1 0 1          5
       0 0 1 1 0          6
       0 0 1 1 1          7              3
       0 1 0 0 0          8
       0 1 0 0 1          9
       0 1 0 1 0          A
       0 1 0 1 1          B
       0 1 1 0 0          C
       0 1 1 0 1          D
       0 1 1 1 0    pattern:digit  ::=    E
       0 1 1 1 1          F              4
       1 0 0 0 0          G
       1 0 0 0 1          H
       1 0 0 1 0          I
       1 0 0 1 1          J
       1 0 1 0 0          K
       1 0 1 0 1          L
       1 0 1 1 0          M
       1 0 1 1 1          N
       1 1 0 0 0          O
       1 1 0 0 1          P
       1 1 0 1 0          Q
       1 1 0 1 1          R
       1 1 1 0 0          S
       1 1 1 0 1          T
       1 1 1 1 0          U
       1 1 1 1 1          V              5
```

```
173:  92

          1
          2
   pattern:format  ::=  3      B      count    P
          4
          5


174:  63

   pointer:directive  ::= !POINTER   pointer:formula  ;
   data:name          ;


175:  5, 8, 67, 97, 101, 118, 121, 174, 180, 208, 237

          integer:constant
   pointer:formula  ::= simple:integer:variable
          (  numeric:formula  )


176:  247

   pointer:variable  ::= numeric:variable


177:  132

   precision:specifier  ::= number
```

178: 221, 233

```
        ABS
        ALL
        ALT       NENT
        AND       NOT
        BEGIN     NULL
        FIT       NWDSEN
        BLOCK     OR
        BY        OVERLAY
        BYTE      PROC
        DEF       PROGRAM
        DEFINE    REF
        DIRECT    REMQUO
        DSIZE     RESERVE
        ELSE      RETURN
        END       SHIFT
primitive ::=           ENTER   SIG
        EQV       SIGNED
        EXIT      SIGNUM
        FOR       SIZE
        FORM      STATUS
        FORMAT    STOP
        FRAC      SWITCH
        GOTO      TABLE
        IF        TEST
        IN        THEN
        INT       TYPE
        ISIZE     UNTIL
        ITEM      WHILE
        JOVIAL    XOR
        LOC       XRAD
        NAME      ZAP
```

179: 181

```
        declaration
procedure:body ::=
        statement
```

```
180:  207

   procedure:call:statement  ::=

      remquo:procedure:call:statement

         procedure:name
                 @ pointer:formula
         alternate:entrance:name

            (  actual:input:parameter      )

            (    actual:input:parameter      :        )

               actual:output:parameter    )



181:  59, 75, 184

   procedure:declaration  ::=  procedure:heading
   procedure:body



182:  181

   procedure:heading  ::=

                              environmental:specifier
   PROC procedure:name
                              data:allocation:specifier

      :  instruction:allocation:specifier

      (    formal:input:parameter        :
      formal:output:parameter      )

         environmental:specifier
                                  item:description
         allocation:specifier

      packing:specifier         [  bit:number  ]

         +
```

= constant ;

183: 5, 10, 53, 68, 90, 96, 101, 122, 138, 146, 180, 182, 167, 193

procedure:name ::= name

184: 54

program:declaration
processing:declaration ::= procedure:declaration
alternate:entrance:declaration

185: 184

independent:program:declaration
program:declaration ::=
dependent:program:declaration

186: 68, 112

program:name ::= name

187: 157

status:list:name
item:name
qualified:status:constant ::= V(table:name :
status)
procedure:name
alternate:entrance:name

```
188:  63

   recursive:directive  ::=  !RECURSIVE  ;



189:  16, 211

   reference  ::=  number



190:  24, 33, 246

         <        less than
         =        equal
         >        greater than
   relational:operator  ::=
         >=       greater than or equal, not less than
         <>       less than or greater than, not equal
         <=       less than or equal, not greater than



191:  180

   remquo:procedure:call:statement  ::=

      REMQUO  (  actual:input:parameter  ,
      actual:input:parameter
         !  actual:output:parameter  ,
         actual:output:parameter  )  ;
```

192: 43

```
        formula
replacement:phrase ::= THEN
        value:formula
```

193: 207

```
        procedure:name
return:statement ::= RETURN          ;
        alternate:entrance:name
```

194: 77, 81

```
scale ::= number
```

195: 63

```
sets:directive ::= !SETS  data:name    ;
```

196: 18, 130

```
shift:function:call ::= SHIFT ( bit:formula ,
numeric:formula )
```

197: 25, 57, 234

```
        letter
sign ::=      numeral
        mark
```

198: 18, 130

signed:function:call ::= SIGNED (numeric:formula)

199: 82

count Z count D . count D
count D . count D
count D .

significand ::= (+ or -)
count D * count D
count * count D
count D count *

200: 130

significand:function:call ::= SIG (numeric:formula)

201: 132

significand:specifier ::= number

202: 130

signum:function:call ::= SIGNUM (numeric:formula)

203: 207

simple:assignment:statement ::= variable = formula ;

204: 175

simple:integer:variable ::= simple:variable

205: 49, 75, 131

simple:item:declaration ::=

environmental:specifier
ITEM item:name
allocation:specifier
item:description packing:specifier

[bit:number] = constant
,

206: 90, 91, 109, 143, 208, 243

simple:item:name ::= name

207: 219

```
        simple:assignment:statement
        assignment:statement
        exchange:statement
        go:to:statement
        exit:statement
        test:statement
   simple:statement  ::= return:statement
        zap:statement
        stop:statement
        loop:statement
        conditional:statement
        switch:statement
        procedure:call:statement
        direct:statement
```

208: 150, 204

```
   simple:variable  ::=  simple:item:name    @
   pointer:formula
```

209: 130

```
        formula
   size:function:call  ::=  SIZE  (             )
        data:block:name
```

210: 132

```
   size:specifier  ::=  number
```

```
211:  63
   skip:directive  ::=  !SKIP    reference    ;


212:  92
   skip:format  ::=  X


213:  63
   space:directive  ::=  !SPACE    character        ;


214:  109
   spacer  ::=  number


215:  216
         null:declaration
   specified:table:body  ::=
   specified:table:item:declaration
         BEGIN specified:table:item:declaration      END    ;


216:  235
   specified:table:declaration  ::=  specified:table:heading
   specified:table:body
```

217: 216

```
specified:table:heading ::=

      environmental:specifier
TABLE table:name
      allocation:specifier

      : allocation:increment    dimension:list

      structure:specifier

      words:per:entry
      bits:per:entry        bit:number    entries:per:word

      packingspecifier        item:description
      packing:specifier

      [ bit:number    , word:number    ]      =
      constant:list    ;
```

218: 131, 215

```
specified:table:item:declaration ::=

   ITEM item:name      item:description

      packing:specifier    [ bit:number

      , word:number     ]    = constant    ;
```

219: 36, 45, 112, 149, 179, 232

```
      null:statement
statement ::=        simple:statement
      compound:statement
      named:statement
```

220: 5, 38, 71, 90, 104, 138, 146, 149, 232

```
statement:name  ::=  name
```

221: 187, 222, 233

```
        primitive
status  ::=   name
        letter
```

222: 157, 223

```
status:constant  ::=  V(  status  )
```

223: 132, 224

```
status:list  ::=
       +                 +
   [            number   ]       status:constant        [
   number  ]  status:constant
       -                 -
```

224: 54

```
status:list:declaration  ::=  STATUS  status:list:name
status:list  ;
```

225: 132, 187, 224

```
status:list:name  ::=  name
```

```
226:   207

   stop:statement  ::=  STOP  ;



227:   166, 217

          P
   structure:specifier  ::=
          T



228:   164, 168

   subordinate:overlay:declaration  ::=  OVERLAY
   subordinate:overlay:expression  ;



229:   231

          item:name
   subordinate:overlay:element  ::=
          (  subordinate:overlay:expression  )



230:   228, 229

   subordinate:overlay:expression  ::=

      subordinate:overlay:string    :
      subordinate:overlay:string
```

231: 230

subordinate:overlay:string ::=
subordinate:overlay:element

232: 207

switch:statement ::=

SWITCH numeric:formula ; statement:name :
+
BEGIN [number] statement
, END ;
-

233:

primitive
ideogram
name
letter:control:variable
symbol ::= abbreviation
number
constant
comment
directive:key
status

234: 25

system:dependent:character

Most computer systems can read and write more characters than are encompassed in the set of JOVIAL sign. The entire set that can be handled is know as the set of characters. The characters that are not signs are known as system:dependent:characters.

235: 49, 51, 75

ordinary:table:declaration
table:declaration ::=
specified:table:declaration

236: 5, 52, 67, 90, 109, 118, 138, 155, 166, 187, 217, 251, 252

table:name ::= name

237: 117
table:variable ::= item:name [index] @
pointer:formula

238: 43

WHILE
conditional:formula
terminator:phrase ::= UNTIL

value:terminator

239: 207

test:statement ::= TEST control:variable ;

240: 63

time:directive ::= !TIME character ;

241: 63

```
trace:directive ::= !TRACE ( conditional:formula )
name ;
```

242: 130

```
type:function:call ::= TYPE ( formula )
```

243: 61

```
number
upper:bound ::=
simple:item:name
```

244: 63

```
uses:directive ::= !USES data:name ;
```

245: 97, 192, 246

```
value:formula ::= [ formula ]
```

246: 238
value:terminator ::=

WHILE value:formula relational:operator variable
UNTIL variable relational:operator value:formula

247: 5, 6, 14, 70, 203, 246

pointer:variable
numeric:variable
variable ::= bit:variable
character:variable
format:variable

248: 5, 6, 14, 70, 203, 246

named:variable
variable ::= letter:control:variable
functional:variable

249: 217, 218

word:number ::= number

250: 217

words:per:entry ::= number

```
251:   130

   words:per:entry:function:call  ::=  NWDSEN  (  table:name
   )



252:   207

         table:name
   zap:statement  ::=  ZAP        ;
         entry:variable
```

**********TABLES**********

Column	0	1	2	3	4	5
6 7	8	9	10	11	12	13
14 15						
Code	0	1	2	3	4	5
6 7	8	9	A	B	C	D
E F						
Row						
0		space	0	@	P	
p						
1		!	1	A	Q	a
q						
2		"	2	B	R	b
r						
3		#	3	C	S	c
s						
4		$	4	D	T	d
t						
5		%	5	E	U	e
u						
6		&	6	F	V	f
v						
7		'	7	G	W	g
w						
8		(	8	H	X	h
x						
9		)	9	I	Y	i
y						
10		*	:	J	Z	j
z						
11		+	;	K	[	k
12		,	<	L	\	l
13		-	=	M	]	m
14		.	>	N		n
15		/	?	O		o

Notes: row 0, column 3: zero
row 1, column 3: one
row 7, column 2: prime, often rendered as a vertical mark in JOVIAL
row 12, column 6: a lowercase letter
row 15, column 4: an uppercase letter

Figure 2-1. Characters

fixed:constant		value	size	precision
19A0	19	5	0	
19A3	19	8	3	
19A-2	16	3	-2	
2,3A0	2	2	0	
2,3A-1	2	1	-1	
2,3A2	2,25	4	2	
2,3A5	2,28125	7		5
2,3A6	2,296875	8		6

Left symbol ends in	Right symbol starts with: numeral	letter	$	'	,	"
numeral		SR		SR	SR	SR
letter		SR		SR	SR	SR
$	SR		SR	SR	SR	
'	SR		SR	SR	SR	
,	SR		SR			
"						

SR

	Value of the conditional:formula 0	1
IF	Skip the controlled:statement following this conditional:formula, Execute the controlled:statement following the matching ELSE	Execute the following controlled:statement then skip the controlled:statement immediately following the matching ELSE if there is one, if there is one,
UNTIL	Execute the controlled:statement, Go on to the next control:clause	or exit the loop if there is

no further control clause.

WHILE Go on to the next contol clause Execute the controlled statement,
or exit the loop if there
is no further control clause.

bit formula	110	01011100	01010111
10000101	01111100		
padded	00000110	01011100	01010111
10000101	01111100		
selected	01011100	01010111	
10000101			

x	y	x\y	x	y	x\y
7	0	undefined	-3,7	2	0,3
1	2	1	4,6	1,5	0,1
2	2	0	-0,1	1,5	1,4
3	2	1	1	-2	-1
-3	2	1	3,7	-2	-0,3
3,7	2	1,7	-3,7	-2	-1,7

First or Only Character	Meaning
I	Number of integer bits
A	Number of fraction bits (bits after the point).
Z	Maximum size I+A the system normally allows for fixed and integer arithemetic.
Y	An even larger maximum size I+A allowed under evaluation:control (often about 2*z).
V	Value.

Second Character	Meaning: "Of the..."
1	First operand
2	Second operand
M	Modulus (for x\y).
N	Numerator (for x/y or x\y).
D	Denominator (for x/y).
I	Integer operand (if the other is fixed).
A	Fixed operand (if the other is integer).
R	Result (preliminary result if S exists).
S	Result required by evaluation:control.
B	Base in exponentation.

E Exponent.

Value of original bit:formula	Value of numeric:formula	Value of resulting from SHIFT
11111	3	11000
11111	-1,7	01111
00000100000	5	10000000000
00000100000	-3	00000000100
101	3	000
101	-3	000
101	-2	001

bit:formula	&	bit:formula	result
10	1	101	
111001	00011110101	11100100011110101	
00010000	0000010	000100000000010	
0	0	00	

p	q	NOT p	p OR q	p EQV q	p AND q	P XOR q
0	0	1	0	1	0	0
0	1	1	1	0	0	1
1	0	0	1	0	0	1
1	1	0	1	1	1	0

```
0     = (assignment)  == (exchange)
1     EQV    XOR
2     OR
3     AND      (logical)
4     NOT
5     =  <  >  <=  >=  <>      (relational)
6     &
7     + -
8     ' / \    ( with or without evaluation:control )
9     **
10    indexing @       ( pointing, evaluation:control )
      @@       ( attribute:association )
```

In the algorithm it is necessary to consider several operands and operations simultaneously. The following diagram shows the relationships. All are pegged in relation to the present operand. Any operation may replace #.

A # B # C # D # E # F # G # H # I # J # K

the next operation

the prior operand the next operand

the prior operation the current operation

the present operand

The leftmost operand of the formula is initally the present operand.

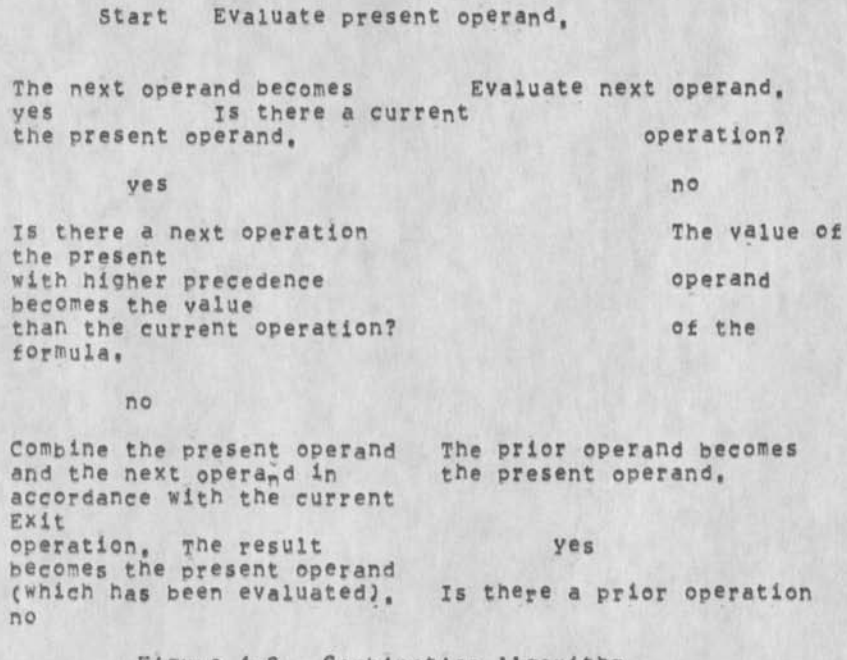

Figure 4-2. Combination Algorithm

ABC Operation type	XYZ type	Char	Bit	Int	Fix	Float
		Converted to				
ABC assignment (also parameter matching and exchange, both ways)	XYZ Char	Char	Bit	Bit	Bit	Bit
	Bit	Bit	Bit	Bit	Bit	Bit
	Int	Bit	Int	Int	Int	Int
	Fix	Bit	Bit	Fix	Fix	Fix
	Float Bit	Bit	Float	Float	Float	
ABC arithmetic XYZ	Float	Note 1	Note 2	Float	Float	Float
XYZ arithmetic ABC	Other	Note 3	Note 4	Scale	Scale	Float
ABC relational XYZ	Char	Note 5	Int	Note 6	Note 6	Note 6
	Bit Note 6	Int	Note 6	Note 6	Note 6	
or	Int Note 6	Int	Int	Scale	Float	
XYZ relational ABC	Float	Note 6	Int	Float	Float	Float
ABC & XYZ	Char	Char	Bit	Bit	Bit	Bit
XYZ & ABC	Other	Bit	Bit	Bit	Bit	Bit
ABC logical XYZ	Any	Bit	Bit	Bit	Bit	Bit
XYZ logical ABC						
Indexing, pointing		Note 7	Int	Int	Int	Int

Figure 4-3, Type Conversion

Entrance used	ALT (procedure:name)
integer	status:constant
normal 0	V (procedure:name)

```
first alternate          1          V ( first
alternate:entrance:name )
second alternate         2          V ( second
alternate:entrance:name )
etc.     etc.                       etc.
```

```
  parameter   type:function:call
qualified:status:constant
     type              value   status:constant

bit:formula            0              V(BIT)
V(TYPE:BIT)

integer:formula                1              V(INT)
V(TYPE:INT)
(signed or unsigned)
fixed:formula          2              V(FIX)
V(TYPE:FIX)
(signed or unsigned)
floating:formula               3              V(FLOAT)
V(TYPE:FLOAT)

character:formula              4              V(BYTE)
V(TYPE:BYTE)
```

```
START


          0-- skip A1, do A2
  A?       E=3

     1-- do A1

          0-- skip B1              0-- skip D1, do D2
(NULL
  B?                          D?
              do B2                          or none),
skip A2

     1-- do B1                   1-- do D1

                                        Skip D2 (if any
and A2
                                 E=2

          0-- skip C1, do C2 (NULL or none), skip B2
and A2
  C?


     1-- do C1

           Skip C2 (if any), B2 and A2
E=1        EXIT
```

	No terminator:phrase	Terminator:phrase
No initial:phrase, replacement:phrase, or increment:phrase	1A, Leave control:variable alone, execute controlled:statement just once, onterminator:phrase,	1B, Same as 1A except execute controlled:statement zero or one time depending
Initial:phrase only	2A, Initialize control:variable, Execute controlled:statement just once,	2B, Same as 2A except execute controlled:statement zero or one time depending on terminator:phrase,
Replacement:phrase only	3A, Leave control:variable alone for the first execution, Before each subsequent execution of the controlled:statement, replace the value of thecontrol:variable,	3B, Same as 3A except test in accordance with terminator:phrase before every execution of the controlled:statement -- even the first one, Repeat executions "forever",
Increment:phrase only	4A, Same as 3A except add to value of control:variable instead of replacing value,	4B, Same as 4A except test as in 3B,
Initial:phrase and replacement:phrase	5A, Initialize control:variable, Execute controlled:statement once, Replace value of control:variable before each subsequent execution, Repeat executions "forever",	5B, Same as 5A except check for termination before each execution,
Initial:phrase and increment:phrase	6A, Initialize control:variable, Execute controlled:statement once, Add to value of control:variable before each subsequent execution,	6B, Same as 6A, except check for termination before each execution,

Repeat executions "forever",

```
                1
start           ALPHA           Set BETA to 3
          =
          ?     2 or 3    Set GAMMA to the
                          value of BETA

                4         If GAMMA equals 2
                          set BETA to 2

                6         Set BETA to the
                          value of GAMMA



                          Set ALPHA to 7  next

                <1, 5 or >6
                          Undefined
```

```
                                                    Format;Lists

Input Buffer  Field              ,,10C              10C,,

28,3b'bABCD'bABb                 1          '28,3bb'
'28,3b$27bABC'
              2                  'bABCDb'   'D$27bbbb'
              3                  'ABbbbbbbbb'   'ABbbbb'

ALPHABET'AbTHERMOPILEb                      1       'ALPHAb'
'ALPHAbBET$27'
              2                  'BET$27Ab'     'Abbbbb'
              3                  'THERMOPILE'   'THERMO'
```

```
      4B3PS3pS3PS3p   5B5PS5P

Bah!  000 042 616 821 00011 62Q11
Humbug        487 56D 627 567 28ELM M4TB7



Input Buffer  Field   5N, 6N, SNSN

1.2E3-b485bb3b7               1        1.2E3
              2        -485
              3         37

+46b7.00015A1B.               1        illegal field
(contains blank)
              2        .00015
              3        1.



      +SD4D-3ZDDR     -SDSZSZSZ       +SD","ZZ","DDR

  1573.64      + 1573   1574   1 5 7 3          + 15.74
  - 27         - 0027    -27  -2 7          - 27
    0.0          0000     00   0              00
-10740         undefined      -10740  undefined       -
1.07.40



                                  Format:Lists
Input Buffer  Field   "SPEEDb",DDD,"MPH       ,DDDSSSS

SPEEDb100bMPH         1               100     'SPEED'
              2                               100
```

```
      +ZZDD.DZZ        -4Z*3DR -4Z*     -4Z2*R

 1573.6405     +1573.64          1573641 1573      157
  -27    -27.0  -27000  -27       -3
    0.0            00.0     000
-10740          undefined        undefined        undefined
-1074
```

```
      1573.6400        1573.6410        1573.0000
1570.0000
     -27      -27      -27      -30
   0.0              0.0      0.0      0.0
     undefined        undefined        undefined
-10740
```

```
      +.6DE+3ZR        -SD*6ZSES-3Z*    -S3D.SES-3DR
+S3*5DSES+S3Z

+ 39.7528      +.397528E+2        397528  E   1   398. E
-001   + 39752 E + 4
-.008711246    -.871125E-2      - 8711246 E  -3 - 871. E
-005   - 87112
```

```
PROGRAM AA
   XX (table;name)

      PROC BB
         XX (item;name)

            PROC CC
               no occurrence of XX

                  PROC DD
                  XX used but not declared



                  PROC EE
```

Figure 7-1. Scope of Names

Serial Structure	Parallel Structure
1st half AB[0]	1st half AB[0]
2d half AB[0]	1st half AB[1]
XY[0]	1st half AB[2]
1st half AB[1]	1st half AB[3]
2d half AB[1]	2d half AB[0]
XY[1]	2d half AB[1]
1st half AB[2]	2d half AB[2]
2d half AB[2]	2d half AB[3]
XY[2]	XY[0]
1st half AB[3]	XY[1]
2d half AB[3]	XY[2]
XY[3]	XY[3]

Example: Table MN has 2 items, AB and XY, and 4 entries, 0, 1, 2, and 3.
Item AB occupies 2 words.
Item XY occupies 1 word.
Note: 12 consecutive computer words are shown in each illustration above.

Figure 7-2. Serial and Parallel Table Structure

Tight Structure

entry [0]	entry [1]	entry [2]
entry [3]	entry [4]	entry [5]

A table of six entries is medium packed, three entries to the word.

```
OVERLAY AA, AB, AC : BA, (BX : BY, BZ), BC ;

  AC              AA                AB

   BA               BX                BC

                   BY    BZ
```

```
RELATED STRUCTURE          AA,100,(BB : EE)          CC,EE,DD
200,FF,GG : DD

            *
CC          AA                    AA                CC

 100                  100
                                    *
EE          BB                    BB
                       EE
                                                      *
DD                                                    DD

 200                                                  200

   FF                                                 FF

   GG                                                 GG
```

Figure 7-4. Allocation of a Related Structure

```
entrance        number          status:constant

normal                  0        V( procedure:name )
first alternate                  1       V(
alternate:entrance:name )
second alternate                 2       V(
alternate:entrance:name )
```

RELATIVE WORD	SERIAL		PARALLEL	
0	1	AB[0,0,0]	1	AB[0,0,0]
1	2	AB[0,0,0]	1	AB[0,0,1]
2		XY[0,0,0]	1	AB[0,1,0]
3	1	AB[0,0,1]	1	AB[0,1,1]
4	2	AB[0,0,1]	1	AB[0,2,0]
5		XY[0,0,1]	1	AB[0,2,1]
6	1	AB[0,1,0]	1	AB[0,3,0]
7	2	AB[0,1,0	1	AB[0,3,1]
8		XY[0,1,0]	1	AB[1,0,0]
9	1	AB[0,1,1]	1	AB[1,0,1]
10	2	AB[0,1,1]	1	AB[1,1,0]
11		XY[0,1,1]	1	AB[1,1,1]
12	1	AB[0,2,0]	1	AB[1,2,0]
13	2	AB[0,2,0]	1	AB[1,2,1]
14		XY[0,2,0]	1	AB[1,3,0]
15	1	AB[0,2,1]	1	AB[1,3,1]
16	2	AB[0,2,1]	1	AB[2,0,0]
17		XY[0,2,1]	1	AB[2,0,1]
18	1	AB[0,3,0]	1	AB[2,1,0]
19	2	AB[0,3,0]	1	AB[2,1,1]
20		XY[0,3,0]	1	AB[2,2,0]
21	1	AB[0,3,1]	1	AB[2,2,1]
22	2	AB[0,3,1]	1	AB[2,3,0]
23		XY[0,3,1]	1	AB[2,3,1]
24	1	AB[1,0,0]	2	AB[0,0,0]
25	2	AB[1,0,0]	2	AB[0,0,1]
.		.	.	
.		.	.	
.		.	.	
62		XY[2,2,0]		XY[1,3,0]
63	1	AB[2,2,1]		XY[1,3,1]
64	2	AB[2,2,1]		XY[2,0,0]
65		XY[2,2,1]		XY[2,0,1]
66	1	AB[2,3,0]		XY[2,1,0]
67	2	AB[2,3,0]		XY[2,1,1]
68		XY[2,3,0]		XY[2,2,0]
69	1	AB[2,3,1]		XY[2,2,1]
70	2	AB[2,3,1]		XY[2,3,0]
71		XY[2,3,1]		XY[2,3,1]

Figure 10-1 Indexing and Storage Allocation

		Bits				Bits
0-1	2-11	12-21	22-31	0-1	2-11	12-21
22-31						
0		BB[0,0]	BB[0,1]	bB[0,2]		BB[0,0]
BB[0,1]	BB[0,2]					
1		BB[0,3]	BB[0,4]	BB[0,5]		BB[0,3]
BB[0,4]	BB[0,5]					
2		BB[0,6]	BB[0,7]			BB[0,6]
BB[0,7]	BB[1,0]					
3		BB[1,0]	BB[1,1]	BB[1,2]		BB[1,1]
BB[1,2]	BB[1,3]					
4		BB[1,3]	BB[1,4]	BB[1,5]		BB[1,4]
BB[1,5]	BB[1,6]					
5		BB[1,6]	BB[1,7]			BB[1,7]
BB[2,0]	BB[2,1]					
6		BB[2,0]	BB[2,1]	BB[2,2]		BB[2,2]
BB[2,3]	BB[2,4]					
7		BB[2,3]	BB[2,4]	BB[2,5]		BB[2,5]
BB[2,6]	BB[2,7]					
8		BB[2,6]	BB[2,7]			BB[3,0]
BB[3,1]	BB[3,2]					
9		BB[3,0]	BB[3,1]	BB[3,2]		BB[3,3]
BB[3,4]	BB[3,5]					
10		BB[3,3]	BB[3,4]	BB[3,5]		BB[3,6]
BB[3,7]	BB[4,0]					
11		BB[3,6]	BB[3,7]			BB[4,1]
BB[4,2]	BB[4,3]					
12		BB[4,0]	BB[4,1]	BB[4,2]		BB[4,4]
BB[4,5]	BB[4,6]					
13		BB[4,3]	BB[4,4]	BB[4,5]		BB[4,7]
BB[5,0]	BB[5,1]					
14		BB[4,6]	BB[4,7]			BB[5,2]
BB[5,3]	BB[5,4]					
15		BB[5,0]	BB[5,1]	BB[5,2]		BB[5,5]
BB[5,6]	BB[5,7]					
16		BB[5,3]	BB[5,4]	BB[5,5]		
17		BB[5,6]	BB[5,7]			

Figure 10-2 Indexing and Allocating Tight Structure Tables

NSW; Components, Tools and Senerio of Use

(J31099) 26-SEP-74 16:04;;;; Title: Author(s): Duane L. Stone/DLS;
Distribution: /ARB([INFO-ONLY]) RDK([INFO-ONLY]) FJT([
INFO-ONLY]) JLM([INFO-ONLY]) MAW([INFO-ONLY]) ;
Sub-Collections: RADC; Clerk: DLS;

DLS 26-SEP-74 16:04 31099

NSW; Components, Tools and Senerio of Use

A draft of my view of the NSW, with reasons why the Air Force should support it.

NSW; Components, Tools and Senerio of Use

BACKGROUND 1

The priniciple short term way in which we can improve the current problems of unreliable and costly software systems, is to support programmers and their management with modern methods and tools....ref. Monterey Study. 1a

Experience, methods and tools to support modern programming practices exist within many R&D environments around the ARPANET, but unfortunately seldom on the machines for which AF systems are being developed. 1b

The NSW will allow AF (and eventually AF contractor) programmers and their managers to use these tools where they exist, in a straightforward manner, without a costly capital investment for each system. 1c

NSW 2

The NSW represents the first concerted effort to accomplish the primary goal for which the ARPANET was initially constructed, ie resource sharing. It consists of four components, none of which can be removed with out reducing the whole thing to ashes. 2a

The network exec or the "works manager", 2a1

the detailed implementor or "foreman" 2a2

the user interface or "front end", 2a3

and the tools themselves. 2a4

WORKS MANAGER--MCA EFFORT 3

MCA will develop the network executive or "Works Manager". It will keep track of the NSW files, where they are stored, their versions, who last updated them etc. It will be responsible for knowing who has access to which tools and which files. It will keep track of tool useage and do the accounting and billing. In the case of tools that reside on more than one host, it will determine loading and connect the user to the least loaded host. 3a

FOREMAN--MCA EFFORT 4

MCA will develop the software which performs the detailed login into individual hosts, moves files from one host to another, etc. In general, the Foreman is responsible for seeing that the commands given by the Manager are carried out, and if they are aborted, reports the status back to the Manager. 4a

NSW; Components, Tools and Senerio of Use

FRONT END--SRI EFFORT 5

SRI will develop the NSW "front end" system, which will provide a coherent user interface to the NSW and hence to the tools. 5a

The front-end will reside in a PDP-11, running the ELF time-sharing executive. It will contain the grammer and user profile for the particular tool requested by the user. It will perform the terminal mapping functions of a TIP, as well as the command feedback functions of the program being used at the moment. 5a1

SRI will develop the NSW protocols which allow tools to be interfaced with the works manager, and also ARPANET protocols which allow file transfer etc. to be accomplished in a more efficient manner. 5b

TOOLS--INITIAL 6

NLS 6a

SRI will reconfigure NLS, so that it can be interfaced with the NSW; ie, the division into a front-end and a back-end. The NLS front-end will be written in the Command Meta Language (CML), compiled on the PDP-10X and shipped over the ARPANET to the PDP-11. The front end will contain the command language parsing, user feedback and terminal support functions. The back-end (executable procedures) will reside on one or more PDP-10Xs. This is being done for two purposes: 6a1

to develop the protocols for interfacing tools in general and their command languages to the NSW, 6a1a

to cut the cost of NLS use in half, 6a1b

Sri will add to the capability of NLS, to make it a more complete documentation system. In particular they will 6a2

modify the output processor to improve its ease of use and to generate code acceptable to the Linotron at WPAFB, 6a2a

add an elementary line drawing capability. 6a2b

SRI will add features designed specifically to support COBOL programmers operating in the RJE mode. 6a3

B-4700 6b

The priniciple tool residing on the B-4700 at Gunter AFB will

be a COBOL compiler. It will be used by AFDSDC programmers to develop software for the B-3500s. 6b1

LINTRON 6c

The Linotron at WPAFB is capable of generating quality output in several fonts and styles. Experimentation with a commercial COM organization indicates that this is desireable when a document is widely distributed and updated infrequently. The total process of publishing documents in this manner is less costly than conventional typesetting, however the particular commercial COM service now in use is unduly expensive. The Linotron at WPAFB is government owned, underused and cheaper to use, since it goes directly to paper with out the intermediate step of microfilm. 6c1

DATACOMPUTER 6d

The data computer is a mass storeage divice (a trillion bits) accessed via a data management system on a PDP-10. It will be used as the main backup storage device for the NSW. 6d1

SENERIO FOR NSW USE 7

The following is my understanding of a "typical" senerio of how the NSW is initially expected to be used. 7a

A programmer at AFDSDC in Alabama would use terminals located in his working environment, connected to a PDP-11, connected to an TIP or IMP, to log into the NSW computer. After being recognized as a legitimate NSW user, he would indicate which tool he wished to use. 7a1

Let us assume that he wants to work further on a COBOL program he has been writing. He would give the works manager a command to "get me some NLS". 7a2

After determining which machine the user should be connected to, the works manager gives the command to the foreman "Connect this terminal to host XYZ and log him into NLS". The foreman makes the network connection, logs into TENEX and starts the NLS subsystem running. While this is happening, the works manager sends the NLS grammer and user profile to the PDP-11, to set up the NLS front-end tailored to that particular user. The NSW machine invokes a dynamic reconnect and takes itself out of the loop, so that the front-end and back-end have a direct network connection. 7a3

During the course of the session in NLS, the user can access

others files, copy code into his program, edit, rearrange, get special views, etc. He might have need to refer to a program specification document. He would obtain it by saying Load File and giving the directory and file names. If it was not on the local host, a signal would be sent to the NSW host, which would consult its catalogues, determine that it resided at the Data Computer, issue a command to the foreman to retrieve it and send a copy to the user's current host; probably flagged as read only. 7a4

Presuming that the user had the program written to the best of his ability, he would invoke an NLS user program which would make preliminary syntax checks, correct misspelled reserved words, and notify the user of reserved words not declared. After the user had corrected these errors he would invoke a second user program that would insert the necessary "control cards", strip the indentation associated with NLS, prepare a file in a format suitable for entry to the B-4700 via RJE and notify the works manager of its existence. 7a5

The user would then contact the works manager and request the B-4700 COBOL compiler tool. He would specify the file to be compiled, whether or not he wished to be notified when the job finished or aborted and the file name of any results returned from the compilation. The works manager would issue the appropriate commands to the foreman, who would see that the file was moved from the NLS machine into the RJE queue on the B-4700 disc. 7a6

In doing so, it will talk with the B-4700 NCP, which is resident in the pDp-11. The pDp-11 will communicate with the B-4700 by simulating a device the B-4700 operating system knows (initially a tape drive). 7a6a

The operator would be notified of its existance, and schedule it according to the procedures used within the B-4700 facility. 7a7

Upon logging out of the NSW, the Works Manager would request charges from the hosts used during that session, add them to the user's account and at the end of the month, prepare and send him a bill. 7a7a

In a similar manner, the user could instruct the NSW to archive the object code file, in which case it would be moved from the B-4700 to the Data Computer. 7a8

The same general techniques would be used to access any tool available via the NSW. A document could be partially prepared on different machines, using editors like QED, TECO, NLS, etc.

It could be merged under NLS, reformated, draft copies printed at each coordinators' site, changes incorporated, output processor directives inserted, a sequential file created, moved to the tape unit on the WPAFB TIP, carried to the Linotron, photocomposed, multiple copies printed and distributed to recipients. This would all be accomplished through the facilities and under the control of NSW. It can be accomplished now, but an individual would have to know all the idiosyncracies of each tool; their login procedures, their operating systems, the command languages,,syntax and grammer,,of their user programms, file compression and transfer protocols, etc,, etc,, etc. 7a9

REASONS FOR AIR FORCE SUPPORT OF NSW 8

AIR FORCE in General 8a

Direct coupling of R&D and using commands in the computer system development area. 8a1

lower cost, more reliable software, which is more responsive to the user's needs. 8a1a

shortened development-to-applications cycle, 8a1b

better use of R&D computer expertize. 8a1c

Mechanism whereby commericially developed (as well as AF sponsored) tools can be placed in a competitive marketplace for use and evaluation, 8a2

Minimize the replication and transferal of useful tools to many machines and development sites. 8a3

Using commands 8b

Initially AFDSDC 8b1

Access to NLS, the most powerful documentation and programming suport system in the country today!!! 8b1a

ability to write COBOL source code, edit it, have it checked for misspellings etc, and formated for entry into the RJE system on the B-4700. 8b1a1

ability to maintain all source code listings, system documentation, management data, etc;cross index them, retrieve them, modify them quickly,, 8b1a2

ability to publish documents via the Linotron, 8b1a3

Soon WWMCCS and others 8b2

besides the above capabilities; the WWMCCS community and other AF users on the ARPANET will have access to RADC developed tools 8b2a

Jovial compilers 8b2a1

preprocesors to support structured programming 8b2a2

test tools 8b2a3

and any other tools we care to make available via the NSW, 8b2a4

RADC 8c

vehicle for "exporting" tools we develop in a meaningful way, 8c1

direct involvement in the development of a prototype operational system for AFDSDC 8c2

supported 50% by General Robbins out of O&M money, 8c2a

supported 25% by ARPA, 8c2b

supported 25% by RADC, 8c2c

in on the ground floor of the first major ARPANET development effort aimed specifically at its initial goal of resouce sharing, 8c3

We can apply the knowledge gained to: 8c3a

AFSC network 8c3a1

WWMCCS network 8c3a2

. 8c3a3

. 8c3a4

. 8c3a5

DLS 26-SEP-74 16:17 31100

3rd COM Test run...Equations and Tables

(J31100) 26-SEP-74 16:17;;;; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC([INFO-ONLY]) ; Sub-Collections: RADC; Clerk:
DLS;

DLS 26-SEP-74 16:17 31100

3rd COM Test run...Equations and Tables

Contains revised Tabstops to line up equations and tables better. This has not been edited for mistakes. It contains directives for overall format and type styles which are good.

3rd COM Test run,,,Equations and Tables

**********EQUATIONS**********

Appendix A

SYNTAX EQUATIONS

The following pages contain the complete syntactic description of JOVIAL (J73). The metalinguistic equations are in alphabetical order of the metalinguistic terms being defined. In general, each defining equation is individually boxed. The boxes are nubered sequentially in the upper left hand corner by a number in italics followed by a colon. Following the colon is a list of the box numbers in which the current term is used as a part of the definitions of other terms. The metalinguistic symbology is explained in section 1.4.

Equations 94 and 95 are in one box. These are both valid and necessary definitions for format:list. Equation 144 is the definition for mark. In the same box, opposite each mark is a metalinguistic term (or two). These marks constitute the definitions of these terms. Equation 172 defines pattern:digit. In the same box is information giving the bit patter corresponding to each pattern:digit, depending on the order of the pattern:constant. In the box with equation 190, the definition of relational:operator, is a list of the meanings of the relational:operators. Box 234 contains a definition for system:dependent:character, but the definition is a prose description; a metalinguisitic equation is not feasible. Equations 247 and 248 are in one box. They are both valid and necessary definitions for variable.

1: 233

abbreviation ::= letter

2: 63

abnormal:directive ::= !ABNORMAL data:name ;

```
3:     130
   absolute:function:call  ::=  ABS  (  numeric:formula  )


4:    58, 170
            definition
   actual:define:parameter  ::=
         " definition "


5:     101, 170, 180, 191
         STOP
                  alternate:entrance:name
         RETURN
                  procedure:name
         TEST     control:variable

         EXIT     statement:name
   actual:input:parameter  ::=
         statement:name
         procedure:name
         formula
         table:name
         data:block:name
         variable
         @  pointer:formula


6:     170, 180, 191
   actual:output:parameter  ::=  variable
```

7: 166,217

allocation:increment ::= number

8: 9, 49, 166, 182, 205, 217

allocation:specifier ::= @ pointer:formula

9: 75, 184

alternate:entrance:declaration ::=

ENTER alternate:entrance:name

(formal:input:parameter

: formal:output:parameter)

environmental:specifier item:description
allocation:specifier

packing:specifier [bit:number]

= + constant ;
-

10: 130

alternate:entrance:function:call ::= ALT (procedure:name)

11: 5, 9, 53, 101, 122, 138, 180, 187, 193

alternate:entrance:name ::= name

```
12:    159

          +
          -
          *
   arithmetic:operator  ::=
          /
          \
          **


13:

   assignment:operator  ::=    =


14:    207

   assignment:statement  ::=
                  formula
          variable                indexed:variable:range  ;
          indexed:variable:range =
                  format:function:call

                  formula
      format:variable    =                 ;
                  indexed:variable:range


15:    159

   attribute:association  ::=  @@  [  description:attibute  ]
```

16: 63

begin:directive ::= !BEGIN reference ;

17: 18

bit:form ::= form

18: 18, 29, 97, 159, 196

pattern:constant
entry:variable
comparison
chain:comparison
bit:string:function:call
shift:function:call
bit:formula ::= bit:form
bit:formula logical:operator bit:formula
NOT bit:formula
bit:formula & bit:formula
(bit:formula)
numeric:formula
character:formula

19: 9, 182, 205, 217, 218

bit:number ::= number

20: 18, 130

bit:string:function:call ::=

BIT (formula , numeric:formula ,
numeric:formula)

21: 247

bit:variable ::=

entry:variable
BIT (named:variable , numeric:formula ,
numeric:formula)

22: 217

bits:per:entry ::= number

23: 130

byte:string:function:call ::=

BYTE (character:formula , numeric:formula ,
numeric:formula)

24: 18

chain:comparison ::= comparison relation:operator
formula

25: 26, 32, 46, 62, 100, 112, 120, 137, 213, 234, 240

sign
character ::=
system:dependent:character

```
26:   29, 39

   character:constant  ::=   count  '  character  '


27:   29

   character:form  ::=  form


28:   29

   character:format  ::=     count   C


29:   18, 23, 93, 94, 97

         character:constant
         character:variable
         character:form
   character:formula  ::=        character:function:call
         character:formula  &  character:formula
         (  character:formula  )
         bit:formula


30:   29

   character:function:call  ::=  function:call
```

31: 29, 96, 247

```
        named:character:variable
character:variable ::=      BYTE  (
named:character:variable  ,  numeric:formula

                  ,  numeric:formula    )
```

32: 233

```
comment ::= "  character      "
```

33: 18, 24

```
comparison ::= formula  relational:operator  formula
```

34: 63

```
compool:directive ::=
                  name
       compool:name
   !COMPOOL           (  name  )      ;

                 (  compool:name  )
```

35: 34

```
compool:name ::= name
```

36: 219

```
        declaration
compound:statement ::= BEGIN                END ;
        statement
```

37: 38, 97, 238, 241

```
conditional:formula ::= formula
```

38: 207

```
conditional:statement ::=

   IF conditional:formula ; controlled:statement

          statement:name ;    ELSE
      controlled:statement
```

39: 9, 42, 182, 205, 233

```
        numeric:constant
constant ::= pattern:constant
        character:constant
```

40: 97

```
constant:formula ::= ( formula )
```

41: 166, 167, 217, 218

```
constant:list ::=

      [ index ]
    [ index ]    constant:list:element
   constant:list:element
            ,
```

42: 41, 42

```
          ,     +     constant     ,
constant:list:element ::=                 -
      count ( constant:list:element )
      constant:list:element
```

43: 136, 148

```
        increment:phrase          terminator:phrase
        replacement:phrase
control:clause ::=      initial:phase
                increment:phrase
        terminator:phrase         replacement:phrase
```

44: 5, 239

```
        named:variable
control:variable ::=
        letter:control:variable
```

45: 38, 141

```
controlled:statement ::= statement
```

46: 63

```
copy:directive ::= COPY character ;
```

47: 26, 28, 42, 72, 78, 95, 98, 103, 120, 124, 126, 171, 173, 199

```
count ::= number
```

48: 182

```
data:allocator:specifier ::= @
```

49: 49, 51, 75

```
data:block:declaration ::=

                environmental:specifier
   BLOCK  data:block:name                    ;
                allocation:specifier

                simple:item:declaration
                table:declaration
      BEGIN                        END     ;
                data:block:declaration
                independent:overlay:declaration
```

50: 5, 49, 52, 90, 109, 138, 209

```
data:block:name ::= name
```

```
51:    54

          item:declaration
   data:declaration  ::= table:declaration
          data:block:declaration
          overlay:declaration



52:    2, 129, 174, 195, 244

          item:name
   data:name  ::=        table:name
          data:block:name



53:    130

          procedure:name
   data:size:function:call  ::=  DSIZE  (              )
          alternate:entrance:name



54:    36, 54, 112, 179

          status:list:declaration
          form:declaration
          data:declaration
          null:declaration
   declaration  ::=      define:declaration
          name declaration
          processing:declaration
          external:declaration
          BEGIN declaration      END    ;



55:    54

   define:declaration  ::=
```

```
DEFINE define:name    (  formal:define:parameter    )
"  definition  ";
```

```
56:    55, 58

   define:name  ::=  name
```

```
57:    4, 55

   definition  ::=  sign
```

```
58:

   definition:invocation  ::=  define:name    (
   actual:define:parameter      )
```

```
59:    185

   dependent:program:declaration  ::=  procedure:declaration
```

```
60:    15, 69

         item:name
   description:attribute  ::=
         item:description
```

(equ61)

61: 166, 217

```
dimension:list ::= [      lower:bound : upper:bound
]
```

62: 207

```
direct:statement ::= DIRECT character JOVIAL ;
```

63:

```
        compool:directive
        skip:directive
        begin:directive
        end:directive
        trace:directive
        copy:directive
        abnormal:directive
        sets:directive
directive ::=       uses:directive
        pointer:directive
        order:directive
        recursive:directive
        time:directive
        space:directive
        linkage:directive
        interference:directive
        frequency:directive
```

```
64:    233

          !COMPOOL
          !SKIP
          !BEGIN
          !END
          !TRACE
          !COPY
          !ABNORMAL
          !SETS
   directive:key  ::=    !USES
          !POINTER
          !ORDER
          !RECURSIVE
          !TIME
          !SPACE
          !LINKAGE
          !INTERFERENCE
          !FREQUENCY



65:    63

   end:directive  ::=  !END ;



66:    217

   entries:per:word  ::=  number



67:    18, 21, 117, 252

   entry:variable  ::=  table:name    [    index    ]      @
   pointer:formula
```

68: 9, 49, 166, 182, 205, 217

```
                    program:name
          IN        procedure:name
environmental:specifier  ::=          RESERVE

      RESERVE
```

69: 159

```
eValuation:control  ::=  @  [  description:attribute  ]
```

70: 207

```
exchange:statement  ::=  variable  ==  variable  ;
```

71: 207

```
exit:statement  ::=  EXIT  statement:name  ;
```

72: 82

```
                  count    D      count    Z

      +
exrad  ::=              count    Z      count    D
      -
                  count    Z      *
```

73: 130

```
exrad:function:call ::= XRAD ( numeric:formula )
```

74: 132

```
exrad:specifier ::= number
```

75: 54

```
external:declaration ::=
                simple:item:declaration
                table:declaration
                data:block:declaration
                name:declaration
   DEF                  procedure:declaration
                alternate:entrance:declaration
   REF                  simple:item:declaration
                table:declaration
      BEGIN     data:block:declaration  END    ;
                name:declaration
                procedure:declaration
                alternate:entrance:declaration
```

76: 87

```
field:width ::= number
```

77: 157

```
fixed:constant ::=
   number
```

```
                    E                scale    A
      scale
                    -                         -
      number    .   number
```

```
78:   158
                          *          count    D
                    integer:part
                          .          fraction:part
          +
   fixed:format  ::=                integer:part
   count    *    R
          -
                    count     *
                                     fraction:part
                          .
```

```
79:   160
   fixed:function:call  ::=  function:call
```

```
80:   162
   fixed:variable  ::=  named:variable
```

```
81:   157
   floating:constant  ::=
                +
        number     E    scale
                -
       number      .                                M    +
      scale
```

```
                    +
        E                    scale
                    -
number  ,     number
```

82: 158

floating:format ::= significand E exrad R

83: 162

floating:function:call ::= function:call

84: 162

floating:variable ::= named:variable

85: 141

for:clause ::= FOR loop:control ;

86: 17, 27

form ::= form:name (formula)

87: 54

B

```
   form:declaration  ::=  FORM  form:name
   field:width       ;
         C



88:    86, 87

   form:name  ::=  name



89:    55, 170
   formal:define:parameter  ::=  letter



90:    9, 170, 182

         statement:name
         simple:item:name
   formal:input:parameter  ::=   procedure:name
         table:name
         data:block:name



91:    9, 170, 182

   formal:output:parameter  ::=  simple:item:name



92:    95

         null:format
         insert:format
   format  ::=   skip:format
         character:format
         pattern:format
         numeric:format
```

```
93:    14, 130

   format:function:call  ::=

      FORMAT  (  character:formula    ,      format:list
      ,  procedure:name      )



94:    93, 95, 96

   format:list  ::=  character:formula



95:    93, 95, 96

         format
   format:list  ::=
         count  (  format:list  )



96:    14, 102, 247

   format:variable  ::=

      FORMAT  (  character:variable    ,    format:list
      ,  procedure:name      )
```

```
97:    5, 14, 20, 24, 33, 37, 40, 86, 119, 159, 192, 203,
209, 242, 245

          pointer:formula
          numeric:formula
          bit:formula
   formula  ::=  conditional:formula
          character:formula
          value:formula
          numeric:formula
          constant:formula



98:    78

          count    D          count    Z

   fraction:part  ::=

          count    D



99:    130

   fraction:part:function:call  ::=  FRAC  (  numeric:formula
   )



100:   65

   frequency:directive  ::=  !FREQUENCY  character      ;
```

```
101:  30, 79, 83, 125

        intrinsic:function:call

        procedure:name
      function:call  ::=          @ pointer:formula
        alternate:entrance:name

              (    actual:input parameter        )


102:  248

        format:variable

        BYTE  (  named:character:variable  ,
      numeric:formula

      functional:variable  ::=           , numeric:formula
      )

        BIT  (  named:variable  ,  numeric:formula

              ,  numeric:formula    )


103:  158

      generalized:numeric:formula  ::=     count   N       R


104:  207

      go:to:statement  ::=   GOTO   statement:name   [
      index   ]   ;
```

105: 115

high:point ::= numeric:formula

106: 233

```
+
-
*
/
**
\
&
=
==
<
>
<=
>=
```

ideogram ::= <>

```
'
,
:
;
!
"
#
(
)
[
]
@
@@
```

107: 43

numeric:formula
increment:phrase ::= BY
numeric:value:formula

```
108:   49, 168

   independent:overlay:declaration  ::=

         [ number ]
      OVERLAY            independent:overlay:expression  ;
         [ pattern:constant ]



109:   111

         spacer
         simple:item:name
   independent:overlay:element  ::=      table:name
         data:block:name
         (  independent:overlay:expression  )



110:   108, 109

   independent:overlay:expression  ::=

      independent:overlay:string    |
      independent:overlay:string



111:   110

   independent:overlay:string  ::=
   independent:overlay:element



112:   185

   independent:program:declaration  ::=
         statement
      PROGRAM program:name    ( character    )    ;
         declaration
```

113: 41, 67, 104, 237

index ::= index:component

114: 113, 116

index:component ::= numeric:formula

115: 116

index:component:range ::= low:point : high:point

116: 118, 155

index:component:range
index:range ::=
index:component

117: 150

table:variable
indexed:variable ::=
entry:variable

118: 14

```
indexed:variable:range ::=

        item:name
                        [ index ]      @ pointer:formula
        table:name

        item:name
ALL (                   @ pointer:formula    )
        table:name
```

119: 43

```
initial:phrase ::= formula
```

120: 92

```
      count    S

                numeral
insert:format   ::=    count     /
                letter

      count     "  character      "
```

121: 182

```
instruction:allocation:specifier ::= pointer:formula
```

122: 130

```
      procedure:name
instruction:size:function:call  ::=  ISIZE  (          )
      alternate:entrance:name
```

123: 157, 175

```
integer:constant  ::=  number
```

124: 158

```
                count    Z          count    D
      +
integer:format  ::=                   R
      -
                count    D          count    Z
```

125: 160

```
integer:function:call  ::=  function:call
```

126: 78

```
      count    Z          count    D
integer:part  ::=
      count    D
```

```
127:  130

   integer:part:function:call  ::=  INT  (  numeric:formula )



128:  162

         named:variable
   integer:variable  ::=
         letter:control:variable



129:  63

   interference:directive  ::=  !INTERFERENCE    data:name  !
   data:name                    ;



130:  101

         format:function:call
         byte:string:function:call
         bit:string:function:call
         alternate:entrance:function:call
         number:of:entries:function:call
         location:function:call
         shift:function:call
         absolute:function:call
         words:per:entry:function:call
   intrinsic:function:call  ::=  exrad:function:call
         significand:function:call
         signed:function:call
         signum:function:call
         size:function:call
         type:function:call
         fraction:part:function:call
         integer:part:function:call
         instruction:size:function:call
         data:size:function:call
```

131: 51

simple:item:declaration
item:declaration ::= ordinary:table:item:declaration
specified:table:item:declaration

132: 9, 60, 166, 167, 182, 205, 217, 218

item:description ::=

C size:specifier

F , R significand:specifier ,
exrad:specifier

status:list
S status:list:name
, R size:specifier +
U , precision:specifier
=

133: 52, 60, 118, 167, 187, 205, 218, 229, 237,

item:name ::= name

134: 1, 89, 120, 135, 136, 145, 197, 221

letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

135: 44, 128, 233, 248

letter:control:variable ::= letter

136: 140

letter:loop:control ::= letter (control:clause)

```
137:  63
   linkage:directive  ::=  !LINKAGE    character      ;


138:  130
         statement:name
         named:variable
   location:function:call  ::=  LOC  (   table:name      )
         data:block:name
         procedure:name
         alternate:entrance:name


139:  130
         AND
         OR
   logical:operator  ::=
         EQV
         XOR


140:  85
         named:loop:control
   loop:control  ::=
         letter:loop:control


141:  207
   loop:statement  ::=  for:clause  contolled:statement
```

142: 115

low:point ::= numeric:formula

143: 61

number
lower:bound ::=
simple:item:name

144: 197

	+	plus:sign
	-	minus:sign
	*	asterisk
	/	slash
	\	back:slash
	&	ampersand
	>	greater:than:sign
	<	less:than:sign
	=	equals:sign
	@	at:sign
mark	::=	. decimal:point
	:	colon
	,	comma
	;	semicolon
		space
	(	left:parenthesis, parenthesis
	)	right:parenthesis, parenthesis
	[	left:bracket, bracket
	]	right:bracket, bracket
	'	prime
	"	quotation:mark
	$	dollar:sign
	!	exclamation:point

145: 11, 34, 35, 50, 56, 88, 133, 183, 186, 206, 220, 221, 225, 233, 236, 241

```
                letter
                numeral
      letter    $
name  ::=       '
        $
```

146: 54, 75

```
      statement:name
name:declaration  ::=  NAME          ;
      procedure:name
```

147: 31, 102

named:character:variable ::= named:variable

148: 140

named:loop:control ::= named:variable (control:clause)

149: 219

named:statement ::= statement:name : statement

150: 21, 44, 80, 84, 102, 128, 138, 147, 148

```
        simple:variable
named:variable  ::=
        indexed:variable
```

151: 54, 164, 215

```
        NULL    ;
null:declaration  ::=
        BEGIN    END    ;
```

152: 92

```
null:format  ::=
```

153: 219

```
        NULL    ;
null:declaration  ::=
        BEGIN    END    ;
```

154: 7, 19, 22, 47, 66, 74, 76, 77, 81, 108, 123, 143, 169, 177, 189, 194, 201, 210, 214, 223, 232, 233, 243, 249, 250

```
number  ::=  numeral
```

```
155:   130

   number:of:entries:function:call  ::=  NENT  (  table:name
   (  index:range  )    )



156:   120, 145, 154, 197

          0
          1
          2
          3
   numeral  ::=  4
          5
          6
          7
          8
          9



157:   39, 159

          integer:constant
          fixed:constant
   numeric:constant  ::= floating:constant
          status:constant
          qualified:status:constant



158:   92

          generalized:numeric:format
          integer:format
   numeric:format  ::=
          fixed:format
          floating:format
```

159: 3, 18, 20, 21, 23, 31, 73, 97, 99, 102, 105, 107, 114, 127, 142, 159, 161, 175, 196, 198, 200, 202, 232

numeric:constant
numeric:variable
numeric:function:call
+
numeric:formula
-

numeric:formula ::= numeric:formula arithmetic:operator

evaluation:control numeric:formula
evaluation:control
formula
attribute:association
(numeric:formula)
bit:formula

160: 159

integer:function:call
numeric:function:call ::= fixed:function:call
floating:function:call

161: 97, 107

numeric:value:formula ::= [numeric:formula]

162: 159, 176, 247
integer:variable
numeric:variable ::= fixed:variable
floating:variable

```
163:  63

   order:directive  ::=  !ORDER  ;



164:  165

         null:declaration
         ordinary:table:item:declaration

   ordinary:table:body  ::=
                  ordinary:table:item:declaration
         BEGIN              END    ;
                  subordinate:overlay:declaration



165:  235

   ordinary:table:declaration  ::=  ordinary:table:heading
   ordinary:table:body



166:  165

   ordinary:table:heading  ::=
         environmental:specifier
      TABLE  table:name
         allocation:specifier

         :  allocation:increment     dimension:list

         structure:specifier         packing:specifier

         item:description         =  constant:list     ;
```

167: 131, 164

ordinary:table:item:declaration ::=

ITEM item:name item:description
packing:specifier = constant:list ;

168: 51

independent:overlay:declaration
overlay:declaration ::=
subordinate:overlay:declaration

169: 9, 166, 167, 182, 205, 217, 218

N

packing:specifier ::= M number

D

170:

actual:define:parameter
formal:define:parameter
parameter ::= actual:input:parameter
formal:input:parameter
actual:output:parameter
formal:output:parameter

```
171:   18, 39, 108

         1
         2
   pattern:constant ::= 3    B    count    ' pattern:digit
         4
         5


172:   171

         pattern pattern:digit   order

      0 0 0 0 0          0
      0 0 0 0 1          1              1
      0 0 0 1 0          2
      0 0 0 1 1          3              2
      0 0 1 0 0          4
      0 0 1 0 1          5
      0 0 1 1 0          6
      0 0 1 1 1          7              3
      0 1 0 0 0          8
      0 1 0 0 1          9
      0 1 0 1 0          A
      0 1 0 1 1          B
      0 1 1 0 0          C
      0 1 1 0 1          D
      0 1 1 1 0    pattern:digit ::=    E
      0 1 1 1 1          F              4
      1 0 0 0 0          G
      1 0 0 0 1          H
      1 0 0 1 0          I
      1 0 0 1 1          J
      1 0 1 0 0          K
      1 0 1 0 1          L
      1 0 1 1 0          M
      1 0 1 1 1          N
      1 1 0 0 0          O
      1 1 0 0 1          P
      1 1 0 1 0          Q
      1 1 0 1 1          R
      1 1 1 0 0          S
      1 1 1 0 1          T
      1 1 1 1 0          U
      1 1 1 1 1          V              5
```

```
173:  92

          1
          2
   pattern:format  ::=  3      B      count    P
          4
          5


174:  63

   pointer:directive  ::=  !POINTER    pointer:formula  ;
   data:name          ;


175:  5, 8, 67, 97, 101, 118, 121, 174, 180, 208, 237

          integer:constant
   pointer:formula  ::=  simple:integer:variable
          (  numeric:formula  )


176:  247

   pointer:variable  ::=  numeric:variable


177:  132

   precision:specifier  ::=  number
```

178: 221, 233

```
        ABS
        ALL
        ALT       NENT
        AND       NOT
        BEGIN     NULL
        FIT       NWDSEN
        BLOCK     OR
        BY        OVERLAY
        BYTE      PROC
        DEF       PROGRAM
        DEFINE    REF
        DIRECT    REMQUO
        DSIZE     RESERVE
        ELSE      RETURN
        END       SHIFT
primitive ::=     ENTER     SIG
        EQV       SIGNED
        EXIT      SIGNUM
        FOR       SIZE
        FORM      STATUS
        FORMAT    STOP
        FRAC      SWITCH
        GOTO      TABLE
        IF        TEST
        IN        THEN
        INT       TYPE
        ISIZE     UNTIL
        ITEM      WHILE
        JOVIAL    XOR
        LOC       XRAD
        NAME      ZAP
```

179: 181

```
        declaration
procedure;body ::=
        statement
```

```
180:  207

   procedure:call:statement  ::=

      remquo:procedure:call:statement

         procedure:name
                 @ pointer:formula
         alternate:entrance:name

            (  actual:input:parameter       )

            (    actual:input:parameter       :          ;

               actual:output:parameter     )



181:  59, 75, 184

   procedure:declaration  ::=  procedure:heading
   procedure:body



182:  181

   procedure:heading  ::=

                              environmental:specifier
   PROC procedure:name
                              data:allocation:specifier

      :  instruction:allocation:specifier

      (    formal:input:parameter          :
      formal:output:parameter       )

         environmental:specifier
                                   item:description
         allocation:specifier

      packing:specifier          [  bit:number  ]

         +
```

```
        =           constant      ;
          -
```

183: 5, 10, 53, 68, 90, 96, 101, 122, 138, 146, 180, 182, 167, 193

```
   procedure:name  ::=  name
```

184: 54

```
         program:declaration
   processing:declaration  ::=   procedure:declaration
         alternate:entrance:declaration
```

185: 184

```
         independent:program:declaration
   program:declaraton  ::=
         dependent:program:declaration
```

186: 68, 112

```
   program:name  ::=  name
```

187: 157

```
         status:list:name
         item:name
   qualified:status:constant  ::=  V(    table:name      :
   status  )
         procedure:name
         alternate:entrance:name
```

188: 63

```
recursive:directive  ::=  !RECURSIVE  ;
```

189: 16, 211

```
reference  ::=  number
```

190: 24, 33, 246

```
        <       less than
        =       equal
        >       greater than
relational:operator  ::=
        >=      greater than or equal, not less than
        <>      less than or greater than, not equal
        <=      less than or equal, not greater than
```

191: 180

```
remquo:procedure:call:statement  ::=

   REMQUO  (  actual:input:parameter  ,
   actual:input:parameter
      !  actual:output:parameter  ,
      actual:output:parameter  )  ;
```

192: 43

```
        formula
replacement:phrase ::= THEN
        value:formula
```

193: 207

```
        procedure:name
return:statement ::= RETURN          ;
        alternate:entrance:name
```

194: 77, 81

```
scale ::= number
```

195: 63

```
sets:directive ::= !SETS  data:name    ;
```

196: 18, 130

```
shift:function:call ::= SHIFT ( bit:formula ,
numeric:formula )
```

197: 25, 57, 234

```
        letter
sign ::=     numeral
        mark
```

198: 18, 130

signed:function:call ::= SIGNED (numeric:formula)

199: 82

count Z ::= count D . count D
| count D . count D
| count D .

significand ::= count D * count D
| count * count D
| count D count *

200: 130

significand:function:call ::= SIG (numeric:formula)

201: 132

significand:specifier ::= number

202: 130

signum:function:call ::= SIGNUM (numeric:formula)

203: 207

simple:assignment:statement ::= variable = formula ;

204: 175

simple:integer:variable ::= simple:variable

205: 49, 75, 131

simple:item:declaration ::=

environmental:specifier
ITEM item:name
allocation:specifier
item:description packing:specifier

+
[bit:number] = constant
;
=

206: 90, 91, 109, 143, 208, 243

simple:item:name ::= name

207: 219

```
        simple:assignment:statement
        assignment:statement
        exchange:statement
        go:to:statement
        exit:statement
        test:statement
   simple:statement  ::= return:statement
        zap:statement
        stop:statement
        loop:statement
        conditional:statement
        switch:statement
        procedure:call:statement
        direct:statement
```

208: 150, 204

```
   simple:variable  ::=  simple:item:name    @
   pointer:formula
```

209: 130

```
        formula
   size:function:call  ::=  SIZE  (              )
        data:block:name
```

210: 132

```
   size:specifier  ::=  number
```

```
211:  63
   skip:directive  ::=  !SKIP    reference    ;


212:  92
   skip:format  ::=  X


213:  63
   space:directive  ::=  !SPACE    character        ;


214:  109
   spacer  ::=  number


215:  216
        null:declaration
   specified:table:body  ::=
   specified:table:item:declaration
        BEGIN specified:table:item:declaration      END    ;


216:  235
   specified:table:declaration  ::=  specified:table:heading
   specified:table:body
```

217: 216

```
specified:table:heading ::=

      environmental:specifier
TABLE table:name
      allocation:specifier

      : allocation:increment    dimension:list

      structure:specifier

      words:per:entry
      bits:per:entry        bit:number    entries:per:word

      packing:specifier       item:description
      packing:specifier

      [ bit:number    , word:number    ]      =
      constant:list    ;
```

218: 131, 215

```
specified:table:item:declaration ::=

   ITEM item:name      item:description

      packing:specifier    [ bit:number

      , word:number     ]    = constant    ;
```

219: 36, 45, 112, 149, 179, 232

```
      null:statement
statement ::=        simple:statement
      compound:statement
      named:statement
```

220: 5, 38, 71, 90, 104, 138, 146, 149, 232

statement:name ::= name

221: 187, 222, 233

status ::= primitive
name
letter

222: 157, 223

status:constant ::= V(status)

223: 132, 224

status:list ::=
[number]+ status:constant [number]+ status:constant

224: 54

status:list:declaration ::= STATUS status:list:name status:list ;

225: 132, 187, 224

status:list:name ::= name

```
226:   207

   stop:statement  ::=  STOP  ;


227:   166, 217

          P
   structure:specifier  ::=
          T


228:   164, 168

   subordinate:overlay:declaration  ::=  OVERLAY
   subordinate:overlay:expression  ;


229:   231

          item:name
   subordinate:overlay:element  ::=
          (  subordinate:overlay:expression  )


230:   228, 229

   subordinate:overlay:expression  ::=

      subordinate:overlay:string    :
      subordinate:overlay:string
```

231: 230

subordinate:overlay:string ::=
subordinate:overlay:element

232: 207

switch:statement ::=

SWITCH numeric:formula ; statement:name :
+
BEGIN [number] statement
, END ;
-

233:

primitive
ideogram
name
letter:control:variable
symbol ::= abbreviation
number
constant
comment
directive:key
status

234: 25

system:dependent:character

Most computer systems can read and write more characters than are encompassed in the set of JOVIAL sign. The entire set that can be handled is know as the set of characters. The characters that are not signs are known as system:dependent:characters.

```
235:   49, 51, 75

         ordinary:table:declaration
   table:declaration  ::=
         specified:table:declaration



236:   5, 52, 67, 90, 109, 118, 138, 155, 166, 187, 217, 251,
252

   table:name  ::=  name



237:   117
   table:variable  ::=  item:name    [    index    ]        @
   pointer:formula



238:   43

         WHILE
                  conditional:formula
   terminator:phrase  ::=        UNTIL

         value:terminator



239:   207

   test:statement  ::=  TEST    control:variable    ;



240:   63

   time:directive  ::=  !TIME    character        ;
```

```
241:  63
   trace:directive  ::=  !TRACE    (  conditional:formula  )
   name    ;

242:  130
   type:function:call  ::=  TYPE  (  formula  )

243:  61
         number
   upper:bound  ::=
         simple:item:name

244:  63
   uses:directive  ::=  !USES  data:name     ;

245:  97, 192, 246
   value:formula  ::=  [  formula  ]
```

```
246:   238
   value:terminator  ::=

      WHILE       value:formula  relational:operator  variable
      UNTIL       variable  relational:operator  value:formula



247:   5, 6, 14, 70, 203, 246

         pointer:variable
         numeric:variable
   variable  ::= bit:variable
         character:variable
         format:variable



248:   5, 6, 14, 70, 203, 246

         named:variable
   variable  ::= letter:control:variable
         functional:variable



249:   217, 218

   word:number  ::=  number



250:   217

   words:per:entry  ::=  number
```

```
251:   130

   words:per:entry:function:call  ::=  NWDSEN  (  table:name
   )



252:   207

         table:name
   zap:statement  ::=  ZAP        ;
         entry:variable
```

**********TABLES**********

Column	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15

Code	0	1	2	3	4	5	6	7
	8	9	A	B	C	D	E	F

Row								
0			space	0	@	P		p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			'	7	G	W	g	w
8			(	8	H	X	h	x
9			)	9	I	Y	i	y
10			*	:	J	Z	j	z
11			+	;	K	[	k	
12			,	<	L	\	l	
13			-	=	M	]	m	
14			.	>	N		n	
15			/	?	O		o	

Notes: row 0, column 3: zero
row 1, column 3: one
row 7, column 2: prime, often rendered as a vertical mark in JOVIAL
row 12, column 6: a lowercase letter
row 15, column 4: an uppercase letter

Figure 2-1. Characters

fixed:constant		value	size	precision
19A0	19	5	0	
19A3	19	8	3	
19A-2	16	3	-2	
2.3A0	2	2	0	
2.3A-1	2	1	-1	
2.3A2	2.25	4	2	
2.3A5	2.28125	7		5
2.3A6	2.296875	8		6

Left symbol ends in	Right symbol starts with: numeral	letter	$	'	.	"
numeral		SR		SR	SR	SR
letter		SR		SR	SR	SR
$	SR		SR	SR	SR	
'	SR		SR	SR	SR	
.	SR		SR			
"						

SR

Value of the conditional:formula

	0	1
IF	Skip the controlled:statement following this conditional:formula,	Execute the following controlled:statement then skip the controlled:statement
	Execute the controlled:statement following the matching ELSE if there is one,	immediately following the matching ELSE if there is one,
UNTIL	Execute the controlled:statement,	Go on to the next control:clause or exit the loop if there is

no further control:clause,

WHILE Go on to the next contol:clause Execute the
controlled:statement,
or exit the loop if there
is no further control:clause,

bit:formula 110 01011100 01010111
10000101 01111100
padded 00000110 01011100 01010111
10000101 01111100
selected 01011100 01010111
10000101

x	y	x\y	x	y	x\y
7	0	undefined	-3,7	2	0,3
1	2	1	4,6	1,5	0,1
2	2	0	-0,1	1,5	1,4
3	2	1	1	-2	-1
-3	2	1	3,7	-2	-0,3
3,7	2	1,7	-3,7	-2	-1,7

First or Only Character	Meaning
I	Number of integer bits
A	Number of fraction bits (bits after the point).
Z	Maximum size I+A the system normally allows for fixed and integer arithemetic.
Y	An even larger maximum size I+A allowed under evaluation:control (often about 2*z).
V	Value.

Second Character	Meaning: "Of the..."
1	First operand
2	Second operand
M	Modulus (for x\y).
N	Numerator (for x/y or x\y).
D	Denominator (for x/y).
I	Integer operand (if the other is fixed).
A	Fixed operand (if the other is integer).
R	Result (preliminary result if S exists).
S	Result required by evaluation:control.
B	Base in exponentation.

E Exponent.

Value of original bit:formula	Value of numeric:formula	Value of bit:formula resulting from SHIFT
11111	3	11000
11111	-1.7	01111
00000100000	5	10000000000
00000100000	-3	00000000100
101	3	000
101	-3	000
101	-2	001

bit:formula	&	bit:formula	result
10	1	101	
111001	00011110101	11100100011110101	
00010000	0000010	000100000000010	
0	0	00	

p	q	NOT p	p OR q	p EQV q	p AND q	P XOR q
0	0	1	0	1	0	0
0	1	1	1	0	0	1
1	0	0	1	0	0	1
1	1	0	1	1	1	0

```
0     = (assignment)  == (exchange)
1     EQV    XOR
2     OR
3     AND     (logical)
4     NOT
5     =  <  >  <=  >=  <>     (relational)
6     &
7     + -
8     * / \   ( with or without evaluation:control )
9     **
10    indexing @      ( pointing, evaluation:control )
      @@      ( attribute:association )
```

In the algorithm it is necessary to consider several operands and operations simultaneously. The following diagram shows the relationships. All are pegged in relation to the present operand. Any operation may replace #.

```
                A # B # C # D # E # F # G # H # I # J # K

                                          the next operation

      the prior operand                    the next operand

      the prior operation                 the current operation

                        the present operand
```

The leftmost operand of the formula is initally the present operand.

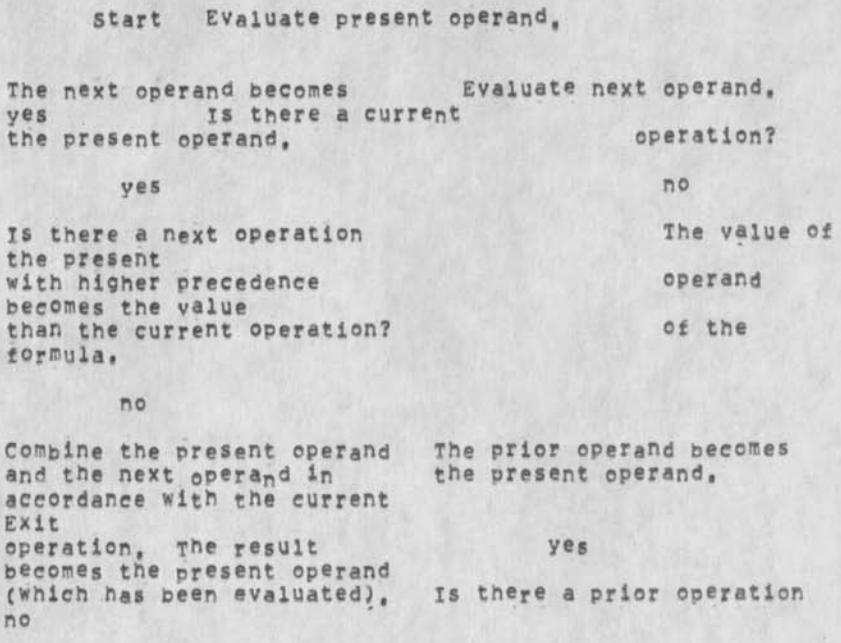

Figure 4-2. Combination Algorithm

ABC Operation type	XYZ type Char	Bit	Int	Fix	Float
	Converted to				
ABC assignment XYZ	Char	Char	Bit	Bit	Bit
(also parameter	Bit	Bit	Bit	Bit	Bit
matching and	Int	Bit	Bit	Int	Int
exchange, both ways)	Fix	Bit	Bit	Fix	Fix
	Float	Bit	Bit	Float	Float
ABC arithmetic XYZ	Float	Note 1	Note 2	Float	Float
XYZ arithmetic ABC	Other	Note 3	Note 4	Scale	Float
ABC relational XYZ	Char	Note 5	Int	Note 6	Note 6
	Bit	Note 6	Int	Note 6	Note 6
or	Int	Note 6	Int	Int	Scale
XYZ relational ABC	Float	Note 6	Int	Float	Float
ABC & XYZ	Char	Char	Bit	Bit	Bit	Bit
XYZ & ABC	Other	Bit	Bit	Bit	Bit	Bit
ABC logical XYZ	Any	Bit	Bit	Bit	Bit	Bit
XYZ logical ABC						
Indexing, pointing		Note 7	Int	Int	Int	Int

Figure 4-3, Type Conversion

Entrance used	ALT (procedure:name)
integer	status:Constant
normal 0	V (procedure:name)

```
first alternate          1          V ( first
alternate:entrance:name )
second alternate         2          V ( second
alternate:entrance:name )
etc.     etc.                       etc.
```

```
  parameter   type:function:call
qualified:status:constant
     type              value   status:constant

bit:formula            0              V(BIT)
V(TYPE:BIT)

integer:formula                1              V(INT)
V(TYPE:INT)
(signed or unsigned)
fixed:formula          2              V(FIX)
V(TYPE:FIX)
(signed or unsigned)
floating:formula               3              V(FLOAT)
V(TYPE:FLOAT)

character:formula              4              V(BYTE)
V(TYPE:BYTE)
```

```
START


          0-- skip A1, do A2
  A?       E=3

     1-- do A1

          0-- skip B1              0-- skip D1, do D2
(NULL
  B?                          D?
             do B2                              or none),
skip A2

     1-- do B1                   1-- do D1

                                        Skip D2 (if any
and A2
                                 E=2

          0-- skip C1, do C2 (NULL or none), skip B2
and A2
  C?

     1-- do C1

           Skip C2 (if anY), B2 and A2
E=1        EXIT
```

	No terminator:phrase	Terminator:phrase
No initial:phrase, replacement:phrase, or increment:phrase	1A. Leave control:variable alone. Execute controlled:statement just once.	1B. Same as 1A except execute controlled:statement zero or one time depending onterminator:phrase.
Initial:phrase only	2A. Initialize control:variable. Execute controlled:statement just once.	2B. Same as 2A except execute controlled:statement zero or one time depending on terminator:phrase.
Replacement:phrase only	3A. Leave control:variable alone for the first execution. Before each subsequent execution of the controlled:statement, replace the value of thecontrol:variable.	3B. Same as 3A except test in accordance with terminator:phrase before every execution of the controlled:statement -- even the first one. Repeat executions "forever".
Increment:phrase only	4A. Same as 3A except add to value of control:variable instead of replacing value.	4B. Same as 4A except test as in 3B.
Initial:phrase and replacement:phrase	5A. Initialize control:variable. Execute controlled:statement once. Replace value of control:variable before each subsequent execution. Repeat executions "forever".	5B. Same as 5A except check for termination before each execution.
Initial:phrase and increment:phrase	6A. Initialize control:variable. Execute controlled:statement once. Add to value of control:variable before each subsequent execution.	6B. Same as 6A, except check for termination before each execution.

Repeat executions "forever".

```
                1
start           ALPHA           Set BETA to 3

          =

          ?     2 or 3    Set GAMMA to the
                          value of BETA

                4         If GAMMA equals 2
                          set BETA to 2

                6         Set BETA to the
                          value of GAMMA



                          Set ALPHA to 7  next

                <1, 5 or >6
                          Undefined
```

```
                                                        Format:Lists

Input Buffer  Field             ,,10C           10C,,

28,3b'bABCD'bABb                1           '28,3bb'
'28,3bs27bABC'
              2           'bABCDb'          'D$27bbbb'
              3           'ABbbbbbbbb'      'ABbbbb'

ALPHABET'AbTHERMOPILEb                      1       'ALPHAb'
'ALPHAbBETs27'
              2           'BETs27Ab'        'Abbbbb'
              3           'THERMOPILE'      'THERMO'
```

```
      4B3PS3PS3PS3P   5B5PS5P

Bah!  000 042 616 821 00011 62Q11
Humbug      487 56D 627 567 28ELM M4TB7



Input Buffer  Field   5N, 6N, SNSN

1.2E3-b485bb3b7             1       1.2E3
              2          -485
              3           37

+46b7,00015A1B,             1       illegal field
(contains blank)
              2          ,00015
              3          1,



      +SD4D-3ZDDR     -SDSZSZSZ       +SD","ZZ","DDR

  1573,64      + 1573   1574   1 5 7 3          + 15,74
  - 27         - 0027    -27  -2 7        - 27
    0,0          0000     00   0              00
-10740         undefined      -10740  undefined       -
1,07,40



                                  Format;Lists
Input Buffer  Field   "SPEEDb",DDD,"MPH       ,DDDSSSS

SPEEDb100bMPH         1               100     'SPEED'
              2                               100
```

```
      +ZZDD,DZZ        -4Z*3DR -4Z*     -4Z2*R

 1573,6405     +1573,64         1573641 1573      157
  -27    -27,0  -27000  -27       -3
    0,0           00,0     000
-10740         undefined        undefined        undefined
-1074


      1573,6400        1573,6410        1573,0000
1570,0000
       -27      -27      -27      -30
   0,0            0,0      0,0      0,0
      undefined        undefined        undefined
-10740


      +,6DE+3ZR        -SD*6ZSES-3Z*    -S3D,SES-3DR
+S3*5DSES+S3Z

+ 39,7528      +,397528E+2        397528  E   1   398, E
-001   + 39752 E + 4
-,008711246    -,871125E-2      - 8711246 E  -3 - 871, E
-005   - 87112
```

```
PROGRAM AA
   XX (table:name)

      PROC BB
         XX (item:name)

            PROC CC
               no occurrence of XX

                  PROC DD
                  XX used but not declared



                  PROC EE
```

Figure 7-1. Scope of Names

Serial Structure	Parallel Structure
1st half AB[0]	1st half AB[0]
2d half AB[0]	1st half AB[1]
XY[0]	1st half AB[2]
1st half AB[1]	1st half AB[3]
2d half AB[1]	2d half AB[0]
XY[1]	2d half AB[1]
1st half AB[2]	2d half AB[2]
2d half AB[2]	2d half AB[3]
XY[2]	XY[0]
1st half AB[3]	XY[1]
2d half AB[3]	XY[2]
XY[3]	XY[3]

Example: Table MN has 2 items, AB and XY, and 4 entries, 0, 1, 2, and 3,
Item AB occupies 2 words,
Item XY occupies 1 word,
Note: 12 consecutive computer words are shown in each illustration above.

Figure 7-2. Serial and Parallel Table Structure

Tight Structure

entry [0]	entry [1]	entry [2]
entry [3]	entry [4]	entry [5]

A table of six entries is medium packed, three entries to the word.

```
OVERLAY AA, AB, AC : BA, (BX : BY, BZ), BC ;

  AC              AA                AB

   BA               BX                BC

                   BY    BZ
```

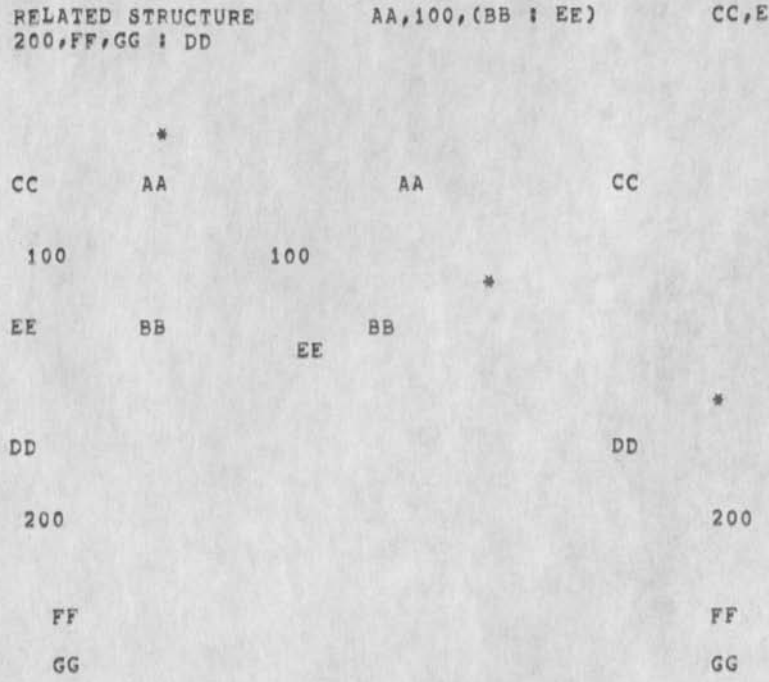

Figure 7-4. Allocation of a Related Structure

entrance	number	status:constant
normal	0	V(procedure:name)
first alternate	1	V(alternate:entrance:name)
second alternate	2	V(alternate:entrance:name)

RELATIVE WORD	SERIAL	PARALLEL
0	1 AB[0,0,0]	1 AB[0,0,0]
1	2 AB[0,0,0]	1 AB[0,0,1]
2	XY[0,0,0]	1 AB[0,1,0]
3	1 AB[0,0,1]	1 AB[0,1,1]
4	2 AB[0,0,1]	1 AB[0,2,0]
5	XY[0,0,1]	1 AB[0,2,1]
6	1 AB[0,1,0]	1 AB[0,3,0]
7	2 AB[0,1,0	1 AB[0,3,1]
8	XY[0,1,0]	1 AB[1,0,0]
9	1 AB[0,1,1]	1 AB[1,0,1]
10	2 AB[0,1,1]	1 AB[1,1,0]
11	XY[0,1,1]	1 AB[1,1,1]
12	1 AB[0,2,0]	1 AB[1,2,0]
13	2 AB[0,2,0]	1 AB[1,2,1]
14	XY[0,2,0]	1 AB[1,3,0]
15	1 AB[0,2,1]	1 AB[1,3,1]
16	2 AB[0,2,1]	1 AB[2,0,0]
17	XY[0,2,1]	1 AB[2,0,1]
18	1 AB[0,3,0]	1 AB[2,1,0]
19	2 AB[0,3,0]	1 AB[2,1,1]
20	XY[0,3,0]	1 AB[2,2,0]
21	1 AB[0,3,1]	1 AB[2,2,1]
22	2 AB[0,3,1]	1 AB[2,3,0]
23	XY[0,3,1]	1 AB[2,3,1]
24	1 AB[1,0,0]	2 AB[0,0,0]
25	2 AB[1,0,0]	2 AB[0,0,1]
.	.	.
.	.	.
.	.	.
62	XY[2,2,0]	XY[1,3,0]
63	1 AB[2,2,1]	XY[1,3,1]
64	2 AB[2,2,1]	XY[2,0,0]
65	XY[2,2,1]	XY[2,0,1]
66	1 AB[2,3,0]	XY[2,1,0]
67	2 AB[2,3,0]	XY[2,1,1]
68	XY[2,3,0]	XY[2,2,0]
69	1 AB[2,3,1]	XY[2,2,1]
70	2 AB[2,3,1]	XY[2,3,0]
71	XY[2,3,1]	XY[2,3,1]

Figure 10-1 Indexing and Storage Allocation

Bits				Bits		
0-1	2-11	12-21	22-31	0-1	2-11	12-21	22-31
0		BB[0,0]	BB[0,1]	bB[0,2]		BB[0,0]	BB[0,1]
BB[0,2]							
1		BB[0,3]	BB[0,4]	BB[0,5]		BB[0,3]	BB[0,4]
BB[0,5]							
2		BB[0,6]	BB[0,7]			BB[0,6]	BB[0,7]
BB[1,0]							
3		BB[1,0]	BB[1,1]	BB[1,2]		BB[1,1]	BB[1,2]
BB[1,3]							
4		BB[1,3]	BB[1,4]	BB[1,5]		BB[1,4]	BB[1,5]
BB[1,6]							
5		BB[1,6]	BB[1,7]			BB[1,7]	BB[2,0]
BB[2,1]							
6		BB[2,0]	BB[2,1]	BB[2,2]		BB[2,2]	BB[2,3]
BB[2,4]							
7		BB[2,3]	BB[2,4]	BB[2,5]		BB[2,5]	BB[2,6]
BB[2,7]							
8		BB[2,6]	BB[2,7]			BB[3,0]	BB[3,1]
BB[3,2]							
9		BB[3,0]	BB[3,1]	BB[3,2]		BB[3,3]	BB[3,4]
BB[3,5]							
10		BB[3,3]	BB[3,4]	BB[3,5]		BB[3,6]	BB[3,7]
BB[4,0]							
11		BB[3,6]	BB[3,7]			BB[4,1]	BB[4,2]
BB[4,3]							
12		BB[4,0]	BB[4,1]	BB[4,2]		BB[4,4]	BB[4,5]
BB[4,6]							
13		BB[4,3]	BB[4,4]	BB[4,5]		BB[4,7]	BB[5,0]
BB[5,1]							
14		BB[4,6]	BB[4,7]			BB[5,2]	BB[5,3]
BB[5,4]							
15		BB[5,0]	BB[5,1]	BB[5,2]		BB[5,5]	BB[5,6]
BB[5,7]							
16		BB[5,3]	BB[5,4]	BB[5,5]			
17		BB[5,6]	BB[5,7]				

Figure 10-2 Indexing and Allocating Tight Structure Tables

Interim Report

BACKGROUND: 1

Maj. Smith AFCS called IS at the suggestion of Col. McGinnis. 1a

Maj. Smith asked that someone from here call Sgt. Mixon, AFCS/DOMA, 465-2295 to provide AFCS with information on whether or not there is commercially available equipment at their price(about $1K per month per installation) that they can use in satisfying their requirement. 1b

PROBLEM: 2

AFCS has stations world-wide that they would like to replace. There are a total of 12 locations. The locations seem to consist of low data rate relay stations. The major job is to call up preformatted message forms which include the addresses. Teletypes which activate a tape punch directly are used To fill in the message information. After a number of messages have been put on tape the tape is read into a tape reader which puts out the info for the comm lines. 2a

The maximum message size is twelve 70 character lines. The tape itself is five level Baud 0 code which is 11/16 inch wide. 2b

The baud rate is only 50. 2c

They would like to replace the present equipment, up to the point where the tape itself is punched, by three CRT's and a 'black box' that can be used to call up the formats, fill in the blancks and insert the messages, edit on line, store the work of three men in one location where the information can be output in a single stream to punch the tape that gets carried over to the tape reader and then out over the comm lines. 2d

They figure they can save 48 people if they can get this equipment set up and properly working. 2e

COURSE OF ACTION: 3

Start with the terminal study that was done in ISIM (the reason we were given this honor in the first place). 3a

Check with ISF to see what they can contribute. 3b

Check with Jim Pope who should be 'up' on these typees of things. 3c

Call Sgt. Mixon by this Friday or by Monday at the latest. If not by then, then the call should be the following Friday(TDY). 3d

DISCUSSION WITH FRANK TROILO; 4

Frank gave the opinion that the technology is available to do this however, the equipment that is available is all eight level code which is not compatible with the five level code that is presently in use, 4a

His first suggestion was that they get rid of the tape punchers and tape readers and go directly to the comm input. I explained that they didn't want to make waves that big, 4b

His second suggestion was that they throw out the slow 50 baud network and substitute a modern high speed eight baud level net. I indicated that probably you couldn't do that for about $1K per month, 4c

I think that Frank's suggestions are good, or at least similar to the ones I would make myself, but ... 4d

The next reasonable alternative is to build a converter to cut eight baud back to five baud level. But this throws away a lot of capability that you paid for in the first place and limits the character reportory, 4e

Frank indicated that he was quite heavily loaded and would prefer not to take the thing over at this time, but he said he would be glad to act as a consultant, 4f

DISCUSSION WITH JIM POPE 5

The call to pope resulted in some interesting revelations: 5a

1. AFCS has already completed a study and come up with exactly what we were asked to do. The 'bosses' in NCA like it and want ot buy a version for checkou, but the Sergeants do not. So things have come to a screeching halt, 5a1

2. Dave Griffin of Comm has done a study on the same problem for AFCS and come up with a $50K solution for the first model. He has been checking wwith Jim Pope also, 5a2

3. We are apparently the ones that AFCS has chosen to go to for a third opinion, 5a3

Jim agreed to call Sgt. Mixon to day. He will let me know what happens as a result. (Jim knows both Maj. Smith and Sgt. Mixon) 5b

Pso Schedule

The schedule for Donna and Sharon is identical:		1
Monday	1030 - 1700 (every other week)	1a
Tuesday	1230 - 1700	1b
Wednesday	1030 - 1700	1c
Thursday	1230 - 1700	1d
Friday	1030 - 1700	1e
Duayna is scheduled for:		2
Monday	1430 - 1700	2a
Tuesday	0815 - 1000 and 1530 - 1700	2b
Wednesday	1500 - 1700	2c
Thursday	1430 - 1700	2d
Friday	1430 - 1700	2e

PTI meeting

Meeting with PTI 9-25-74 1

Present were: 1a

Dr. Elizabeth Cuthill (NSRDC) 1a1

Larry Avrunin (NSRDC) 1a2

Frank Brignoli (NSRDC 1a3

Arleigh Marlsham (?) (PTI 1a4

Stan Goldberg (PTI) 1a5

Regional Manager 1a5a

Urban Technology System 1a5b

Public Technology Inc. 1a5c

1140 Connecticut Ave. N.W. 1a5d

Washington, D.C. 20036 1a5e

Tel: 202-223-8240 1a5f

Mr. Joseph D. Antinucci (NSF) 1a6

Program Manager 1a6a

Federal Laboratories Liaison 1a6b

Office of Intergovernmental Science and Research Utilization 1a6c

Room 601 1a6d

National Science Foundation 1a6e

1800 G St. N.W. 1a6f

Washington, D.C. 20550 1a6g

Tel: 202-632-5924 1a6h

Discussed: 1b

Mr. Antinucci of NSF is program manager for a DOD consortium of Army, Navy, and Air Force labs (approx. 30). This is becoming a federal consortium of labs and is expected to grow to include

non-DOD labs (maybe 700 throughout USA). They want technology transfer from DOD out. Currently such transfer occurs thru meetings, telephone calls, etc. They might like to automate it. (This is the part amenable to NLS). They also have an information acquisition and dissemination problem; mentioned NTIS & 1498's as example. (this is the part not directly addressed by NLS; they will continue to have a problem for some time to come). 1b1

Messrs. Goldberg and Marlsham (?) are members of Public Technology Inc. (PTI), a not for profit organization organized by a number of state and local government associations. They are 3 months into an NSF funded 53 month experiment in which they have set up a network of 27 cities and counties with populations in the range of 50K to 500K. In each one, they have placed a staff technology agent (a technical manager if I understood correctly) who works with the city to identify problem areas,etc. amenable to technology transfer. They want a automated system that would allow their agents to communicate such items as problems, solutions existing and proposed, and background research material. (They fall, in my opinion, directly under the umbrella of NLS services). 1b2

I demonstrated NLS to them, offered to allow them experimental use of the system under our group, and suggested they contact ARC directly for a detailed explanation of the NLS service. 1b3

Feedback on NNLS

This file has two purposes: 1

Give me a file to play with while I learn NLS-8 1a

To document some of my findings as I go along. 1b

NOTA BENE: THIS IS A PERSONAL FILE WHICH I MAY CHOOSE TO MAIL FOR THE PURPOSE OF LETTING SRI KNOW MY REACTIONS. TAKE THE COMMENTS IN THE SPIRIT IN WHICH THEY ARE OFFERED. THERE IS NO INTENT TO SLANDER OR LIBEL ANY INDIVIDUAL OR ORGANIZATION, EXCEPT POSSIBLY 'THEY' WHICH IS UNDEFINED AND INDEFINITE. 2

25 Sep 74 3

The PROCESS command does not seem to exist in the TTY mode. All I am able to get is print and playback. 3a

27 Sep 74 4

I think Stoney reported that he was unable to get a printfile going in any of the modes. I too have managed nothing except to get the header in. 4a

In addition I can't seem to find something in NLS-8, that works, that will do the old ODT. 4b

In the SENDMAIL subsystem, I think I indicated earlier that the source request confused the hell out of me, but I got that squared away. Today I didn't use the interrogate command as I usually do for mail. So I went through it by myself without the prompts from interrogate. When I got to the point where I wanted to tell it where the material was I typed an s and got sendamail. So went to HELP after looking over the menu that I got from the ?. While in the HELP I typed source and was told HELP couldn't find it. So I tried a ? sometime later and was told ? is not available call ARC. Consider yourself called. 4c

I am finding the HELP helpful, now that I have gotten some experience and have a feel for its capabilites and, forgive me, limitations. 4d

The new feature of having the command go away after it is executed takes a little bit of getting used to. BUT, at least it is a failsafe feature. I screwed up on occasion when I forgot that my last command was a jump or a delete using the 'old' NLS. 4e

Inserting statements, I find the control b and the control e options very helpful, particularly on the TTY. Combining them with the feedback option for levadj, should make it relatively

simple for novices to keep track of what they are doing on the TTY. 4f

30 Sep 74 5

My major complaint is that the system is so slow. 5a

Second comes the inability to make a print copy. 5b

Third I guess is the absence of the user-progs, otherwise things seem to be running fairly smooth on the IMLAC. Of course, I have not gone too far into some of the new features. First I am trying to see what it takes to transition from NLS-7 to -8. 5c

My conclusions to this point are that the IMLAC user will not have too much trouble in doing his daily work in his accustomed manner. As he develops skill he will start using the new features in much the way people have been doing all along. The TTY user, flying blind as it were, is going to have a very difficult time. I feel that some intensive retraining is needed. Since most of our users here are TTY users I think we have a problem that calls for a serious look at our training requirements. 5d

EJK 30-SEP-74 14:47 31105

Close-out Report - Maj. Smith, Sgt. Mixon, AFCS

This is several pages long don't print it out on a teletype terminal.

EJK 30-SEP-74 14:47 31105

Close-out Report - Maj. Smith, Sgt. Mixon, AFCS

BACKGROUND: 1

Maj. Smith AFCS called IS at the suggestion of Col. McGinnis. 1a

Maj. Smith asked that someone from here call Sgt. Mixon, AFCS/DOMA, 465-2295 to provide AFCS with information on whether or not there is commercially available equipment at their price(about $1K per month per installation) that they can use in satisfying their requirement. 1b

PROBLEM: 2

AFCS has stations world-wide that they would like to replace. There are a total of 12 locations. The locations seem to consist of low data rate relay stations. The major job is to call up preformatted message forms which include the addresses. Teletypes which activate a tape punch directly are used To fill in the message information. After a number of messages have been put on tape the tape is read into a tape reader which puts out the info for the comm lines. 2a

The maximum message size is twelve 70 character lines. The tape itself is five level Baud 0 code which is 11/16 inch wide. 2b

The baud rate is only 50. 2c

They would like to replace the present equipment, up to the point where the tape itself is punched, by three CRT's and a 'black box' that can be used to call up the formats, fill in the blancks and insert the messages, edit on line, store the work of three men in one location where the information can be output in a single stream to punch the tape that gets carried over to the tape reader and then out over the comm lines. 2d

They figure they can save 48 people if they can get this equipment set up and properly working. 2e

COURSE OF ACTION: 3

Start with the terminal study that was done in ISIM (the reason we were given this honor in the first place). 3a

Check with ISF to see what they can contribute. 3b

Check with Jim Pope who should be 'up' on these typees of things. 3c

Call Sgt. Mixon by this Friday or by Monday at the latest. If not by then, then the call should be the following Friday(TDY). 3d

DISCUSSION WITH FRANK TROILO: 4

Frank gave the opinion that the technology is available to do this however, the equipment that is available is all eight level code which is not compatible with the five level code that is presently in use. 4a

His first suggestion was that they get rid of the tape punchers and tape readers and go directly to the comm input. I explained that they didn't want to make waves that big. 4b

His second suggestion was that they throw out the slow 50 baud network and substitute a modern high speed eight baud level net. I indicated that probably you couldn't do that for about $1K per month. 4c

I think that Frank's suggestions are good, or at least similar to the ones I would make myself, but ... 4d

The next reasonable alternative is to build a converter to cut eight baud back to five baud level. But this throws away a lot of capability that you paid for in the first place and limits the character reportory. 4e

Frank indicated that he was quite heavily loaded and would prefer not to take the thing over at this time, but he said he would be glad to act as a consultant. 4f

DISCUSSION WITH JIM POPE: 5

The call to Pope resulted in some interesting revelations: 5a

1. AFCS has already completed a study and come up with exactly what we were asked to do. The 'bosses' in NCA like it and want to buy a version for checkout, but the Sergeants do not. So things have come to a screeching halt. 5a1

2. Dave Griffin of Comm has done a study on the same problem for AFCS and come up with a $50K solution for the first model. He has been checking wwith Jim Pope also. 5a2

3. We are apparently the ones that AFCS has chosen to go to for a third opinion. 5a3

Jim agreed to call Sgt. Mixon today. He will let me know what happens as a result. (Jim knows both Maj. Smith and Sgt. Mixon) 5b

DISCUSSION BETWEEN JIM POPE AND SGT. MIXON: 6

Jim Pope told Sgt. Mixon of the work that they did at NCA on the problem of replacing the teletype/tape punch equipment with display terminals. 6a

Although there are a number of of ways of soving their problem, the cleanest solution seems to be the Raytheon UPI Editing System. This consists of a mini-computer with as many CRT's as you want. Three is well within the capabilility of the mini-. The system uses Cassette tapes and is a five level Baud 0 system that was designed for such use. The kicker is that the system sells for about 50K and I have no idea of what it would rent for but 10 to 12K per year is not unreasonable. 6b

The Raytheon point of contact is Dan Kelly 443-9521 x2341. (Can get him through Hanscom switchboard) 6c

STATUS: 7

There are of course, many alternatives. One of these is the use of the Beehive terminal with a suitable means of getting the formats stored so that they can be inputted to the internal logic of the display terminal and displayed so that the messages can be inserted. Then a sendprint type of command would output the stored messages to the punch. 7a

Jim proposes the use of a Sykes recorder and Frank Troilo suggests the use of hardwired boards to provide the fixed formats. 7b

I will do a little more homework and then call Sgt. Mixon. 7c

Discussion with Frank Troilo 8

I reviewed the situation and what I had been able to determine with Frank. We agreed that the five level Baudot code was a limiting factor and the Raytheon approach, which seemingly was designed to do the job that AFCS was after, seemed to be the best one. It is possible by doing a lot of work, maybe 2 or 3 man-months we might be able to come up with something better. But this does not appear to be either desirable or cost-effective. 8a

The Super-Bee/Tape Cassette seems an interesting possibility, but again there is a limit inasmuch as it is eight level and again we would need a way to cut back to five level. 8b

Frank said he has the specs and will check to see if any of these machines have a five level option. I told him to cool it until after I called AFCS. 8c

Close-out Report - Maj. Smith, Sgt. Mixon, AFCS

We discussed the five level code and the fact that it is outmoded and should be replaced. Agreed I would ask about the code. 8d

Telecall to Sgt. Mixon 9

The first part of the call was a replay of what we already knew. The only additional innformation was that the five level Baudot and the 50 Baud per second rate is required by the Comm systems with which they interface. 9a

The two systems are the Autodin and the IKO (International Communications Organization) 9b

Although 'they' have been threatening to update these to modern eight level code for the last several years it has not been done. The changeover is however still planned for the future. I alerted the Sgt. to the fact that whatever they buy it must be capable of changing over to the eight level when the time comes. 9c

I discussed with him the possible alternatives that I had picked out of the thousands that are probably available. These were: 9d

The Raytheon UPI System 9d1

The Superbee terminal - with some method of storing the formats. Tape would be my first choice. 9d2

The Teletype Corp Model 40 - which was designed as a replacement for the model 33 and the ruggedized 33. 9d3

The MILGO Model 40+. Designed to compete with the model 40 9d4

The VT-50 DECscope 9d5

The Bendix message oriented CRT. 9d6

I pointed out to Sgt. Mixon that basically he faced two options: 9e

He could opt for inexpensive terminals tied to a fairly expensive mini-computer. 9e1

He could go for the more expensive 'intelligent' terminals. 9e2

The cost of the hardware using either of these two options would be roughly comparable. 9f

In addition I pointed out to him the dangers of selecting equipment and trying to configure a system using components not designed to play together. I suggested that the best option for

him at this time is the Raytheon. Strangely enough he din't seem to be interested in anything except the cost. Then he indicated that AFCS needed to perform a cost-effectiveness study to get any approval to do anything. His major goal seems to have been to assure himself that something can be bought on the commercial market and to determine the approximate cost of that something. He stated that in any event AFCS would go out open bid. 9g

I recommended to him that he get a copy of the current Modern Data 197[illegible]for a review of the current state of the art and a listing of the most prominent, and stable manufactureres. In fact, I offered to send him a copy but he wasn't interested, I guess he already had all he needed. 9h

FUTURE ACTION: 10

It was left at the level of "Don't call us - We'll call you. He has my name and number and I fervently hope he loses them. 10a

SMT 30-SEP-74 19:00 31106

My travel this week -- 10/2 -10/4-5

I am 'forced' to return to "God's" country, to attend meetings at the Army Labs located at NASA/AMES. I'll arrrive Wed. 10/2, around noon. I'll drop by to spend some time with you and your people. I'd like to work out a few mysteries on/in NNLS. Also have further conversations with JHB, et al, regarding BRL training sessions. I'll be glaad to spend late hours, after we finish oursessions at AMES -- even Sat. am is a possibility. Will see you Wed. afternoon.
STAN

1

MIKE 1-OCT-74 10:04 31107

copy of Bedford to Ruggles memo re coop'n between groups

Well, Inez Mattiuz is now fully debriefed after your Architect's workshop on the West Coast; she related how a number of you arch's, were less than enthusiastic about the support provided to date by ARC.
She also mentioned that the arch's, were planning to get together informally to exhange information and opinions regarding use of the system, and I agreed with her that this seemed to be a fairly promising alternative given the current state of affairs.
In this context, I was wondering if you or any of your staff would be inerested in speaking with either Inez, or Penny Napke, the programmer here who is developing the info. ret'l. package (based on the Content Analyzer subsystem) for our use in H.Q. planning. I know we talked about this possiblity when we last met (first met), and I'd like to know if you feel this would be an appropriate time.
Inez and I would be interested in any othe ideas you might have relating to the exchange of information between our two groups. I think we all recognize that (whatever its shortcomings) we have a pretty valuable tool here, and we're interested in exploring its potential for putting groups such as yours and ours in closer touch.
Look forward to hearing from you.

MIKE 1-OCT-74 10:04 31107

copy of Bedford to Ruggles memo re coop'n between groups

Well, Inez Mattiuz is now fully debriefed after your Architect's workshop on the West Coast; she related how a number of you arch's, were less than enthusiastic about the support provided to date by ARC.

She also mentioned that the arch's, were planning to get together informally to exhange information and opinions regarding use of the system, and I agreed with her that this seemed to be a fairly promising alternative given the current state of affairs.

In this context, I was wondering if you or any of your staff would be inerested in speaking with either Inez, or Penny Napke, the programmer here who is developing the info. ret'l. package (based on the Content Analyzer subsystem) for our use in H.Q. Planning. I know we talked about this possiblity when we last met (first met), and I'd like to know if you feel this would be an appropriate time.

Inez and I would be interested in any othe ideas you might have relating to the exchange of information between our two groups. I think we all recognize that (whatever its shortcomings) we have a pretty valuable tool here, and we're interested in exploring its potential for putting groups such as yours and ours in closer touch.

Look forward to hearing from you.

1

JLM 3-OCT-74 06:34 31109

insw

o This is a subset of on of the nsw reports,for your info,I have to journal it to free up some space.

insw

THE NATIONAL SOFTWARE WORKS 1

INTRODUCTION 2

The production and maintenance of large programs is still an outrageously expensive activity. The costs are not only high, but also difficult to predict or control. Aside from the manifest payoffs derived from the use of compilers and (some) operating systems and a certain amount of improvement experienced by programmers who code interactively, it is not at all clear that the last twenty years of research and development in programming technology have made any serious dent in the problem. 2a

This situation is particularly interesting in the light of a general suspicion that, in principle, the problem ought to be eased by the creation of better software to support the program production and maintenance process for surely a great deal has been spent in the effort to invent just such software. The reasons for our failure are arguable and a variety of hypotheses have been put forward: 2b

- that the necessary tools -- or, at least, many of them -- exist in the research centers but are not being effectively delivered to the practical programming community 2b1

- that feedback from the user community has insufficient influence on the research laboratories, so that research emphasis if unrelated to user needs 2b2

- that the necessary tools exist, but are diffused over a variety of hardwares in many physical locations; the problem is that of difficulty of access. 2b3

Each of these hypotheses -- and the list may readily be extended -- doubtless contains a certain amount of truth, and collectively they surely suggest that dramatic improvements in the way programs are built are less likely to come from marginal improvements in present tools (or the invention of some magical new one) than from better methods of tool access and delivery, and better communication between research laboratory and end user. 2c

The idea of a National Software Works (NSW) on the ARPANET [1] arose fairly naturally from these considerations. If some number of end users were put on the network, and enough additional off-the-shelf software were brought up on the network to supply a complete set of conventional tools -- compilers, documentation aids, debugging systems, etc. -- for normal program development work, some useful results might be expected to follow: 2d

* The user would immediately have more convenient access to standard tools unavailable on his own hardware (or seldom available if his hardware is often tied up running production). 2d1

* The user would find it easy to access novel tools in use at research facilities presently on the network, but not otherwise available to him. 2d2

* Contact between the research laboratories and the user community would naturally improve. 2d3

In sum, the NSW might both immediately improve the present situation of the user, and in the long term, provide an effective vehicle for the communication of need from user to researcher and of responsive tool from researcher to user. 2e

It was soon recognized, however, that a view of the NSW as a mere lash-up of tools which happened to reside on the ARPANET would be extremely short-sighted. The fact that all programmer contact with tools would pass through a common communication mechanism with immense computing resources created a golden opportunity for the study -- and perhaps control -- of the whole process of large program creation and maintenance. This thought was particularly attractive in the light of our feeling that one of the most weakly supported areas in the production and maintenance process is project management: the absence of any tool which keeps track of what is going on, relating particular programmer activities to each other or to the overall picture, appears on the face of it rather a bad idea. 2f

To clarify the sort of support we have in mind and to suggest its influence upon the NSW design, we will digress briefly to talk in general about the program production process. 2g

THE PROGRAMMING PROCESS 3

In the production of a large program, numerous programmers cooperate in a venture whose end product is, in some sense, a single entity. In the course of their work toward this goal, programmers prepare, edit, and manipulate a very large number of pieces of "text" of various types: routines in a programming language, data descriptions, structured data objects, modules of object code produced by a compiler, assemblages of such modules linked together by a link editor, items of program documentation, and so on. 3a

To the degree that all of these types of text are either machine-processable or machine-producible, it is reasonable to say that they are all either prepared (and repaired) by programmers or

produced by "tools", by which we mean elements of support software invoked by programmers to operate on pieces of text. 3b

The number of such pieces of text which come into existence in the course of a large project can be astronomical, and even the number in some kind of active status at a particular time is likely to be huge. It ought to be clear that any absence of control over this large and shifting inventory material is an invitation to confusion and the almost total absence of any support software for "inventory control" might have something to do with the high and uncontrolled cost of program production (and perhaps something to do with our difficulties in figuring out what we are doing wrong). 3c

Suppose by contrast that the total inventory of text pieces were explicitly regarded as one logically integrated data base -- the Project File -- and that some piece of support software were charged with the responsibility of managing that data base. This piece of software - for the moment, let us call it the File Manager - would, of course, keep books on the contents of the Project File. These books would include not only the character and status of each item in the Project File, but also its relationship to other items in the File (that A is a later version of B, that C is the object code module corresponding to COBOL text D, and so on). 3d

It should be obvious that, if we have designed the books correctly and arranged matters so that they are always kept accurately and completely, they provide the data crucial to any serious attempt by management to explore or control what is happening in the project. 3e

It is, of course, essential to any interesting use of the project books that they be always complete and correct, that there be no path on entry to the Project Files unguarded by the File Manager. This suggests strongly that an individual programmer's use of his tools -- at least when that use yields a non-transitory (filed) result -- must always be reported to (and, perhaps, controlled by) the File Manager. 3f

To arrange matters so that this requirement is met is extremely difficult when the support software designer is confined to the resources of a particular local hardware: to keep the File Manager and its books effectively on line at all times may be insupportably expensive. Indeed, if a projects development work is performed on several computers with no communication among them, it may be logically impossible to create a reasonable File Manager. Thus, it is not surprising that there has been no serious attempt to provide a facility of the sort we have described: at least the naturalness, if not the feasibility, of

the idea depends on a unification and scale of computing resource found only in gigantic machines or in networks. 3g

THE NSW ENVIRONMENT 4

Against the background of our feeling that serious progress in rationalizing large program production will come less from the polishing of particular tools than from a frontal attack on the issue of improved access to tools and centralized management of the vast inventory of text floating around a large project, the logic of our strategy for the National Software Works becomes easy to see. 4a

- First of all, it is our intent to put a projects programmers on-line to the ARPANET. This has the immediate effect of giving them access to many tools unavailable on their own local hardware. 4a1

- Second, we will supply interactive editing packages, both a general text editor and editors which "speak" one or two common programming languages; the effect of such tools in facilitating program preparation and modification is too well known to require any defense here. 4a2

- Third, projects will be able to store these files on very inexpensive on-line mass storage devices (the Datacomputer (2)). This should relieve a considerable part of a projects local off-line file maintenance problems, and facilitate load-sharing, when the projects local computer is busy. 4a3

- Fourth, a File Manager will be always on-line monitoring the content and structure of the projects files and keeping the books up to date, as text pieces are created and manipulated. 4a4

The presence of the first three facilities will permit the project to conduct its business more or less as it does now (using the same languages, the same tools, etc.) with certain improvements in ease of tool access and foreign hardware access, editing, and file management. In addition, the project may, at its option, experiment with the use of different tools scattered around the network. 4b

The fourth facility opens the door to some genuinely new ways of controlling projects in the future. To begin with, a fairly powerful query system will be provided to answer questions about any filed entity: what it is, where it came from, what other entities depend on it, etc. Later we will introduce a variety of experimental tools for project control which use the File Manager's books as their primary data or use the fact of the File

Manager's existence as their means of invocation (after all, the later provides a single control point "awakened" every time anything interesting happens). Here are some proposed tools: 4c

* Project Status Reporter: This relates the present status of the files to the overall project plan (in machine-readable form), identifying bottlenecks, critical paths, etc. 4c1

* Project Accountant: This produces reports on the frequency and cost of various patterns of activity interesting to project management. 4c2

* Policy Enforcer: Everybody in Section A must use the same version of function X; no programmer may link up two routines until each is adjudged debugger by a section manager; no programmer may start debugging until all his code is written; no programmer may write any code for phase 2 of the project until he has written all his code for phase 1; no programmer may start writing a new routine until his last is documented. The above list of (rather inane) policy dicta are meant to suggest a large family of more reasonable policies which might apply to some or all programmers at various phases of a project. If a plausible way of expressing such dicta in machine-readable form can be developed, it is no great trick to devise a tool which is invoked by the File Manager to verify that the present action of some programmer is consistent with policy, so that the action may be inhibited or permitted accordingly. 4c3

The use of such new tools by the project would 4d

of course, be optional. In any event, the research community can make use of such tools to collect the data it needs to discover what makes program development and maintenance so expensive. 4e

SUPPORTING TECHNOLOGY 5

Virtually all multi-program operating systems have attempted to create a suitable programming environment by providing a set of tools. Some merely provided a library from which tools could be selected one at a time by the programmer. Others, like MULTICS [3], CP-67 [4], VS [5], and TENEX [6], have provided an on-line environment for program building and debugging. 5a

All of these systems have been built on a single computer and this has severely limited their capability to provide the type of environment described in the previous section. In fact, until recently a combination of several such hardware and software technical problems existed which prevented the conception and

implementation of this type of environment. These problems and their solution in the NSW are given below. 5b

1. Single machine implementation (all tools provided had to exist on the same machine): Computer networks, such as the ARPA Network [1], have established a communication mechanism whereby cooperating programs in different machines can function together as a single system (the technical basis for eliminating these problems are provided by computer networks, centralized mass storage, the Programmer's Interface [7], ACTORS [8], and Execution Machines (see SYSTEM DESCRIPTION section below). The Programmer's Interface has utilized this net technology to create an on-line programming environment combining tools which run on different machines. 5c

2. Non-integrated "tool-at-a-time" systems: current systems either segregate their tools into non-interacting components which are invoked one at a time or else provide highly complex integrated versions of these with the interactions between them built into the systems themselves. The type of programming environment we envision requires that actions or events in one part of the system permeate throughout the rest to maintain consistency and coordination between the component parts. The concept of ACTORS, by externalizing and removing the control and communication between the component parts, greatly simplifies constructing an integrated and coordinated system. 5d

3. Machine Independence: although tools running on different machines may be integrated into a single one the technology does not exist to run a single program on several different machines and obtain the same results. Therefore, software being produced must be executed and tested on the machine for which it is intended to run in production mode. Thus, if the software environment is to be used to produce programs for more than one machine, each of these must be hooked in through the computer network and a small portion of the system replicated on each Execution Machine to provide for translation and run-time monitoring capabilities. The rest of the software environment is common and can be shared independent of the machine for which execution is intended. 5e

4. Language Independence: currently, if software is to be produced for more than one language then the tools must either be duplicated in separate and distinct integrated programming environments, or else available in a non-integrated tool-at-a-time mode. The Programmer's Interface has shown that many of these tools are language independent or only slightly language dependent and has demonstrated how such tools can be extended to handle a wide set of programming languages. It utilizes the programming

tools (editors, file systems, debuggers, Programmer's Assistant [9], etc.) developed for one language (LISP [10]) for the development of software in other languages (e.g., ECL [11]). It has established interface requirements for other languages which would greatly reduce the effort required to transform these from simple interactive programming languages into an extensive programming environment. 5f

5. Economics: in addition to the costs of creating an appropriate programming environment, addressed above, there are several economic factors which currently limit the use and utility of existing programming environments. Most machines are sized for their production requirements not their development ones. Hence, typically they do not contain enough mass storage for the files that would be required in an on-line environment, nor enough memory to support both the code being developed and the tools for that development. 5g

Additionally, access to the system is limited by the priorities of the production work load. Networking and economies of scale provide solutions by providing access to a system specifically designed and sized for software development and on which no production workload exists. Changes would be based on usage and development costs for the system spread over a much wider community of users because of the language and machine independence aspects of the system. In addition, very cost effective mass storage can be provided (by the Datacomputer [2]) which provides a trillion bit on-line memory at a cost of about a dollar per megabit per year. 5h

SYSTEM DESCRIPTION 6

The hardware for the NSW, shown in Figure 1, consists of three logical components interconnected by the ARPA Network: The data computer, composed of a trillion bit store and a file management system in the Mass Storage component. The Execution Machine component is a set of machines responsible for running the program being developed and for collecting data on its execution. For each program being developed, the Execution Machine chosen is automatically the same as the production machine for that program. Thus, during development, a program is executed on the same (actually a copy of the) machine it will run on during production. This mechanism eliminates all machine-dependence compatibility issues at the cost of replicating the execution software in each machine for which this capability is desired and the cost of having that machine available in the NSW. On the other hand, it provides the great advantage of allowing the final component, the Interactive Machine (or machines), to be independent of the choice of production machine, thereby allowing it to handle a wider set

of implementation efforts. This component contains most of the system's software and provides all of the facilities of the NSW except those described above. 6a

The ARPA Network not only interconnects the NSW components, it also provides access for users to the system and supports a variety of terminals. However, the NSW will be oriented towards the use of high capacity video terminals. 6b

Although the system is distributed across the ARPA Network, it is organized so that neither the user nor the component software modules are aware of this. The user sees a single integrated facility. The mechanisms described in the Framework Section enable modules either locally or remotely connected to communicate without knowing each others precise location. 6c

TOOL INTEGRATION 7

Our discussion so far has concentrated on issues of tool access and integration of project data management; we have not yet talked about tools themselves. It should be clearly understood that it is not the intention of the NSW to go into massive tool production. Initially, standard tools will be brought up on the various machines in the network, and novel ones will continually be brought up by the research organizations as they have been in the past. In time, improved versions of old tools or quite new ones may be developed specifically for the NSW, as the user's needs become clearer, but there is no present plan for major activity along these lines. 7a

This reliance upon tools which already exist - or will develop - outside the NSW creates the obvious problem of integration of such foreign objects into the NSW. Here there are a number of issues: 7b

* Arranging communication between the tool and the rest of the system (the Project Files, the local console, etc.). 7b1

* Providing standard linguistic conventions for programmers to use when talking to any tool: Surely the highest level command language utterances which invoke tools should be standard over the whole tool population. Whether lower level transactions with various tools can be standardized at reasonable expense is debatable. 7b2

* Providing, for each tool, on line instruction in its use. 7b3

* Constraining tool contact with the Project Files to be consistent with the File Manager's role as infallable guard. 7b4

Certain of these problems must be met squarely: thus, the NSW will enable communication to happen, and will provide a consistent high-level command language. Others may be impractical to solve in any complete or consistent way: thus some tool may have so idiosyncratic a pattern of on-line communication that it would be prohibitively expensive to distort it to what might be called a standard form; on-line instruction might plausibly be quite complete for a small, unusual tool, but we have no intention of demanding a while CIA course in COBOL programming as a necessary accompaniment for a COBOL compiler. 7c

The last issue raised may deserve some discussion. The point is that every effort to deliver filed text to a tool or to a file text produced by a tool be subjected to whatever inhibitions the File Manager may then be authorized to impose and particularly by the requirements that enough information be suppplied to permit the books to be kept up to date. 7d

Ideally, foreign tools should be "naturalized", by which we mean that they should be so modified that this information all gets transmitted behind the programmer's back, through negotiation between the tool and the File Manager: thus a tool might accompany each piece of text produced by a standard chunk of descriptive information. In practice, the modification of some tools may be impractically expensive; in this situation, attempts to file tool output will invoke a standard conversational program which extracts the necessary information from the programmer and blocks filing until it is sufficiently well informed. In effect, we will naturalize tool outputs, if it is too expensive to naturalize the tools themselves. 7e

FRAMEWORK 8

This naturalization and integration will be effected by the framework which is designed: to allow the system to be quickly put together, to obscure the difference between local and remote communication, and to provide a mechanism for smooth, long-term growth for the inclusion of new facilities and improved cooperation and coordination among the tools within the NSW. This framework consists of two components. The first is a slight extension to the traditional subroutine invocation mechanism. It enables subroutine calls both within the same machine and across the ARPA Network to another machine. This mechanism determines which type of linkage to perform, utilizing the name of the routine being invoked to look up its location. Using this location determines which type of linkage should be performed. 8a

The second mechanism, and Executive, is more complex and is based on a combination of Actors [8], Ports [12], and Co-routines [13]

(see especially Ref. [8]). The Executive acts as an exchange between what is happening and the modules that need to be informed. The Executive, when notified that an action has occurred (or is about to occur), is responsible for invoking (using the mechanism described above) all those modules within the NSW that need to know about this event. 8b

Each of the modules within the NSW can be thought of as a self-contained asynchronous unit which utilizes these communication mechanisms to keep informed about events which affect it and to announce its behavior to those concerned. Within this framework each module has four responsibilities: to identify those actions that it performs; to notify the Executive when one of these occurs (or is about to occur); to supply appropriate information to the Executive for interpreting this action; and to request notification from the Executive of other actions of concern. 8c

This control structure is highly asynchronous, being driven by the actions that occur and the modules that need to be informed. Although it is specified in an interpretive form, the control structure can be "compiled" into normal intermodule invocations. The important point is that each module does not need to know what other modules are affected by its actions, but only has to agree to announce those actions. 8d

Initially the NSW will be composed of large modules with few, if any, actions identified. As such, it will operate in a largely conventional manner without much cooperation between the modules. Over time, the coordination and cooperation between the modules will be gradually tightened through the replacement of these modules and the incorporation of new ones that identify and report more of their behavior and which utilize the framework to keep informed of the behavior of other modules. In this way, the NSW can smoothly evolve toward a more unified and comprehensive set of capabilities. 8e

ACKNOWLEDGEMENTS 9

The design reported here is the result of an extensive committee effort spanning several months and involving an on-going network conference as well as frequent meetings. We would like to acknowledge the effort and contribution to the NSW of these committee members: 9a

Barry Boehm (TRW) 9a1

John Brown (TRW) 9a2

Michael Busch (CSC) 9a3

F. J. Corbito (MIT) 9a4

Peter Deutsch (XEROX PARC) 9a5

Jerry Feldman (Stanford) 9a6

Cordell Green (Stanford) 9a7

J. C. R. Licklider (MIT) 9a8

Tom Lippiatt (Rand) 9a9

Barbara Liskov (MIT) 9a10

Richard Watson (SRI) 9a11

Clark Weissman (DaSDC) 9a12

We would also like to acknowledge the inputs from the following people who took the time and effort to enlighten us on their software production problem and requirements: 9b

William Clark (NAVSHIPS) 9b1

L/C Robert O Keefe (USAF-ESD) 9b2

Major Harold Arthur (USAF-ESD) 9b3

Norman Glick (NSA) 9b4

John Mott-Smith (USAF-ESD) 9b5

Major Zara (USAF-ESD) 9b6

REFERENCES 10

(1) Roberts, L. G., Wessler, B. D., "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, Vol. 36, 1970, pp. 543-549. 10a

(2) Data computer Software Architecture, Data computer Project Working Paper, 5 Febuary 1972, Computer Corporation of America. 10b

(3) Corbato, F. J., and Vyssotsky, V. A., "Introduction and Overview of the Multics System," AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 185-196. 10c

insw

(4) International Business Machines Corp. 1969 (June), CP-67/CMS, Program 360D-05.2.005, Hawthorne, New York. 10d

(5) Auslander, M.A., Jaffee, J. F., Scherr, A. L., and Birch, J. P., "Functional Structure of IBM Virtual Storage Operating Systems," IBM Systems Journal, Vol. 12, No. 4, 1973, pp. 368-413. 10e

(6) Bobrow, D. G., Burchfiel, J.D., Murphy, D. L., and Tomlinson, R. S. (Bolt Beranek and Newman Inc., Cambridge, MA), TENEX, A Paged Time Sharing System for the PDP-10, August 1971, 29p. 10f

(7) Balzer, R. M., A Language-Independent Programmer's Interface, USC/Information Sciences Institute RR-73-15, November 1973. 10g

(8) Hewitt, C., Bishop, P., and Steiger, R., A Universal Modular ACTOR Formalism for Artificial Intelligence, 3IJCAI, 1973, pp. 235-245. 10h

(9) Teitelman, W., Automated Programming - The Programmer's Assistant, FJCC 1972, pp. 917-921. 10i

(10) Teitelman, W., Bobrow, D. G., Hartley, A. K., and Murphy, D. L., BBN-LISP TENEX reference manual, BBN Report July 1971. 10j

(11) Wegbriet, B., "The ECL Programming System," FJCC 1971, pp. 253-262. 10k

(12) Balzer, R. M., Ports - A Method for Dynamic Interprogram Communication and Job Control, The RAND Corporation, August 1971. 10l

(13) Conway, M., "Design of a Separable Transition-Diagram Compiler," CACM, Vol. 6, No. 7, July 1963, pp. 396-398. 10m

IMM 3-OCT-74 11:07 31111

Evaluation of Singer's 'The Social Functions of the Telephone,'
October 1/74

On the whole it is a neat sort of a document - full of all sorts of interesting stuff of things like party-lines, yellow pages, weather services, crank calls etc.
Also, there's a lot of statistical data of importance to somebody interested in describing average phone use, but little use to most (no. of phones in the bedroom, police reaction to reported crank calls, and no. of calls to a gov't office, etc.
Actually, that second category is sort of interesting, in the sense that each individual item relates to a potential money-making service.
I would have liked to see that the document commissioning this study, to see if it was less academic in nature. The document itself would be more valuable if they introduced some relationships other than purely telephone-oriented ones. (i.e. it is difficult to tell what type of people he was talking to, whether they were real or not). The report needs a different perspective to flash it out, make it more substantial. The text does little more than reproduce the data with appropriate verbs were needed.
The report seems "down" on the phone company in the few instances where the authors' opinions do show through.

pg 155 - public would demand more of company if they knew it was not government controlled (not clear if sample is solely Bell Canada territory).

pg 158 - "surprising" that over 50% L.D. rates were "just right"

pg 163 - "only 35 percent" would pay $.25 for a weather call - "only" 35 percent ???

I think it is a neat document that we might use as a source book for a lot of stuff we do; I presume Bell already has much of this data (from a more reliable sample, too!)

1

DLS 3-OCT-74 09:54 31112

Numbers and NLS-8

I think that the current design for moving copying etc numbers is BAD. One often wants to move numbers etc around that are mixed in with text. I think it should be changed...back to what it was in old NLS. 1