MAPPER

software run designer's reference

level 31R1

SPERRY

UP-9662.2

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry representative.

Sperry reserves the right to modify or revise the content of this document. No contractual obligation by Sperry regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry.

PRINTED IN U.S.A.

1. MAPPER Runs; This Manual and Its Conventions

1.1. What Is a MAPPER Run?

A MAPPER run is a sequence of control statements based on MAPPER functions that specify step-by-step instructions for generating reports or results or for executing other applications. You enter these MAPPER run control statements in a run control report (see Appendix C for some examples).

MAPPER runs efficiently execute sets of MAPPER functions. Runs are especially appropriate for repetitive processing, since they provide both report generating as well as automatic data base updating capabilities.

You can make logical decisions based on variables or results (e.g., jumping, branching, and decisions based on data content). You can also design tutorial MAPPER runs that run users can either interrupt to display data on the screen or can control manually, or both.

You can format reports in your MAPPER runs to suit your needs.

are you ready?

To design efficient MAPPER runs, you must have a thorough working knowledge of MAPPER manual functions. Before attempting to design a run, your MAPPER coordinator must determine your qualifications: have you attended a MAPPER run design class and do you have enough experience with MAPPER software?

Until you're a registered MAPPER run designer, sign on as JDOER to practice writing runs.

Use the EXAM run to check your MAPPER run-writing skills online.

1.2. How to Use The Manual

This is a reference manual. You should be familiar with the MAPPER manual functions presented in the MAPPER Software Reference, UP-9193 (see Preface). Ideally, you should feel quite comfortable using MAPPER software before attempting to write your own MAPPER runs.

Here's what you get in this manual:

- The rest of this section explains the style conventions used in the manual. Read this part carefully to save yourself a lot of time and worry.
- Section 2 tells you how to formulate MAPPER run control statements.
- Section 3 introduces the different types of variables with many examples, and includes some MAPPER run design aids specific to variables. This section ends by introducing reserved words—always captured with variables. Appendix B has the complete table of reserved words and an example using them.
- Section 4 presents online run design aids in the form of MAPPER runs.
- Section 5 tells you how to handle reports and results in your run, and how to design and register a run.
- Section 6 presents the MAPPER run control statements alphabetically. Be sure to use the handy MAPPER Software Summary (UP-9196 [see Preface]) once you're familiar with the special information about each function or statement.
- Section 7 is a tutorial of six exercises. These exercises take you through the MAPPER run-writing process step by step. You should read through these exercises first. Then, when you're ready, you can do the exercises step by step just as they're outlined in the manual; or you can experiment as you go by changing little things here and there to see what happens. If you follow the step-by-step procedure, you should get the same results we show in the manual. If you want to experiment and deviate from the presentation, you should pick up new insights as you go.
- The appendices summarize statements, options, and reserved words. They also show some sample MAPPER runs and offer suggestions to make your own runs more efficient.

1.3. Style Conventions

We want you to understand this manual and find MAPPER software easy to use, so we use certain stylistic devices (commonly called "style conventions").

capital letters

These items appear in capital letters (also called uppercase letters):

- MAPPER functions, both the spelled-out name of the function and the call (e.g., CHANGE [CHG] and SEARCH [SRH])
- MAPPER runs (e.g., FCC run, RUNA run)
- MAPPER files (e.g., MAPER1 file)
- Reserved words (e.g., INPUT$)

In the text, user entries also appear in uppercase letters. However, unless otherwise indicated, you can make all entries in lowercase letters (i.e., you don't have to shift). For example, for ABCD, simply enter:

```
abcd
```

italics

Italics (type that slants to the right) appear for a number of reasons:

- The italicized letter N (*n* or *N*) stands for a *numeral* (*nn* calls for two digits; *nnn* for three, etc.); other italicized letters (e.g., *x* and *y*) usually call for other information explained in the accompanying discussion.
- In date formats, *y* or *Y* means "year"; *m* or *M* means "month"; and *d* or *D* means "day."
- Some words and phrases in the text are in ***bold italics*** to draw your attention to important points.

run control statement syntax

The format of a MAPPER run control statement has these things in common:

- The call is capitalized (e.g., **CHG**) because it is constant. Type it in lowercase letters if you wish.
- Fields and subfields are italicized whenever they call for *variable* data, i.e., whatever information you may legitimately enter, depending on the explanation that follows the statement.
- Fields are separated by two spaces in the text for easier reading; however, you need type in only *one* space.

- Fields or subfields enclosed in brackets are optional. In this example, *field2, subfield1, and subfield2* are optional:

  field1 [*field2* *subfield1,subfield2*]

  Whenever you make an entry in an optional subfield, you *must enter all intervening commas* (e.g., `@aux,2,b,2,999,cop,,,y,,2` .)

- Braces around an item mean that you may choose from among the items listed, e.g.:

  { *item1* / *item2* } or: { *item1* | *item2* }

tab characters

A special character called a quadrate (□) always represents a tab character. You cannot see tab characters on the screen unless you place the cursor over it, in which case the cursor blinks.

color/shading/boxes

User input in screen illustrations, as well as run control statement formats are in color.

In Section 7 (the tutorial exercises), some parts of certain examples are shaded or boxed in (sometimes both) to highlight points made in the discussion preceding them.

"transmit" and "resume"

"Transmit" means "press XMIT ."

"Resume" means "press F1 (you must use the UPPER FUNCTION key also); or enter RSM and press XMIT ."

RIDs and reports

RID means *report identifier* and is a number. This manual identifies a report by its RID number and form type:

RID = report identifier	RID 1
Report = RID + type	Report 1c

examples

Some examples appear in uppercase, some in lowercase. Some examples appear with brief comments similar to comments you might enter in your run control report, for example:

```
@dlr,2,b,3      delete report 3b in mode 2
```

These comments may start anywhere after the space-period-space that follows a run control statement.

2. Formulating Run Control Statements

2.1. Run Control Statement Format

Here's a typical MAPPER run control statement:

@ label:statement,mode,type,rid options characters line-type,parameters variables

This @SRH (SEARCH) control statement uses data from mode 2, alphabetic form type C, RID 1:

```
@7:SRH,2,C,1 D 41-1 '□',O/R,9 .
```

likewise, with variables:

```
@SRH,2,C,1 D 41-1 □,O/R,9 V415,V516 .
```

where:

@	control character
7	label (optional)
SRH	statement call
2	mode number
C	alphabetic *or* numeric form type of report
1	report identifier (RID)
D	D option (Delete search information lines)
41-1	column-character positions to process: column 41 for one character position

□,0/R,9 line type and parameters:

- □ tab character: edited line type (use apostrophes if you want to remind yourself of invisible tab character)
- 0 parameter
- / slant: multiple parameters (for when you would manually use two or more lines below the function mask)
- R range search
- 9 parameter

V4I5 5-character integer variable named V4, which captures the number of finds

V5I6 5-character integer variable named V5, which captures number of lines searched

valid statements and error messages

MAPPER software considers any line in a MAPPER run that begins with an at sign (@) a control statement line--the line must have a valid statement. If the MAPPER processor finds something invalid in a line, it responds with an error message like one of these:

```
▽UNABLE TO FIND ALL THE FIELDS REQUIRED▽

▽THAT CONTROL WORD IS NOT VALID▽
```

formulating control statements

Follow these guidelines when formulating MAPPER run control statements:

- Type an at sign (@) in column 1.
- Enter multiple control statements on one line and separate them with spaces. Use just one @ per line.
- Terminate a line with a space-period-space (Δ.Δ), or by beginning the line with an at sign-period (@.) or an at sign-label-period (@ *label*.). You can begin comments (which are useful for analyzing MAPPER runs) anywhere after the space-period-space on any line.

  These statements terminate the line; the MAPPER processor ignores statements after them on the same line: @ESR, @DSP, @OUT, @RRN, @RSR, @RTN, @RUN, and @WAT.
- Specify columns in the *column-characters* field from lower- to higher-numbered columns, e.g.:

  2-2, 16-4, 35-6

  except for @RDC, @RDL, or @RLN statements.
- Define *columns* only once in the *column-characters* field.
- Terminate fields with *spaces*. A comma does *not* terminate a field.
- Separate subfields with *commas* including blank subfields.
- Subfields that define character positions to process must correspond to control parameter subfields. The subfields must be equal in number to and in the same sequence as the character position subfields to which they correspond.
- You may use variables in fields and subfields.
- *Don't* exceed 20 variables in a parameter subfield.
- Enclose fields or subfields that require significant spaces within apostrophes (' ').
- The required fields and subfields for a control statement vary from statement to statement. Include all required fields. Enter two apostrophes ('') if you're not entering options in the *options* field.
- To get the *literal* representation of variables and special characters in the output area, enclose them in apostrophes (e.g., '/', ';', 'V1').
- Whenever you use a comma for something other than a subfield separator, enclose it in apostrophes (e.g., a comma in the R option for FIND or SEARCH—**R2',5**).
- In control statements that require double masks or two reports, designate the issuing report first, the receiving report next.

See Appendix D also for more ways to improve your MAPPER runs.

2.2. Labels

You can identify any control statement line in a run with a label. Use labels:

- at the start of a control statement to match a label specified in the *label* subfield of another control statement;
- in logical @IF GTO statements; and
- to identify sections in a run.

A comment in a run might, for example, say:

```
check v1 at label 5 for proper information at your site
```

format

@*n*: *(followed by a statement)*

or:

@*n*.

where *n* is a line label number from 1 through 199.

Each label must be unique; i.e., duplicates are not allowed.

2.3. Special Characters

Use these special characters in MAPPER runs:

- the semicolon (;) as a field delimiter;
- the slant (/) to indicate multiple parameters (e.g., in a range search); and
- the reverse slant (\) for continuing a MAPPER run control statement on the next line.

semicolon (;)—field delimiter

In an @ART or @CAL statement, a semicolon separates expressions. For example:

```
@ART V2+V3;V4*4 V10F12.6,V1119 .
```

and:

```
@cal,16,f,41,6,5 r.01 10-2,21-13,35-13,49-13 □,d,a,b,c ; \
total=5000-0;a=total/5;tot=tot+a;b=tot;c=total-b+0;d,r-=line .
```

In an @IF statement, a semicolon controls more than one decision on the same line. For example:

```
@if v1 = 3 gto 9 ;if v1 < 6 gto 8 ;if v1 = 6 gto 10 ;gto end .
```

In colon (:) lines (see 3.2), a semicolon establishes more than one variable. For example:

```
:V1H3□123;V2H4□1234;V3H3□AS1
```

slant (/)—multiple parameters

Use a slant to separate multiple parameters in a statement. For example:

```
@TOT,40,B,6,99 ' ' 2-3,8-15,40-3 *,+,=,+/*,,+ .
```

reverse slant (\)—continue statement

Use a reverse slant whenever a control statement is too long for one line. For example:

```
@MCH,40,B,6,40,B,77 'N' 2-3,8-15,40-3 *,1,2,A 2-3,\
8-15,60-3 *,1,2,A .
```

This statement spans more than one line. The reverse slant at the end of the first line causes it to be continued on the second line.

NOTES:

1. *You may use up to 640 characters in a control statement on multiple lines. The MAPPER processor counts all characters in the last line; i.e., it counts unused spaces.*
2. *For readability, use the reverse slant at the end of a subfield and, if possible, avoid starting a second or succeeding line with a space.*

3. Variables and Reserved Words

3.1. What Are Variables; Naming and Using Them

According to the Glossary, a variable is "a labeled entity that can assume different values." A variable's label is the letter V; its value is whatever you assign to it.

Name variables with a V followed by a number from 1 through 199.

Before using a variable, you must initialize it, assigning it a variable type, size (the number of characters it can have), and an initial value.

After initializing a variable, refer to it by its variable name—V and the number. For example:

```
V10
```

You can refer to parts (substrings) of a variable:

variable-name(position-characters)

where *position-characters* are the starting character in the variable and the number of characters. For example:

```
V10(1-3)
```

You can use another variable that has a number from 1 through 199 to name a variable. For example:

```
VV1
```

You can use another variable that has a valid number (depending on the type of variable) to specify the size of a variable. For example:

```
V1iV2
```

You can use variables in many places and for many purposes in MAPPER runs. As you become increasingly familiar with MAPPER run control statement syntax and special commands, you are better able to visualize where a variable might be useful.

Here are just a few of the places you can use variables in place of hard-coded data:

- In any field or subfield in control statements
- In the run's output area (see 6.7)
- As counters and checks for logical decisions (see 6.33)

The six types of variables are:

A Alphanumeric

H Hollerith (any characters)

I Integer (whole numbers)

F Fraction

S String

O Octal

Table 3-1 shows variables types, sizes, and limitations.

Table 3-1. Variables: Types, Sizes, Limitations

type	size*	example	contents & how @ART, @CHG, @IF statements handle
A	12	v10a12	Alphanumeric, special characters @ART: must have exclusively numeric characters. @CHG: for arithmetic, numeric characters produces numeric answers; changes letters and special characters to next letter in character set. @IF: treats spaces and some special characters (. + -) as zeros.
H	18	v10h12	Any characters. @ART: not allowed. @CHG: changes all characters to next character in character set, including numbers. @IF: characters must be identical to satisfy a true condition.
	16	v10i12	Integer (whole numbers), positive and negative. Positive numbers may be unsigned; i.e., they need not have a plus sign (+). Include the sign in the variable size—it is a significant character. @ART, @CHG, and @IF allowed.
F	18	v10f12.6 (e.g., 3.141593)	Fractional numbers, positive and negative. Positive numbers may be unsigned; i.e., they need not have a plus sign (+). Must have decimal point; fractional portion may be up to 10 characters. Include the sign and decimal point in the variable size—they are significant characters. @ART, @CHG, and @IF allowed.
S	132	v10s132	Alphabetic, numeric, and special characters, any combination (for a total of 2,016 characters in all string variables combined). @ART: not allowed. @CHG: not allowed. @IF: substrings up to 18 characters. Redefinable if new size same as or smaller than original size. If reinitialized to another type, still considered type S variable whenever functions performed against it.
O	12	V503	Numbers 0 through 7. @ART: not allowed. @CHG: for arithmetic, produces octal answers. @IF: considers number's decimal value, not its octal representation.

* *Maximum size.*

3.2. Initializing and Redefining Variables

Initialize and redefine variables with:

- an @LDV statement;
- a @CHG statement;
- another statement; or
- a colon.

using an @LDV statement

Using an @LDV statement is the most efficient way to initialize and load a variable, as in these examples:

```
@ldv v1i2=1 .              initialize v1 to 1

@ldv v1a1=a,v2i2=10 .      redefine v1 to A; initialize v2 to 10
```

using a @CHG statement

You can use a @CHG statement against all variables except type S variables. You can initialize multiple variables on one line with multiple @CHG statements, as in this example:

```
@chg v10a3 xyz CHG v11a5 abcde .  initialize v10 and v11
```

This example shows how to name a variable with the contents of another variable:

```
@chg vv10a3 xyz .  initialize a variable with a name (contents of v10)
```

Redefined variables lose their previous content, as in this example:

```
@chg v10a3 aaa chg V11a1 a .          initialize v10 and v11
@chg v10f4.2 1.23 chg v11i4 1234 .    redefine v10 and v11
```

initializing variables with other statements

Several statements initialize variables (e.g., @RDC, @RDL, and @RLN). Some functions (e.g., FND and LOC) place certain information in variables; some functions (e.g., ART and TOT) place values in variables. As you learn the functions, you discover these and other possibilities.

This example shows how to use an @RDL statement to place the data in column 2 for two characters in V10 and the data in column 5 for six characters in V11:

```
@RDL,2,B,1,6 2-2,5-6 V10A2,V11A6 .
```

This example uses an @FND statement to place the RID of the report where the find was made in V10 and the line number in V11:

```
@FND,2,B,,6,99 '' 2-2 □,SH V10A4,V11A4 .
```

This example uses a @TOT statement to place the number of lines processed in V10 and the sum of the values totalized in V11:

```
@TOT,2,B,2,99 E 32-6 □,+ V10A4,V11A8 .
```

initializing variables with a colon

You can initialize all variables by entering a colon (:) in column 1 of the run, the variable name in column 2, followed by a *tab character* and the initial value of the variable. Initialize other variables on the same line by inserting a semicolon before each variable, as in this example:

```
:v10a3□xyz;v11a5□abcde .   initialize v10 and v11
```

Be sure to allow enough character positions between the tab character and the next semicolon to consume the character positions of the variable. For example, when initializing a 12-character variable, leave at least 12 character positions between the tab character and a semicolon. If you leave too few character positions, you won't define the variables that follow the semicolon.

3.3. Changing the Contents of Variables

You can change a variable's content several ways, as in these examples:

```
@ldv v10a3=AAA .                  initialize v10 to AAA

@ldv v10=BBB .                    change v10 to BBB

@ldv v11i3=123 .                  initialize v11 to 123

@chg v11 v11 +1 .                 change v11 to 124

:v12s15□0123456789ABCDE .         initialize v12 to 0123456789ABCDE

@ldv v12(1-3)=v10 .               load the first 3 character positions of
                                  v12 with v10; v12 now equals
                                  BBB3456789ABCDE

@ldv v12(14-2)=v11(2-2) .         load 2 characters of v12 starting in
                                  column 14 with 2 characters of v11
                                  starting in column 2; v12 now equals
                                  BBB3456789ABC24

@ldv v13i1=4,v10=v12(v13-3) .     initialize v13 to 4; change v10 to equal
                                  the 3 character positions starting in
                                  column (v13=4) of v12; v10 now equals 345
```

3.4. Using Exponential Notation with Variables

Type A, type I, and type F variables may have numbers in exponential notation, as in this example:

```
@chg v1a12 12e5 + 1 . v1 equals 1200001
```

If the numbers in a variable get too large for the variable, the MAPPER processor changes the value to exponential notation if possible. For example:

```
@chg v2i8 12345678 * 10 . v2 equals 1.234e+8
```

3.5. Examples Using Variables

This @DSP statement:

```
@dsp,v1,v2,v3 .
```

is requesting that:

- mode v1 (i.e., the mode number in v1);
- type v2 (i.e., the form type in v2); and
- report v3 (i.e., the report number in v3)

be displayed.

Variables V1, V2, and V3 may have been initialized in any number of ways. Here are some possibilities:

- You could have loaded the variables earlier in the run so that if the data you want to display is someday moved to another mode or RID, you would need to change that information only once in your run control report—at the place where you initialized V1, V2, and V3. This is especially useful if the report is processed repeatedly in the run; and is much easier than changing every applicable control statement in the run.
- Perhaps you loaded the variables with reserved words that are user-, date-, time-, terminal-, or run-specific. For example, in the statement:

  ```
  @chg v1i3 emode$ chg v2i6 etype$ chg v3i4 erid$ dsp,v1,v2,v3 .
  ```

  the reserved words EMODE$, ETYPE$, and ERID$ pick up the mode, numeric type, and RID of the run control report itself. The above statement, then, displays the run control report.

- You might have written the run to pick up information from the screen--information entered by the run user. In this example:

```
Enter the location of the report you want & transmit

       mode□   , alphabetic type□ ,  RID□
@brk out,2,e,-0,2,3,1,1,y,,,i .
@chg input$ v1i3,v2a1,v3i4 dsp,v1,v2,v3 .
```

  V1 is initialized as the mode entered, V2 as the alphabetic type entered, and V3 as the RID entered on the screen; and the MAPPER processor displays the requested RID on the screen.

- Certain functions load variables automatically with pertinent information. For example, the statement:

```
@ldv v1i3=2,v2a1=d fnd,v1,v2 '' 22-3 □,'  1' v3i5 dsp,v1,v2,v3 .
```

  is executing a find across all RIDs in mode 2, type D, and loads V3 with the report number of the first find.

- Or, perhaps you loaded the variables with a combination of screen information and internal run loading (@LDV), as in this example:

```
Enter code to find & where to look (mode 2,type B RID)

                Status code□  ,   RID□
@brk out,2,e,-0,2,3,1,1,y,,,i .
@chg input$ v4a2,v3i4 ldv v1i1=2,v2a1=b .
@fnd,v1,v2,v3 '' 2-2 □,v4 ,v5i5 .
@dsp,v1,v2,v3,v5 .  v5=line number of find--start display there
```

- You can use variables as counters that the MAPPER processor increases whenever logically necessary, then later checks to see if looping should continue, as in this sequence of statements:

```
@ldv v1i3=2,v2a1=d,v3i4=1,v4i5=6 lzr,v1,v2,v3 v5i5 .  v5=nr.lines
@fnd,v1,v2,v3,v4,196 '' 22-3 □,'  3' ,v4 dsp,v1,v2,v3,v4 .  v4=line
@chg v4 v4 + 1 if v4 not > v5 gto lin -1 .
@196: .
No more finds.
@gto end .
```

  In this example, V4 is the counter for the line on which to start the find as well as the line number of the find, for use in the display. V5 is the number of lines in the report. As V4 is increased, the find process begins further into the report. When V4 becomes greater than V5, the entire report has been processed, and the run ends.

You can use loops and counters like these in many other ways. In the same example, you could execute a find across RIDs and display the correct RID at the appropriate line by using more variables and checks.

VARIABLE

3.6. VARIABLE Run—Testing Content of Variables

The VARIABLE run determines how variable types and input methods affect a variable's content.

To execute the VARIABLE run, enter:

VARIABLE

example: MAPPER response and value to test entered

```
THIS RUN SHOWS WHAT THE CONTENTS OF THE NONSTRING VARIABLES IN LIMITED
CHARACTER SET LOOK LIKE AFTER INITIALIZATION AND AFTER SOME OPERATIONS.
  ENTER ANY INITIAL VALUE -> 100.00
```

result

```
line▶ 1     fmt▶    rl▶-  ▮    shft▶     hld chrs▶  hld ln▶    ▶ ▿▿▿RESULT▿▿▿ ▶
.DATE 23 JUN 81  08:29:04     REPORT GENERATION     JDOER

VALUE IN VARIABLE WHEN INITIAL ENTRY IS <100.00>
  @CHG INVAR$ V4     . @CHG INPUT$ V1     . @CHG V2 V1 +1      . @CHG V3 V1 *2
 ================= . ================= . ================= . =================
 V4A6    = <100.00> . V1A6    = <100.00> . V2A6    = <   101> . V3A6    = <   200>
 V416    = <   100> . V116    = <   100> . V216    = <   101> . V316    = <   200>
 V4F6.3 = <100.00> . V1F6.3 = <100.00> . V2F6.3 = <101.00> . V3F6.3 = <200.00>
 V406    = <   100> . V106    = <100.00> . V206    = <N/A   > . V306    = <N/A   >
 V4H6    = <100.00> . V1H6    = <100.00> . V2H6    = <100.01> . V3H6    = <N/A   >
        ****** YOU MAY USE A RESUME (RSM OR F1) FOR ANOTHER VALUE ******
```

Resume to test another value.

VAL

3.7. VAL Run—Listing Variables, Line Numbers, and Labels in a Specific Run

The VAL run produces a list of variables and labels assigned in the run and their line location.

Since the VAL run examines *each* character position of the MAPPER run control statements, use it with discretion against large run control reports.

To execute the VAL run, first display the run control report for the MARK run (see Appendix C), and enter:

VAL

result

```
line▶ 1     fmt▶    rl▶- 1    shft▶     hld chrs▶    hld ln▶    ▶ ▽▽▽RESULT△△△ ▶
.DATE 03 JUN 81  09:21:43     REPORT GENERATION    JDOER
        VARIABLES AND LABELS IN RID 3 TYPE 0210
 LINE   VARIABLE
    9    V3H3
    9    V4H8
   14    V5H3
   14    V6H8
   14    V7S17

 LINE   LABEL
   17    @1

          USE THIS RUN WITH CARE. IT USED 53 IO'S AND 518 LLP's
                    ..... END REPORT .....
```

3.8. Reserved Words

A reserved word is a character string that is reserved for specific use in a MAPPER run.

To control data easily in your runs, you can initialize variables to capture reserved words with a @CHG statement.

For example, this @CHG statement initializes V1 as a 6-character type I variable to capture the numeric form type of the current result:

```
@CHG V116 TYPE$ .
```

Here are some common reserved words:

DATE1$	Current date in format *yymmdd*
INPUT$	Data from screen/source external to run; no strings
RID$	RID number of report being processed
SOE$	Start Of Entry (SOE) character
STAT1	Status word one (quantity or status)
STAT2	Status word two (quantity or status)
TYPE$	Numeric form type of current result (-0)

Appendix B lists all reserved words.

HELP

4. Online Aids

4.1. HELP Run

The HELP run gives you MAPPER run design information online.

To execute the HELP run, enter:

```
HELP RUN
```

MAPPER Software response

```
          ****   RUN INFORMATION   ****
                    ROLL ▶ ▮          (enter RUN target if known)
 TARGET                          EXPLANATION
============  ============================================================
ADD           APPEND REPORT
ADR           ADD REPORT
AID           RUN DESIGN AIDS
AQ            ANALYZE/ALTER QUEUE
ART           ARITHMETIC
AUX           AUXILIARY
BFN           BINARY FIND
BRK           BREAK
```

if you want more information about a specific item. . .

To select a specific item from the list, either:

- enter the target after ROLL; or
- tab to the target and transmit.

HELP

To get information about a specific section of the HELP run, enter:

```
HELP RUN,section
```

where *section* is either the actual function call or statement, or an abbreviation for other kinds of information (e.g., AID in the preceding screen illustration).

debugging a MAPPER run

You can also use the HELP run to debug your MAPPER run.

If an error message appears in the control line while you're executing a MAPPER run, enter:

```
HELP
```

and transmit to display more detailed information about the error.

After examining the information, resume to reenter your run control report at the line where the error occurred.

CC

4.2. CC Run—Displaying Horizontal Column Count Positions

The CC run gives a horizontal column position count for a form type.

To execute the CC run, first display a report in the desired form type, and enter:

```
CC
```

MAPPER Software response

The column numbers appear in lines 2 and 3 of the screen. Since the CC run is format sensitive, the column numbers are correct no matter which format you display.

```
1
        1111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
*ST.STATUS.BY. PRODUCT .SERIAL.PRODUC.ORDER.CUST.PRODUC.PRODUC. SHIP .SHIP .SPC.
*CD. DATE .IN.  TYPE   .NUMBER. COST .NUMBR.CODE. PLAN .ACTUAL. DATE .ORDER.COD.
*==.======.==.=========.======.======.=====.====.======.======.======.=====.===.
 IP 741224 LS BLACKBOX1 436767        84389 AMCO 741223 741224
 IP 741225 LS BLACKBOX1 436768        84390 AMCO 741223 741225
 IP 741219 LS BLACKBOX2 637071        84353 INTR 741218 741219
 OR 750110 LS BLACKBOX4               94754 ARCO
 SC 750110 LS BLACKBOX5 675281        97441 FEDS 750131
 IP 741222 LS BLACKBOX5 737582        84040 AMCO 741222 741222
```

Resume to redisplay the report.

To get a column count on a specific line, enter:

```
CC n
```

where *n* is the line number on your display terminal where the column count lines are to start.

Resume.

FCC

4.3. FCC Run—Examining Report Fields

The FCC run gives the field headers, the position of the first character in each field, and the size of each field.

To execute the FCC run, enter:

FCC [*rt*]

where *rt* is the RID and type of a report not on display.

result

The FCC run displays the report headers followed by the field locations and sizes:

```
line▶ 1▮    fmt▶   rl▶-     shft▶     hld chrs▶  hld ln▶    ▶ ▽▽▽RESULT△△△ ▶
.DATE 03 JUN 81  08:18:52     REPORT GENERATION     JDOER
. MODE  16, TYPE 0202, RID 2          ALPHA TYPE B     80 CHARACTER LINE LENGTH
*ST.STATUS.BY. PRODUCT .SERIAL.PRODUC.ORDER.CUST.PRODUC.PRODUC. SHIP .SHIP .SPC.
*CD. DATE .IN.  TYPE   .NUMBER. COST .NUMBR.CODE. PLAN .ACTUAL. DATE .ORDER.COD.
*==.======.==.=========.======.======.=====.====.======.======.======.=====.===.
 2-2       12-2         25-6   32-6   39-5  45-4 50-6   57-6   64-6   71-5  77-3
    5-6       15-9


                     ..... END REPORT .....
```

Take note of the column-character positions or print the result.

Resume if you want to redisplay the original report.

FORM

4.4. FORM Run—Displaying Statement Fields and Subfields

The FORM run defines run control statement fields and subfields. It fills in the syntax of the statements with abbreviated fields and subfields.

To execute the FORM run, first enter the statements in your run control report, and enter:

```
FORM
```

example: request and MAPPER Software response

```
LINE▶ form ▮  FMT▶   RL▶-      SHFT▶     HLD CHRS▶    HLD LN▶    ▶      LCS      ▶
.DATE 03 JUN 81 08:22:20  RID     75    03 JUN 81  JDOER
*RUN FUNCTION DATA: EXAMPLE OF USE OF THE RUN DESIGN AID 'FORM'              E0210
*=============================================================================.
@SRH
@SOR
@MCH
@DSP



                      ..... END REPORT .....
```

```
LINE▶ 1     FMT▶   RL▶- ▮    SHFT▶     HLD CHRS▶    HLD LN▶    ▶      LCS      ▶
.DATE 03 JUN 81 08:22:20  RID     75    03 JUN 81  JDOER
*RUN FUNCTION DATA: EXAMPLE OF USE OF THE RUN DESIGN AID 'FORM'              E0210
*=============================================================================.
@SRH,MD,TP,RD,LINE,LINEQ,LABEL OPT CHR LNTP,PRM VNFNDS,VNLNS   .
@SOR,MD,TP,RD OPT CHR LNTP,PRM   .
@MCH,MD1,TP1,RD1,MD2,TP2,RD2,LABEL OPT CHR LNTP,PRM CHR LNTP,PRM   .
@DSP,MD,TP,RD,LINE,TABQ,FMT,INTRM(Y/N),SH,MSG   .



                      ..... END REPORT .....
```

Fill in the fields and subfields with the appropriate information.

FORMD

4.5. FORMD Run—Getting Formats in Run Control Report

The FORMD run inserts the format of a run control statement on a line in your run control report.

To execute the FORMD run, first display a run control report, and enter:

FORMD

NOTES: 1. *Roll the line that has the call or entire statement whose format you want inserted to the top of the display (i.e., to the line just below the control line), being sure that the first character is an at sign (@).*

2. *Enter FORMD in the control line.*

To get the format of a statement that is not in your run control report, enter:

FORMD *xxx*

where *xxx* is the call *without an at sign (@)* (e.g., FORMD SOR).

MAPPER Software response

The MAPPER processor redisplays your run control report with an added line that shows the format of the statement in front of the line that has your call or statement.

MARS

4.6. MARS Run—Creating Statements In Run Control Report

The MARS (Make A Run Statement) run creates MAPPER run control statements and places them in a run control report. If you don't have a run control report, the MARS run adds one for you.

If you are using a series of manual functions repeatedly, use the MARS run to capture these functions in a run control report. You can call and execute this run with a name. The coordinator will register the named run for you.

To execute the MARS run, first display a run control report, and enter:

MARS

You get a menu of functions:

- tab to the function you want and transmit; or
- enter the function call at the top of the menu and transmit.

If you need help, enter:

- MARS,HELP

or:

- tab to a statement;
- enter ?; and
- transmit.

Resume to return to the menu.

The MARS run leads you through a simulated manual execution of the function. The MARS run writes the MAPPER run control statement for the function into the run control report at the first blank line.

RUNA

4.7. RUNA Run—Analyzing Your MAPPER Run

The RUNA run analyzes runs by identifying certain inefficient run design techniques. The RUNA run is not intended to be an absolute or total test of run design quality. Carefully review any indications of inefficient techniques identified in the analysis and try to correct them.

To get an explanation of the RUNA run online, enter:

```
RUNA
```

To execute the RUNA run, first display a log list of your run, and enter:

```
RUNA
```

See 6.47.

if you want more details. . .

To display a detailed explanation of the recommended corrective guidelines, resume to display the details appended to the result. Print the result if you think you might want to refer to it in the future.

call your MAPPER coordinator

Even if your run passes the RUNA run analysis satisfactorily, it may not be ready for use in production. Your MAPPER coordinator must still approve your run.

See Appendix C for an example of a RUNA run result.

LLN

lab label to go to if no report or form type exists

vln *variable-line-number:* variable to capture line number

example

```
@LLN,2,B,1,2 V113 .
```

This statement determines the number of lines in mode 2, type B, RID 1, and places that quantity in V1. If the report or form type does not exist, it goes to label 2.

@LMG

6.40. @LMG (LIST MERGE)

The @LMG statement extracts lines or partial lines from an issuing report and places them in a receiving report.

control characters

The receiving report must have these control characters:

~ =*cc*-*cq*[,*n*]	extract *part of line* starting at column *cc* for *cq* characters at line *n* beyond tab type line [line *n* must be 4 or less; to process the tab type line, omit the ,*n*] (e.g., ~ =2-4,3)
~ =0,*y*-*z*	extract *full lines* starting at line *y* beyond tab type line for *z* lines; [number of full lines that can follow a tab type line is unlimited] (e.g., ~ =0,3-4)
~ &*cc*-*cq*	extract *tab type lines* starting at column *cc* for *cq* characters and produce one line for each line merged (e.g., ~ &2-17)

run control statement format

@LMG,*iss-m,iss-t,iss-r,rec-m,rec-t,rec-r*[,*lab*]

where:

iss-m,iss-t,iss-r	mode, type, and RID of issuing report
rec-m,rec-t,rec-r	mode, type, and RID of receiving report
lab	label to go to in case of error

example

```
@lmg,2,f,50,2,f,51 .
```

This statement merges lines and partial lines from mode 2, type F, RID 50, into mode 2, type F, RID 51.

6.54. @OUM (OUTPUT MASK)

Use an @OUM statement to display a blank function mask, which you can use to load run parameter data into a variable.

Reserved word: INMSV$
Specify the variables to load with the reserved word INMSV$ before the @OUM statement, just as you specify the data for the reserved words INVAR$ and INVR1$ before an @OUT statement (see 6.55).

You should use string variables (type S), but you can use type A or type H variables if the data fits in them.

An @OUM statement returns the option line and up to four decoded parameter lines. The first variable initialized by INMSV$ has the entire option line. Each parameter line requires three variables:

- The *first variable* has the starting column numbers of the fields where the run user entered parameters. Each starting column designation in the variable is three characters long, right justified, and filled with zeros. The 3-character designations for all starting columns are packed to the left in the variable. For example, parameter entries starting in columns 2 and 45 in the mask yield a variable having 002045.

  Whenever you use a format other than basic, the column numbers are those defined in RID 0 for the format. For example, if format 1 displays a field that actually starts in column 126, the variable has 126 for the starting column number, *not* the column number where the field was displayed on the screen.

  Use a 120-character variable to hold the maximum 40 fields possible. The MAPPER processor fills unused character positions with blanks.

- The *second variable* has the corresponding field sizes, three characters each, packed to the left in the variable.

- The *third variable* has the entire parameter line expanded according to the format of the mask.

The run stalls until you transmit, then continues with the data in the specified variables.

OUM

run control statement format

@OUM,*m,t*[,*r,f*]

where:

m,t.r	mode, type, and RID of report that has mask
f	format in which to display mask and decode input (assumes basic format if not specified)

example

```
@CHG INMSV$ V1S80,V2S120,V3S120,V4S132,V5S120,V6S120,V7S132 .
@OUM,0,A,2,1 .
```

where:

@CHG INMSV$	statement to load INMSV$ with variables
V1S80	variable to load with option line
V2S120	variable to load with first parameter line field starting columns
V3S120	variable to load with first parameter line field sizes
V4S132	variable to load with first parameter line
V5S120	variable to load with second parameter line field starting columns
V6S120	variable to load with second parameter line field sizes
V7S132	variable to load with second parameter line
@OUM	call to display a blank function mask and obtain decoded data from terminal
0	mode number of report that has mask
1	form type of report
2	report identifier
1	format in which to display mask and decode input

OUM

example: extracting parameters

```
@LDV V10I3=1 .              INITIALIZE COLUMN CHARACTER POSITION
@DEF,C V11I3,V2 .           FIND NUMBER OF FIELDS TIMES 3
@1:LDV V12I3=V2(V10-3) .    GET START COLUMN NUMBER
@LDV V13I3=V3(V10-3) .      GET FIELD SIZE
@LDV V14S18=V4(V12-V13) .   GET FIELD FROM PARAMETER LINE
  .                         - save variables each time these
  .                           statements execute so that loop
  .                           can reuse them
@CHG V10 V10 + 3 .          INCREASE COLUMN CHARACTER POSITION BY 3
@IF V10 < V11 GTO 1 .       LOOP BACK IF MORE COLUMNS
```

now what?

You now have all the options, column-character positions, and parameter entries in variables. Use these variables in statements such as @SOR, @SRH, and @TOT, depending on the *type* of parameters the run user enters in the mask. An @SOR statement does not get far, for example, if your variables have search parameters.

@OUT

6.55. @OUT (OUTPUT)

Use an @OUT statement to place data on the screen and clear the output area after executing. An @OUT statement displays specified lines in a report; it need not erase the lines of data on the screen that are below the lines the @OUT statement has just put on the screen.

In an @OUT statement, you can specify where to display the data and where to position the cursor. Also, if your terminal has protect features, you can specify fields to protect. For an explanation of protected fields, see the UP manual for your display terminal.

Reserved words: INPUT$, INVAR$, and INVR1$

An @OUT statement stalls the run until you transmit, unless you select interim display (*interim?*) or forced transmit (*fxmit?*) by entering Y in these subfields.

With *interim display,* the run continues automatically and you *cannot* load variables with INPUT$, INVAR$, or INVR1$.

With *forced transmit,* the run continues automatically, also, but you *can* load variables INPUT$, INVAR$, or INVR1$.

You can use these two subfields to help you debug a MAPPER run.

NOTE: Don't use an @OUT statement in a MAPPER run started from a remote site, i.e., with an @RRN statement at another site.

run control statement format

@OUT,*m,t,r,l,lq,outl* [,*tabp,erase?,interim?,pdq,protect,fxmit?,space-a/b,blink?*],sn,lab

where:

m,t,r mode, type, and RID of report from which to display data

l line number in report where output is to start

lq *line-quantity:* number of lines to display

outl *output-line:* line on screen where display is to start

tabp *tab-position:* tab character position *n* on screen after which to position cursor, where *n* is number of tab positions forward (maximum 63) from home position, and *-n* is number of tab positions backward (maximum 100) from home position

OUT

erase? Y to erase unprotected portion of screen beyond *outl*

interim? Y to interim display (run continues)

pdq push-down-quantity: number of lines to push down on screen

protect *protected-format:* protected format option:

E Erase unprotected fields: erase only unprotected fields (fields beginning with a tab character and ending with a comma) in lines placed on the screen. The E option accelerates the run by reducing the number of words sent to the display terminal: only unprotected character positions are transmitted to the screen.

I Input edit: edit input characters in unprotected fields by character type. Place one of these edit codes in the first character position of the unprotected field in the output area. Use the numeric code to see the data typed in; use the counterpart letter code to make the data entered transparent:

space	Any data, no editing
0/A	Any data, no editing
1/B	Alphabetic data, left justified
2/C	Numeric data, left justified
3/D	No data
4/E	Any data, right justified
5/F	Alphabetic data, right justified
6/G	Numeric data, right justified

Numeric data includes the plus and minus signs (+, –) and the period (.).

P Protect fields: use protected fields when displaying data. If your terminal has a hardware intensity setting, enter one of these protect codes in column 1 of the output lines, or immediately after the comma that ends an unprotected field:

space	normal intensity
0	normal intensity
1	no intensity
2	low intensity
3	blinking characters, alternating low/normal intensity

If protected and unprotected fields appear on the same line, start the unprotected field with a tab character and end it with a comma. To use different intensities on the same line, place an unprotected field between the protected fields.

OUT

4 Four-to-one: fields defined on four lines for one output line:

line 1 M characters (FCC)
line 2 N characters (FCC)
line 3 emphasis characters (UTS 40 and UTS 40W only)
line 4 data to display

5 Five-to-one: fields defined on five lines for one output line:

line 1 M characters (FCC)
line 2 N characters (FCC)
line 3 color line, with one of the codes in the table that follows
line 4 emphasis characters (UTS 40 and UTS 40W only)
line 5 data to display

BACKGROUND

CHARACTER	Blk	Red	Grn	Ylw	Blu	Mgt	Cyn	Wht
Black	@	H	P	X	'	h	p	x
Red	A	I	Q	Y	a	i	q	y
Green	B	J	R	Z	b	j	r	z
Yellow	C	K	S	[	c	k	s	{
Blue	D	L	T	\	d	l	t	|
Magenta	E	M	U	]	e	m	u	}
Cyan	F	N	V	^	f	n	v	~
White	G	O	W	_	g	o	w	?

NOTE: shaded boxes indicate "invisible character" choices, with background color same as character color.

See the UP manual for your display terminal for its field control character (FCC) capability.

fxmit? *forced-transmit?:* Y to force transmit (run continues at next poll--you can load variables with INPUT$, INVAR$, or INVR1$)

space-a/b overrides normal method of printing characters on display terminal (whenever five or more spaces appear in a row, the actual data is not displayed; instead a cursor-positioning function is used; the result is an intermingling of data when the same line or lines are used to display various messages: if A, send the actual data and don't reposition the cursor; if B, don't alter the contents of unprotected fields but send only actual data)

blink? Y to change less-than sign (<) to a left blink character and change greater-than sign (>) to a right blink character before displaying data

example

In this example, the last subfield entered is the push-down quantity:

```
@OUT,40,B,4,2,6,1,2,Y,Y,2 .
```

where:

40,B,4	mode, type, and RID
2,6	line number in report (2) where output is to start; number of lines (6) to display
1	line on screen where display is to start
2	cursor position: second tab character position
Y	erase screen beyond output
Y	interim display (run continues automatically)
2	push down two lines, starting at line 1

OUT

example: using INPUT$ to enter data from the screen

When using INPUT$ to enter data from the screen, remember these things:

- The data entered in variables starts from a tab character: the data after the first tab character in the first variable, the data after the second tab character in the second variable, etc.
- The length of an input field varies: it depends on the variable's defined length.

To terminate an input field:

- Make the variable shorter than the input field (i.e., fewer characters than there are between tab characters, or between a tab character and comma on the same line).
- Make the input field shorter than the variable (i.e., fewer characters than the number of characters for which the variable was initialized).

This example, a portion of a run, uses the reserved word INPUT$ to enter data from the screen:

```
1.  @BRK,20,E  .
2.  ENTER APPROPRIATE DATA AND TRANSMIT.
3.  □2       , ENTER START DATE IN FORMAT YYMMDD
4.  □2       , ENTER END DATE IN FORMAT YYMMDD
5.  PLACE CURSOR HERE ->□   , AND TRANSMIT.
6.  @BRK,0,A OUT,20,E,-0,2,6,1,1,Y,,,I  .
7.  @CHG INPUT$ V116,V216  .
```

comments about this example

1. The first @BRK statement defines the next output area (i.e., the lines that follow) as mode 20, type E.
2. Place this line in output area.
3. Place this line in output area. *(2's are transparent when run is executed.*
4. Place this line in output area. *2 is protected format (FCC) I option code:*
5. Place this line in output area. *allow only numeric data, left justified.)*
6. The second @BRK statement places the preceding lines in the -0 result, mode 20, type E; it also defines the next output area as mode 0, type A. The OUT statement places the new -0 result on the screen. (Note the I option.)
7. @CHG INPUT$ changes the start date to V1 and the end date to V2.

example: using INPUT$ to enter data from a source external to the run

When using INPUT$ to enter data from a source external to the run, remember these things:

- You can enter up to 40 variables.
- Check the maximum size of the variable (see Table 3-1).
- Strings are not allowed.

This example, a portion of a run, uses the reserved word INPUT$ to enter data from a source external to the run and, later, to enter data from the screen:

```
1.  @CHG INPUT$ V1H2,V2I4,V3FS.2 .
      .
      .    (other run statements)
      .
2.  @BRK,0,A .
3.   ENTER PART NUMBER □
4.   ENTER QUANTITY □
5.  @BRK OUT,0,A,-0,2,2,1,1,Y .
6.  @CHG INPUT$ V1H12,V2H8 .
7.  @CHG V2 V2 + 0 .
```

comments about this example

1. The first statement in the run captures three variables from a source external to the run.
2. The first @BRK statement clears the output area and defines the next output area as mode 0, type A.
3. Place this line in output area.
4. Place this line in output area.
5. The second @BRK statement places the output area in a result (-0); the OUT statement displays two lines starting at line 2 of the result--it places the first line on line 1 of the display screen, places the cursor after the first tab character, and erases the remaining lines on the screen.
6. @CHG INPUT$ changes the part number to V1 and the quantity to V2.
7. This @CHG statement right-justifies the contents of V2. You could also use the statement @LDV,R V2=V2; or you could use the I option in the @OUT statement's *protect* subfield along with edit code 4, which right-justifies the data as it is entered on the screen.

OUT

example: using INVAR$ to enter data from the screen

When using INVAR$ to enter data from the screen, remember these things:

- You can enter up to 40 variables.
- You can use strings.
- Specify before the @OUT statement.
- The data entered in variables starts from a tab character: the data after the first tab character in the first variable, the data after the second tab character in the second variable, etc.
- The length of an input field varies: it depends on the variable's defined length.

To terminate an input field:

- Make the variable shorter than the input field (i.e., fewer characters than there are between tab characters, or between a tab character and comma on the same line).
- Make the input field shorter than the variable (i.e., fewer characters than the number of characters for which the variable was initialized).
- Use a comma after a tab character when using protected format.

This example, a portion of a run, uses the reserved word INVAR$ to enter data from the screen:

```
@CHG INVAR$ V1S17,V2I3,V3I6 .
ENTER DESCRIPTION □
ENTER QUANTITY □    ,
ENTER DATE IN FORMAT YYMMDD □      ,
@BRK OUT,0,A,-0,5,3,1,1,Y .
```

In this example, after the user enters the solicited information and transmits, the run continues at the statement that follows the @OUT statement. V1, V2, and V3 have the information the user entered.

using INVR1$ to enter data from the screen

When using INVR1$ to enter data from the screen, remember these things:

- You can enter up to 40 variables.
- You can use strings.
- Specify before the @OUT statement.
- Generally, input fields are terminated by the length of the variable.
- You should not use INVR1$ in protect mode.

using INVR1$ without protected format

The data entered in variables starts from a tab character: the data after the first tab character in the first variable, the data after the second tab character in the second variable, etc. The variable's length determines how many characters including the tab character are loaded into each variable.

using INVR1$ with protected format

Since INVR1$ stores the actual characters coming from the terminal in the first variable and continues until all characters are stored in variables, it does not load the variables the same way it loads them with INPUT$ or INVAR$, or as it stores them as described above without protected format.

With protected format, INVR1$ sends only significant characters (i.e., no leading or trailing blanks) to the site. Also, if an unprotected field (commonly called an input field) bounded by tab characters or a tab character followed by a comma is left blank, it returns only a blank followed by a tab character.

To make INVR1$ load each input field into each variable (when specifying several variables either horizontally or vertically), you must enter a character other than a space in each position of the field. However, since the MAPPER processor initializes type F and type I variables as zero if the input field is blank, this restriction does not apply with these variable types.

If you use INVR1$ with protected format and multiple input fields, you get this (assuming types A and H variables only):

1. All significant characters (not spaces) in order.
2. A space followed by a tab character, if an unprotected field (tab character to comma) has no significant characters.
3. The first variable loaded until it is full, then the next variable loaded until full, etc.

OUT

example: using INVR1$ to enter data from the screen

```
@CHG INVR1$ V1S17,V2I3,V3I6 .
ENTER DESCRIPTION □                    ,
ENTER QUANTITY □     ,
ENTER DATE IN FORMAT YYMMDD □          ,
@BRK OUT,0,A,-0,5,3,1,1,Y .
```

In this example, after the user enters the solicited information and transmits, the run continues at the statement that follows the @OUT statement. V1, V2, and V3 have the information the user entered. Note that protected format is not used.

four- and five-to-one output

If your display terminal lets you control emphasis for special editing and presentation techniques (see the UP manual for your display terminal), you can use either four- or five-to-one output. Four-to-one is for monochrome displays; five-to-one is for color displays.

example: four-to-one output

```
.DATE 25 JAN 83  12:13:14  RID    11     21 JAN 83  COORD
.Run Function Data: 4 to 1 Example, Run t4-1                              F000212
*        1         2         3         4         5         6         7         8
*2345678901234567890123456789012345678901234567890123456789012345678901234567890
*================================================================================
@chg input$ v1a1 if v1 = I dsp,16,g,11 .
@brk .
L
C
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

N                                                                       L
S                                                                       C
!                                                                       !

N                   L             N                                     L
S                   S             S                                     C
!                                                                       !
                      EXAMPLE 4-1
N                                                                       L
S                                                                       C
!                   $$$$$$$$$$$$$                                       !

N                   L             N                                     L
S                   @             S                                     C
!                   %$$$$$$$$$$$$!                                      !
 Any Data                                 Emphasis unprotected
N                   I             N                                     L
S                   A             S                                     C
!                   %$$$$$$$$$$$$!                                      !
 Alpha Data Only                          Emphasis protected
N                   h             N                                     L
S                   F             S                                     C
!                   %$$$$$$$$$$$$!                                      !
 Numeric Right Jst                        Emphasis protected
N                   m             N                                     L
S                   @             S                                     C
!                   %$$$$$$$$$$$$!                                      !
 Off Video Input
N                                                                       L
S                                                                       C
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$!

@brk out,16,g,-0,2,24,1,1,y,,,4 .
@gto end .
                      ..... END REPORT .....
```

Execute the T4-1 run to see the display.

Execute the T5-1 run to see emphasis character control on a color display terminal.

FOUR- AND FIVE-TO-ONE OUTPUT

If your display terminal lets you control emphasis for special editing and presentation techniques (see the documentation for your display terminal), you can use either four- or five-to-one output. Four-to-one is for monochrome displays; five-to-one is for color displays.

Field Control Characters (FCCs) first appeared on the UTS 400. The FCC allows various new screen intensities and immediate validation of numeric only/alpha only fields. It does this without taking up a position on the screen. To make these features available to MAPPER software users, editing codes were added to the existing screen definition procedures. However, these editing codes still used up valuable character positions on the screen.

Beginning with MAPPER software level 29R1, another variation on screen definitions was added. It lets you take full advantage of the FCC without using up a screen position. It also allows the use of the emphasis characters available on the UTS 40. Before we see how to do this, let's take a brief look at FCCs and emphasis characters.

OUT

A BRIEF LOOK AT FCCS

The FCC, as sent from MAPPER software to the terminal, consists of a special sequence of five bytes. The first byte is the US (unit-separator) control character. The second and third bytes designate the Y position (row) and X position (column) on the screen. MAPPER software takes care of these first three bytes for you.

You must specify the fourth and fifth bytes (arbitrarily called the M and N bytes). These carry codes that specify the characteristics of the FCC. The characteristics available for an FCC are grouped as follows:

M Characteristics		N Characteristics	
Intensities	-Normal	Input Validation	-Any input
	-Low/reverse		-Alphabetic only
	-Blinking		-Numeric only
	-Off (UTS 400 only)		-Protected (no input)
Tab Stop Setting	-Tab stop	Justification	-None
	-No tab stop		-Right justified

You control the intensities and tab stops using the M byte. You control the input validation and justification using the N byte. To create a complete FCC, you must specify both the M and N bytes. Select the applicable codes for UTS 20 and UTS 400 display terminals from these tables:

M Characteristics

Code	Description
4	Tab stop, normal intensity
5	Tab stop, off intensity
6	Tab stop, low/reverse intensity
7	Tab stop, blinking intensity
<	No tab stop, normal intensity
=	No tab stop, off intensity
>	No tab stop, low/reverse intensity
?	No tab stop, blinking intensity

N Characteristics

Code	Description
0	Any input, no justification
1	Alpha only, no justification
2	Numeric only, no justification
3	Protected (no input)
4	Any input, right justified
5	Alpha only, right justified
6	Numeric only, right justified

For instance, suppose you want to set up a normal intensity, alpha only field with a tab stop and no justification. First you would select a 4 from the M table and then a 1 from the N table. Suppose you want the field labels to be low intensity and protected. You would select a > from the M table and a 3 from the N table. (See the documentation for your display terminal for specific M and N characteristics.)

Note that the characteristics of an FCC are in effect from the position of the FCC to the last position on the screen or the next FCC, whichever comes first. This means that to define a field, there must be one FCC at the beginning of the field and another at the end.

EMPHASIS CHARACTERS

Emphasis characters consist of the column separator, the underscore, and the strike–through.

Emphasis characters, as sent from MAPPER software to the terminal, consist of a special sequence of two bytes. The first byte is the ESC (escape) control character. MAPPER software takes care of it for you. You specify the second byte, which contains codes that are used to specify the combination of emphasis characters to display. Select the codes from this table:

Emphasis Characters

Code	Column Separator	Underscore	Strike Through
SP	–	–	–
!	YES	–	–
$	–	YES	–
%	YES	YES	–
(	–	–	YES
)	YES	–	YES
,	–	YES	YES
–	YES	YES	YES

For instance, if you wanted your input fields to be displayed with underscores, you would select a $ from the table above. If you wanted to create a vertical line, you would select a series of ! characters.

Now that you have had this brief look at FCCs and emphasis characters, you're ready to see how MAPPER software defines screens that use them.

FOUR-TO-ONE SCREENS

MAPPER software's four-to-one screens make full use of FCCs and emphasis characters. They allow the you to specify (1) an M byte, (2) an N byte, (3) an emphasis character, and (4) a displayable character all for the same screen position. You must place each of the codes/characters in the same column of four successive lines of source code. Place the M byte on the first line, the N byte on the second line, the emphasis character on the third line, and the displayable character on the fourth line.

For example, suppose you wanted to define a STATE field as normal intensity, alpha only, with a tab stop, and no justification. First you would select a 4 from the M table, a 1 from the N table, and then code four lines as follows:

```
        4      (M byte)
        1      (N byte)
               (Emphasis char.)
STATE:         (Displayable char.)
```

If you also wanted the field labels to be low intensity and protected, you would select a > from the M table, a 3 from the N table, and add the codes as follows:

```
>       4  >   (M byte)
3       1  3   (N byte)
               (Emphasis char.)
S TATE: ___    (Displayable char.)
```

OUT

When MAPPER software receives a four–to–one screen, it scans the source lines four at a time, vertically combines the four codes/characters of each column, and generates one line of output. The output line will contain the FCCs, emphasis characters, and displayable characters ready for transmission to the terminal.

Here's an FCC sequence:

```
D L L        (M byte)
A B C        (N byte)
```

Changing a field causes initialization of a new variable even though no tab character was specified.

NOTE: When redefining the editing code, even though no tab character was specified, the contents are placed into the next variable.

To tell MAPPER software that you intend to display a four–to–one screen, enter a 4 as the **protected–format** option on the OUT statement as follows:

```
@BRK OUT,-0,2,16,1,1,Y,,,4 .
```

Now you are ready to code a simple data entry screen. Suppose you need to enter the NAME, STREET, CITY, STATE, ZIP, and PHONE of a new customer. The STATE field should allow only alphabetic input. The input fields should be normal intensity, have tab stops, and be filled with underscores. The field labels should be low intensity and protected.

You could code this data entry screen as follows:

```
1.  @BRK.
2.  >          4                                          >
3.  3    N     0                                          3
4.
5.  NAME...:_____________________________________________
6.             4                                          >
7.             0                                          3
8.
9.  STREET.:_____________________________________________
10.            4                                          >
11.            0                                          3
12.
13. CITY...:_____________________________________________
14.            4 >        4    >          4  >4   >4   >
15.            1 3        2    3          2  32   32   3
16.
17. STATE..:__    ZIP:_____   PHONE:(___)___-____
18. @BRK OUT,-0,2,4,1,1,Y,,,4   .
```

Upon execution of the OUT statement, source lines 2, 3, 4, and 5 combine to form output line A. Lines 6, 7, 8, and 9 combine to form line B. Lines 10, 11, 12, and 13 combine to form line C. Lines 14, 15, 16, and 17 combine to form line D. The system then displays the following screen:

```
A.   NAME...:_____________________________________
B.   STREET.:_____________________________________
C.   CITY...:_____________________________________
D.   STATE..:__  ZIP:_____  PHONE:(___)___-____
```

A similar update screen coded for a four-to-one format might appear as follows:

```
1.  @BRK.
2.  N       d                                       N
3.  S       @                                       S          A
4.          $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
5.  NAME...:V10
6.          d                                       N
7.          @                                       S          B
8.          $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
9.  STREET.:V11
10.         d                                       N
11.         @                                       S          C
12.         $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
13. CITY...:V12
14.         d N     d    N        d   Nd   Nd   N
15.         A S     B    S        B   SB   SB   S             D
16.         $$      $$$$$         $$$$ $$$$ $$$$
17. STATE..:V13  ZIP:V14  PHONE:(V15)V16-V17
18. @BRK OUT,-0,2,4,1,1,Y,,,4 .
```

When the OUT statement is executed, the system displays this screen:

```
A.   NAME...:Taiwan Tool & Die
B.   STREET.:1234 Main Street
C.   CITY...:Los Angeles
D.   STATE..:CA  ZIP:90007  PHONE (213)555-6789
```

The underscored areas are unprotected and of normal intensity. Because the underscores are protected emphasis characters, they are continuous and are not overwritten by input from the keyboard. The field labels are protected and of low intensity. The STATE field allows only alphabetic input. The ZIP and PHONE fields allow only numeric input.

With the advent of four-to-one screens in MAPPER software, you can take full advantage of FCCs and emphasis characters. It is easy to protect the slashes between the day, month, and year of a date field. You can also set up a numeric-only field without losing the first character of your data.

3.5. UTS-Style Control Page

Terminal characteristics can be changed by using the UTS-style control page (Figure 3-4). The terminal characteristics or parameters are entered in the first subfield under the (PARAM) field. The parameter options are entered in the second subfield.

```
(**PRINT*)STA- (**XFER**)PRNT( )XFER( )XMIT( )MM( PARAM)
( / / )ADR-    ( / / )SEARCH( )                    ( / )
```

Figure 3-4. UTS-Style Control Page

Table 3-4 lists the parameters and the parameter options.

Table 3-4. UTS-Style Control Page Parameters (Part 1 of 3)

Parameter	Parameter Code	Option Code	Function
Remote Identifier**	RI	21-7F	Specifies the remote site transmission code (RID).
Station Identifier**	SI	20-7F	Specifies the terminal transmission code (SID).
System Response Mode	SR		System Response Mode for the SVT-1123 or SVT-1124.
		ON*	Enables System Response Mode.
		OFF	Disables System Response Mode.
SITA Timing Requirements	ST		Specifies maximum time that RTS line is allowed to be active. Normal is 45 sec. SITA requirement is .5 sec. or less.
		ON	Enables SITA timing requirements.
		OFF*	Disables SITA timing requirements.

* Default Option
** The terminal must be reset to enable these parameters.

Table 3-4. UTS-Style Control Page Parameters (Part 2 of 3)

Parameter	Parameter Code	Option Code	Function
CRT Saver** (video off time)	VO		Causes the screen to go blank after a specified period (number of minutes) of keyboard activity.
		00	CRT saver disabled.
		05*	Screen goes blank after 5 minutes.
		30	Screen goes blank after 30 minutes.
		60	Screen goes blank after 60 minutes.
Space bar selection	SP		Defines the space bar erase function.
		NS*	Nondestructive space bar.
		DS	Destructive space bar.
Key click	KK		Turns the key click on or off.
		ON*	Keys click audibly.
		OFF	Keys don't click audibly.
Communications mode**	CM		Identifies the type of communications link to the host.
		MX*	Multiplexer.
		MD	Modem or DCM.

* Default Option
** Terminal must be reset to enable these parameters

Table 3-4. UTS-Style Control Page Parameters (Part 3 of 3)

Parameter	Parameter Code	Option Code	Function
Uppercase	UC		Allows the selection of only uppercase alphabetic characters from the keyboard.
		YS	All alphabetic characters are entered in uppercase independent of the SHIFT key.
		NO*	Allows normal entry of uppercase and lowercase alphabetic characters.
Transmit expanded FCCs	TF		Specifies the transmission of expanded FCCs.
		EF	Enables the transmission of expanded FCCs.
		DF*	Disables the transmission of expanded FCCs.
Automatic hangup**	AA		Disconnects the communications line from the host processor after a specified period of inactivity.
		NO*	Does not hang up.
			Hangs up after:
		Y1	30-second timeout.
		Y2	1-minute timeout.
		Y3	3-minute timeout.
		Y4	5-minute timeout.

* Default option.
** The terminal must be reset to enable these parameters.

Table 7-1. ASCII Characters Used as M in the Expanded FCC Sequence

Bit 3	Bit 2	Bit 1	Bit 0	Bit 5	
Tab Status	**Changed Data**	**Intensity***	**Video**	**Emphasis Not Protected**	**Emphasis Protected**
Tab Stop	Field Changed	Normal Intensity	Video on	@ 40	60
			Video off	A 41	a 61
		Low Intensity	Video on	B 42	b 62
			Video off	C 43	c 63
	Field Not Changed	Normal Intensity	Video on	D 44	d 64
			Video off	E 45	e 65
		Low Intensity	Video on	F 46	f 66
			Video off	G 47	g 67
No Tab Stop	Field Changed	Normal Intensity	Video on	H 48	h 68
			Video off	I 49	i 69
		Low Intensity	Video on	J 4A	j 6A
			Video off	K 4B	k 6B
	Field Not Changed	Normal Intensity	Video on	L 4C	l 6C
			Video off	M 4D	m 6D
		Low Intensity	Video on	N 4E	n 6E
			Video off	O 4F	o 6F

*Intensity (bit 1) is ignored if video is off.

Table 7-2. ASCII Characters Used as N in the Expanded FCC Sequence

Bit 3	Bit 2	Bit 1, 0	Bit 4	
Blinking	**Justification**	**Type of Data Entry**	**Normal Intensity**	**Reverse Video**
No blink	Normal Field	Any	@ 40	P 50
		Alpha	A 41	Q 51
		Numeric	B 42	R 52
		Protected	C 43	S 53
	Right-Justified	Any	D 44	T 54
		Alpha	E 45	U 55
		Numeric	F 46	V 56
		Protected	G 47	W 57
Blinking	Normal Field	Any	H 48	X 58
		Alpha	I 49	Y 59
		Numeric	J 4A	Z 5A
		Protected	K 4B	[5B
	Right-Justified	Any	L 4C	\ 5C
		Alpha	M 4D	] 5D
		Numeric	N 4E	^ 5E
		Protected	O 4F	_ 5F

Table 7-3. ASCII Characters Used as M in the UTS 400-Compatible FCC Sequence

Tab Stop	Field Changed	Normal intensity	0 30
		Video off	1 31
		Low intensity	2 32
		Blinking	3 33
	Field Not Changed	Normal intensity	4 34
		Video off	5 35
		Low intensity	6 36
		Blinking	7 37
No Tab Stop	Field Changed	Normal intensity	8 38
		Video off	9 39
		Low intensity	: 3A
		Blinking	; 3B
	Field Not Changed	Normal intensity	< 3C
		Video off	= 3D
		Low intensity	> 3E
		Blinking	? 3F

Table 7-4. ASCII Characters Used as N in the UTS 400-Compatible FCC Sequence

Normal Field	Any	0 30
	Alpha	1 31
	Numeric	2 32
	Protected	3 33
Right-Justified	Any	4 34
	Alpha	5 35
	Numeric	6 36
	Protected	7 37

Emphasis Underline

The terminal provides an emphasis underline that can be displayed in the same position as a data character. The emphasis underline can be protected without protecting data characters appearing in the same position (see Table 7-1). The emphasis underline command sequences are:

Function	Code
Create emphasis	ESC $
Delete emphasis	ESC Z $

Editing

The available editing capabilities are listed below. Refer to UP-10683 for an explanation of these functions.

Function	Code
Erase to end of display	ESC a
Erase to end of field	ESC K
Erase to end of line	ESC b
Erase display	ESC M
Delete in line	ESC c
Delete in display	ESC C
Delete line	ESC k
Insert in line	ESC d
Insert in display	ESC D
Insert line	ESC j
Line duplicate	ESC y
Clear FCC	ESC w

STATION AND PERIPHERAL DEVICE CONTROL CODES

The station and peripheral device control codes used by the UTS 20 are given

Function	Code/Sequence	Par. Ref.	Function	Code/Sequence	Par. Ref.
Cursor address sequence	ESC VT Y X SI	4.2.2.1	Transmit variable	DC1	4.5.6
SOE position	ESC VT Y X NUL SI	4.3.1	Transmit all	ESC DC1	4.5.5
Start of entry (SOE)	RS	4.3.1	Transmit changed	ESC t	4.5.7
Cursor return (new line)	CR	4.2.2.3	Clear changed	ESC u	4.5.8
Cursor to home	ESC e	4.2.2.2			
Send cursor address	ESC T	4.5.1	Call error log	ESC P	4.5.2
			Clear error log	ESC R	4.5.3
Erase unprotected data	ESC a	4.4.3.1			
Erase to end of line	ESC b	4.4.3.3	Initiate confidence test	ESC Q	4.5.4
Erase to end of field	ESC K	4.4.3.2			
Erase display	ESC M	4.4.3.4			
Erase character (space)	SP	4.4.3.5	Blinking start marker	FS	4.4.8.2
			Blinking end marker	GS	4.4.8.3
Delete in line	ESC c	4.4.4.1			
Delete in display	ESC C	4.4.4.2	Lock keyboard	DC4 or ESC DC4	4.5.11
Delete line	ESC k	4.4.4.3			
			Control page access	ESC o	4.5.12
Insert in line	ESC d	4.4.5.2			
Insert in display	ESC D	4.4.5.3	Shift in	SI	4.4.2
Insert line	ESC j	4.4.5.1	Shift out	SO	4.4.2
Line duplication	ESC y	4.4.6			
			Line feed	LF	4.6.2.2
Scan left	ESC g	4.2.2.4	Form feed	FF	4.6.2.1
Scan right	ESC h	4.2.2.4			
Scan down	ESC i	4.2.2.4	Long FCC sequence	US Y X M N	4.4.1.2.1
Scan up	ESC f	4.2.2.4	Immediate FCC sequence	EM M N	4.4.1.2.2
Forward tab	HT	4.4.7.3			
Tab stop set	ESC HT	4.4.7.1	Clear FCC	ESC w	4.4.3.6
Backward tab	ESC z	4.4.7.4			
			Automatic disconnection	DLE EOT	4.5.9
More to come	ETB	3.4.2.4.1, 3.5.1			
			Vertical tabulator	VT	4.6.2.3
Transfer changed	ESC E	4.6.1			
Transfer variable	ESC F	4.6.1	Message waiting	BEL	4.5.10
Transfer all	ESC G	4.6.1			
Print form	ESC H	4.6.1			
Print	DC2	4.6.1			
Print transparent	ESC DC2	4.6.1			

ESCAPE CHARACTERS

		UTS 20	UTS 30	SVT1120
a	Erase to end of display	YES	YES	YES
b	Erase to end of line	YES	YES	YES
c	Delete in line	YES	YES	YES
d	Insert in line	YES	YES	YES
e	Cursor home	YES	YES	YES
f	Cursor up (vertical tab)	YES	YES	YES
g	Cursor left (backspace)	YES	YES	YES
h	Cursor right (forward space)	YES	YES	YES
i	Cursor down (line feed)	YES	YES	YES
j	Insert line	YES	YES	YES
k	Delete line	NO	NO	YES
n	Turn on control page	(TOGGLE CONTROL PAGE)		YES
o	Turn off control page	NO	YES	YES
q	Print screen	YES	YES	YES
t	Transmit	YES	YES	YES
w	Clear FCC	YES	YES	YES
x	Cursor home	NO	NO	YES
y	Line duplicate	YES	YES	YES
z	Back tab	YES	YES	YES
C	Delete in display	YES	YES	YES
D	Insert in display	YES	YES	YES
E,F,G	XFER	NO	NO	YES
K	Erase to end of field	YES	YES	YES
M	Erase display	YES	YES	YES
P	Transmit (improper) (Keyboard in wait on SVT)	YES	YES	NO
Q	Reset terminal	YES	NO	YES
T	Send cursor address (causes 'No significant data for the entry')	YES	YES	YES
U	Transmit (improper)	NO	YES	NO
V	Mess up screen	NO	NO	YES
Y,Z	don't send data	NO	YES	YES
[	hard escape (9B)	YES	YES	YES
R	Keyboard in wait	YES	YES	NO
(Tab)	Insert tab into line	YES	YES	YES

M Characters:

The M Character FCC is specified in Line 1 of the 4-to-1 or 5-to-1 output. Select the FCC character in the M Character Table to suit your needs.

Note: The "Field Changed?" column is only available for IBM 3270 terminals.

M Character Table

FCC	Protected Emphasis	Tab Stop?	Field Changed?	Low Intensity?	Video On?
@	NO	YES	YES	NO	YES
A	NO	YES	YES	NO	NO
B	NO	YES	YES	YES	YES
C	NO	YES	YES	YES	NO
D	NO	YES	NO	NO	YES
E	NO	YES	NO	NO	NO
F	NO	YES	NO	YES	YES
G	NO	YES	NO	YES	NO
H	NO	NO	YES	NO	YES
I	NO	NO	YES	NO	NO
J	NO	NO	YES	YES	YES
K	NO	NO	YES	YES	NO
L	NO	NO	NO	NO	YES
M	NO	NO	NO	NO	NO
N	NO	NO	NO	YES	YES
O	NO	NO	NO	YES	NO
'	YES	YES	YES	NO	YES
a	YES	YES	YES	NO	NO
b	YES	YES	YES	YES	YES
c	YES	YES	YES	YES	NO
d	YES	YES	NO	NO	YES
e	YES	YES	NO	NO	NO
f	YES	YES	NO	YES	YES
g	YES	YES	NO	YES	NO
h	YES	NO	YES	NO	YES
i	YES	NO	YES	NO	NO
J	YES	NO	YES	YES	YES
k	YES	NO	YES	YES	NO
l	YES	NO	NO	NO	YES
m	YES	NO	NO	NO	NO
n	YES	NO	NO	YES	YES
o	YES	NO	NO	YES	NO

N Characters:

The N Character FCC is specified in Line 2 of the 4-to-1 or 5-to-1 output. Select the FCC character in the M Character Table to suit your needs.

N Character Table

FCC	Reverse Video?	Blink?	Right Justify?	Edit Characteristics
@	NO	NO	NO	NO EDITING
A	NO	NO	NO	ALPHA ONLY
B	NO	NO	NO	NUMERIC ONLY
C	NO	NO	N/A	PROTECTED
D	NO	NO	YES	NO EDITING
E	NO	NO	YES	ALPHA ONLY
F	NO	NO	YES	NUMERIC ONLY
G	NO	NO	N/A	PROTECTED
H	NO	YES	NO	NO EDITING
I	NO	YES	NO	ALPHA ONLY
J	NO	YES	NO	NUMERIC ONLY
K	NO	YES	N/A	PROTECTED
L	NO	YES	YES	NO EDITING
M	NO	YES	YES	ALPHA ONLY
N	NO	YES	YES	NUMERIC ONLY
O	NO	YES	YES	PROTECTED
P	YES	NO	NO	NO EDITING
Q	YES	NO	NO	ALPHA ONLY
R	YES	NO	NO	NUMERIC ONLY
S	YES	NO	N/A	PROTECTED
T	YES	NO	YES	NO EDITING
U	YES	NO	YES	ALPHA ONLY
V	YES	NO	YES	NUMERIC ONLY
W	YES	NO	N/A	PROTECTED
X	YES	YES	NO	NO EDITING
Y	YES	YES	NO	ALPHA ONLY
Z	YES	YES	NO	NUMERIC ONLY
[	YES	YES	N/A	PROTECTED
\	YES	YES	YES	NO EDITING
]	YES	YES	YES	ALPHA ONLY
**	YES	YES	YES	NUMERIC ONLY
_	YES	YES	N/A	PROTECTED

** means the caret character

Emphasis Characters:

The emphasis characters sent from MAPPER software to the terminal consist of a special sequence of two bytes. The first byte is the ESC (Escape) control character and is handled by the MAPPER software.

You specify the second byte, which contains codes that are used to determine the combination of emphasis characters to display. The emphasis character code is specified in Line 3 of the 4-to-1 or Line 4 of the 5-to-1 output.

Emphasis Character Table

Code	Strike-through	Underscore	Column Separator
space	NO	NO	NO
!	NO	NO	YES
$	NO	YES	NO
%	NO	YES	YES
(	YES	NO	NO
)	YES	NO	YES
*	YES	YES	NO
-	YES	YES	YES

..... END REPORT .....

6.68. @RRN (REMOTE RUN)

The @RRN statement starts a run at another MAPPER site.

NOTES: *1. Before using an @RRN statement, make sure that a remote run link exists between MAPPER sites.*

2. You must pass all user sign-on and run use security checks at the remote MAPPER site.

3. The remote run must not have any @DSP, @OUT, @REL, or @XIT statements in it. The remote MAPPER processor cannot execute these statements because it does not treat the link the same as a station that might start the run.

Reserved word: STAT1	
Word	**Content**
if remote run is successful. . .	
STAT1	0
if remote run is unsuccessful. . .	
STAT1	> 0

run control statement format

@RRN[*,loc-m,loc-t,loc-r,loc-f,rtnerr?*] *run-name,* { *v* | *data* } *rem-site,rem-user, rem-dept,rem-upsw* [*,msg?,ltpm,rem-m,rem-mpsw,rem-t,rem-f*]

where:

loc-m,loc-t,loc-r,loc-f	local mode, type, RID, and format
rtn-err?	*return-error?:* Y to do a normal return whether or not remote run errs
run-name	name of run to execute at remote site (registered and resident at remote site)
v | *data*	variables or constant data transferred to remote site for use as variables by remote run with INPUT$ (cannot exceed 12 characters – excess characters are truncated).

RRN

rem-site	remote MAPPER site number
rem-user	user-id registered at remote site
rem-dept	user sign-on department number registered at remote site
rem-upsw	user sign-on password registered at remote site
msg?	Y to display link transfer progress messages (which indicate buffers processed) as data passes to remote site
ltpm	*line-transfer-progress-message:* line number on local display screen where transfer progress messages (if specified) are to start
rem-m,rem-mpsw, rem-t,rem-f	remote mode, remote mode password, remote *alphabetic* type, and remote format

If you define the local report, you must also specify the remote report identification (subfields *rem-m, rem-mpwd, and rem-t*). The local report passes to the remote site and is the result (-0) in the remote mode and form type specified.

example

```
@RRN THISRUN,DATA1 2,JDOE,1,A .
```

This statement is requesting that the run THISRUN be started at remote site number 2, and that data DATA1 be transferred to be picked up with INPUT$, with JDOE,1,A the sign-on at the remote site.

@RSI

6.68A. @RSI (REMOTE SYMBIONT INTERFACE)

The RSI statement initiates an interactive demand program through a MAPPER run. When a MAPPER run calls the RSI function, the MAPPER system terminates the run. Then the MAPPER system signs the user on to the 1100 Operating System in demand mode, and gives the user manual control of the display terminal.

If the MAPPER user is not a valid RSI user (as defined in the user registration RID), no @ or @@ commands (except for @@TERM and @@X) are allowed.

run control statement format

@RSI [,*site-id* ,*timeout*] *user-id* ,*psw* [,*acct, qual, file, cyc, elt, ver, lq*] .

where:

RSI — the Remote Symbiont Interface function call.

site-id — 6-character site identifier for this display terminal.

timeout — the number of seconds that the RSI function will wait for demand output to the terminal. If left blank, no timeouts will occur. Pressing the MSG WAIT key before the user receives manual control causes the 1100 OS to abort the RSI function.

user-id — the demand mode user-id.

psw — the demand mode password that validates the user-id.

acct — account number. If specified, the MAPPER system submits a RUN statement, with this format:

@RUN *run-id, acct-no* /,*project-id*

The *run-id* is the active *site-id*. The *project-id* is the *qual* from the RSI function call. If you didn't specify one, this field is left blank.

qual — file name qualifier.

file — If specified, the MAPPER system submits an ADD statement after demand mode sign-on is complete. Any file-related subfield left blank on the RSI function call is left blank by the MAPPER system on the ADD statement.

cyc — file cycle number.

elt — element name.

ver — element version name.

RSI

lq specifies how many lines of text are to be displayed prior to giving manual control to the user. The minimum is one line (the default) and the maximum is one less than the vertical screen size of the display terminal. An exception exists if a demand program executes a control command that is interpreted by the MAPPER CCR before it solicits input from the terminal for the first time. In this case, all output prior to the control command is discarded.

NOTE: Due to the text-oriented nature of demand mode, the MAPPER System doesn't know if the parameters specified on the RSI function call result in an error (user-id/password invalid, ADD statement doesn't exist, etc.).

@SLU

6.73. @SLU (SEARCH LIST UPDATE)

The @SLU statement searches a report like an @SRL statement (6.76) searches a report, i.e., using search parameters listed in another report, and creates a result (▽▽▽UPD RESULT▽▽▽).

You can:

- delete the lines in the result from the report (DEL—see 6.20);
- delete the lines in the result from the report and redisplay the result (EXT—see 6.30); or
- make changes to the result and blend these lines into the report (UPD—see 6.84).

run control statement format

@SLU,*m,t*[,*r,lab*] *o c-c lt,'report-id'* [*vfinds,vls,vrid*]

where:

m,t,r	mode, type, and RID of report to search
lab	label to go to if no data is found
o	options field (see 6.75)
c-c	column-character positions to search
lt	line type to search
'report-id'	identifies report having search parameters
vfinds,vls,vrid	three variables produced by this statement:

vfinds	*variable-finds:* number of lines where finds were made
vls	*variable-lines-searched:* number of lines searched
vrid	*variable-rid:* RID where finds were made

example

```
@slu,2,b,2 '' 2-2 □,'m Otest 5d' del .
```

This statement searches column 2 for two characters in report 2b, mode 2, for the parameters listed in report 5d, mode 0, and deletes the found lines.

@SOR

6.74. @SOR (SORT)

The @SOR statement sorts report data. The MAPPER processor automatically places an update lock on a report whenever the @SOR statement is issued, to maintain the integrity of the report during statement execution.

run control statement format

@SOR,*m,t,r o c-c lt,p*

where:

m,t,r mode, type, and RID of report to sort

o options:

A sort all line types

C(*x*) case sensitivity:

C(F) full character set
C(L) limited character set
C(S) strict character set of report

c-c column-character positions to sort

lt line type to sort

p sort parameters

example: sorting a single level

To sort a single level, enter a 1 after the line type:

```
@SOR,40,B,V1 '' 2-8 *,1 .
```

where:

40,B,V1 mode, type, and RID (value in V1)

' ' indicates no options selected

2-8 start sort in column 2 for eight characters

* sort asterisk type lines

1 sort one level in ascending order

@SRL

6.76. @SRL (SEARCH LIST)

The @SRL statement searches a report like an @SRH statement (6.75), and creates a result.

First, however, you must place search parameters in a report, then identify the report to the @SRL statement in the *'report-id'* subfield. An @SRL statement searches each parameter in each field.

NOTE: Use @SRL statements sparingly; they are usually not as efficient as @SRH statements.

run control statement format

@SRL,*m,t*[,*r,lab*] *o c-c lt,'report-id'* [*vfinds,vls,vrid*]

where:

m,t,r	mode, type, and RID of report to search
lab	label to go to if no data is found
o	options field (see 6.75)
c-c	column-character positions to search
lt	line type to search
'report-id'	identifies report having search parameters
vfinds, vls,vrid	three variables produced by this statement: *vfinds* *variable-finds:* number of lines where finds were made *vls* *variable-lines-searched:* number of lines searched *vrid* *variable-RID-number:* number of reports where finds were made

example

```
@SRL,40,B,10,5 '' 2-10 *,'M 2TEST 4C' V116,V217,V313 .
```

where:

40,B,10	mode, type, and RID
5	label to go to if no data is found
' '	no options specified
2-10	search column 2 for 10 characters
*	search asterisk type lines
'M 2TEST 4C'	identify search list report: mode 2, password TEST, RID 4, form type C
V1I6,V2I7,V3I3	variables: V1 has number of finds; V2 has number of lines searched; V3 has RIDs where finds were made

@SRU

6.77. @SRU (SEARCH UPDATE)

The @SRU statement searches vertically through a report, and creates a result (▽▽▽UPD RESULT▽▽▽).

You can:

- delete the lines in the result from the report (DEL--see 6.20);
- delete the lines in the result from the report and redisplay the result (EXT--see 6.30) or
- make changes to the result and blend these lines into the report (UPD--see 6.84).

run control statement format

@SRU,*m,t*[,*r,l,lq,lab*] *o c-c lt,p* [*vfinds,vls,vrid*]

where:

m,t,r	mode, type, and RID of report to search
l	line number at which to start search
lq	*line-quantity:* number of lines to search.
lab	label to go to if no data is found
o	options field (same as SEARCH, except S option invalid--see 6.75)
c-c	column-character positions to search
lt	line type to search
p	search parameters
vfinds,vls,vrid	three variables produced by the statement:

vfinds	*variable-finds:* number of finds
vls	*variable-lines-searched:* number of lines searched
vrid	*variable-rid:* RID where finds were made

@WAT

6.85. @WAT (WAIT)

Use a @WAT statement to temporarily suspend the execution of a run.

Also, use @WAT statements:

- interspersed with extensive logic loops to reduce the impact on the MAPPER processor; or
- to hold an interim display (@OUT and @DSP) on the screen long enough to read.

The maximum wait time is 60,000 milliseconds, i.e., 60 seconds.

run control statement format

@WAT *milliseconds*

example

```
@wat 100 .  wait 100 milliseconds (1/10 second) before resuming
```

@WDC

6.86. @WDC (WORD CHANGE)

The @WDC statement changes words in a receiving report from a list of target words in an issuing report, and creates a result.

In the issuing report:

- There must be a header-divider (*=) line.
- The first word is the target word; the second word is the replacement word, the word to which to change the target word in the receiving report.
- Words start in column 2 on tab type lines and always end with a comma.
- Words may have alphanumeric characters A through Z, and = through 9, with no embedded spaces.

Here are sample entries in an issuing report:

```
□cat,dog,
□boy,girl,
```

run control statement format

@WDC,*iss-m,iss-t,iss-r,rec-m,rec-t,rec-r* [,*l,col,lab*]

where:

iss-m,iss-t,iss-r	mode, type, and RID of issuing report
rec-m,rec-t,rec-r	mode, type, and RID of receiving report
l	line number in receiving report at which to start changing words
col	column number in first line of receiving report at which to start changing words
lab	label to go to in case of error

example

```
@wdc,0,b,3,0,c,3 .
```

@WDL

6.87. @WDL (WORD LOCATE)

The @WDL statement locates words in a receiving report from a list of target words in an issuing report.

In the issuing report:

- There must be a header-divider (*=) line.
- Target words start in column 2 on tab type lines.
- Words may have alphanumeric characters A through Z, and = through 9, with no embedded spaces.

Here are sample entries in an issuing report:

```
□cat
□dog
□boy
□girl
```

run control statement format

@WDL,*iss-m,iss-t,iss-r,rec-m,rec-t,rec-r* [,*l,col,lab*] *vln,vcol,vcolnext*

where:

iss-m,iss-t,iss-r	mode, type, and RID of issuing report
rec-m,rec-t,rec-r	mode, type, and RID of receiving report
l	line number in receiving report at which to start locating words.
col	column number in first line of receiving report at which to start locating words
lab	label to go to in case of error
vln,vcol,vcolnext	three variables produced by the statement:

- *vln* — *variable-line-number:* line number where word was located
- *vcol* — *variable-column-number:* column number where word starts
- *vcolnext* — *variable-column-next:* column number immediately after word

example

```
@wdl,0,b,3,0,c,3 v1i6,v2i6,v3i6 .
```

@WPR

6.88. @WPR (WORD PROCESS)

The @WPR statement executes word processing commands against reports or results, and creates a result; or allows manual interactive word processing.

Reserved words: STAT1 and STAT2	
Word	Content
if run encounters an error, it goes to label in *lab* subfield. . .	
STAT1	Message number (Use an @LSM statement [see 6.49] to obtain the message.)
STAT2	Line number in report being processed where error occurred

run control statement format

@WPR,*m,t,r* [,*l*,*lab*] { *wpcommand* | ' ' }

where:

m,t,r mode, type, and RID of report to process

l is the line at which to start the display (interactive)

lab label to go to in case of error

wpcommand word processing command:

ADJ	Adjust
ADJDOC	Adjust/document
ADJPRT	Adjust/print
DOC	Document

WPR

FRONT	Front pages
INDEX	Index
LOWER	Lowercase
PG,*n*	Page number
PREP	Prepare
PRT	Print
SEC	Section
TOC	Table of Contents
UPPER	Uppercase

leave this field blank ('') to enter interactive word processing

examples

```
@wpr,0,b,3,,4 adjprt .
```

In this statement, we're requesting that mode 0, type b, RID 3, be adjusted and made ready to print, and that a result (-0) be produced. If an error occurs during execution, go to label 4.

```
@wpr,4,c,6 '' .
```

In this statement, we're requesting that mode 4, type c, RID 6, be displayed on the screen for interactive word processing:

To exit interactive word processing, press F3 .

@WRL

6.89. @WRL (WRITE LINE)

The @WRL statement updates up to 23 lines of a report or result.

The @WRL statement uses either literal values or input from variables and writes the line type designator along with the specified data.

run control statement format

@WRL,*m,t,r,l* [,*no-id?,writepsw*] *cc lt,* { *v* | *data* }

where:

m,t,r	mode, type, and RID number of report to update
l	line number to update
no-id?	*no-time/user-id?:* Y to ***not*** update the time and user-id
writepsw	if update password is entered here, enter it in report also*
	* *You **must** enter data in remaining fields and subfields.*
c-c	column-character positions in which to write data
lt	line type designator to write. The @WRL statement converts all updated lines to the line type specified in this subfield regardless of their original type. Period type lines always start writing in position 2 of the line--ignoring the position specified by the *c-c* position field.
v | *data*	content of variable or data to write

@LOK, @WRL, @ULK statement sequence

Precede all @WRL statements with a @LOK statement, except when a @WRL statement is used against a result, and follow the @WRL statement with a @ULK statement to release update control, as in this example:

```
@LOK,40,B,23 .
@WRL,40,B,23,2  5-6,60-3  *,V4,V5 .
@ULK .
```

In this example, the @LOK statement locks the report for update in mode 40, type B, RID 23. The @WRL statement writes an asterisk type line in line 2 of the report and places the value of variable V4 in column 5 for six characters and V5 in column 60 for three characters. The @ULK statement releases update control.