UNISYS

OS 1100

MAPPER® Run Design

Operations Reference Manual

Relative to Release
Level 34R1

Priced Item

June 1988

Printed in U S America
UP-9662.5

UNISYS

OS 1100

MAPPER® Run Design

Operations Reference Manual

Relative to Release
Level 34R1

Priced Item

June 1988

Printed in U S America
UP-9662.5

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the remark form in this manual, or remarks may be addressed directly to Unisys Corporation, MAPPER System Test and Publications, P.O. Box 64942 MS: 4792, St. Paul, Minnesota, 55164-0942, U.S.A.

Page Status

Page	Issue
ATM-1 through ATM-14	Original
Contents-1 through Contents-9	Original
Section 1 tab	Original
1-1 through 1-7	Original
Section 2 tab	Original
2-1 through 2-13	Original
Section 3 tab	Original
3-1 through 3-10	Original
Section 4 tab	Original
4-1 through 4-30	Original
Section 5 tab	Original
5-1 through 5-17	Original
Section 6 tab	Original
6-1 through 6-15	Original
Section 7 tab	Original
7-1 through 7-7	Original
A-E tab	Original
7-8 through 7-139	Original
F-O tab	Original
7-140 through 7-264	Original
P-Z tab	Original
7-265 through 7-414	Original
Appendixes tab	Original
A-1 through A-18	Original
B-1 through B-11	Original
C-1 through C-8	Original
D-1 through D-10	Original
E-1 through E-10	Original
F-1 through F-26	Original
Glossary/Index tab	Original
Glossary-1 through Glossary-35	Original
Index-1 through Index-15	Original

About This Manual

PURPOSE

This manual provides complete descriptions, formats, and examples for all run statements, as well as instructions for designing runs and getting them registered. By using this manual, the run designer can write and update complete runs, obtain quick access to statement syntax, and learn about efficiency techniques by following examples.

SCOPE

This manual contains reference information about run statements and offers complete sample runs. It is not meant to teach run design; it is designed to provide quick reference to run statement information.

AUDIENCE

The Run Design Reference is for all run designers—from beginning run designers to advanced users who design sophisticated applications.

PREREQUISITES

Before using this manual, beginning run designers should feel comfortable with using MAPPER software and should be familiar with the demonstration database. Ideally, the beginning run designer should understand basic run design concepts. To learn about writing runs, read the *OS 1100 MAPPER Run Design Operations Training Guide* or attend a MAPPER run design class.

HOW TO USE THIS MANUAL

If you are an experienced run designer, review "New Run Design Features" in Section 1. Then use Section 7 and the appendixes as references while writing runs.

If you are a novice run designer, read the first six sections before beginning to write your own runs. Then use Section 7 and the appendixes as references. In addition, you may want to review the sample runs in Appendixes B and C for more insight into MAPPER runs.

Until you're a registered MAPPER run designer, sign on as JDOER to practice writing runs. Ask your coordinator for a valid JDOER sign-on.

Use the EXAM run to check your MAPPER run-writing skills on line. See the *OS 1100 MAPPER Manual Functions Operations Training Guide* for more information.

Conventions

To help you understand this manual and find MAPPER software easy to use, certain style conventions are used. Following is a description of how this manual handles run statement syntax, tab characters, examples, key names, uppercase letters, italics, color, and important terminology.

Run Statement Syntax

The format of a MAPPER run statement consists of these conventions:

- The call is capitalized (for example, **CHG**). However, you can type it in either uppercase or lowercase letters.

- Fields and subfields are italicized whenever they call for ***variable*** data. Variable data is information you supply according to the explanation that follows the statement.

  NOTE: Field and subfield abbreviations are listed and described in Appendix A.

- 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 type all intervening commas. For example:

  ```
  @AUX,0,B,2,123,COP,Y,,,Y,,2 .
  ```

- Braces around items separated by a vertical bar mean that you may choose from among the items listed; for example:

  {*item1* | *item2*}

Tab Characters

A special character called a quadrate (□) always represents a tab character. You can't see tab characters on the screen, but when you place the cursor over a tab character, the cursor blinks.

Examples

The examples in this manual appear in uppercase letters; however, you can type them in lowercase. Some examples appear with brief comments similar to comments you might type in your run control report. These comments may start anywhere after the space-period-space that follows a run statement. For example:

```
@DLR,0,B,3 . DELETE RID 3B IN MODE 0
```

Some examples are presented with directory names as well as with traditional column-character syntax. In these cases, both statements are presented, followed by one set of field descriptions.

CAUTION: Be careful when trying examples that update the demonstration (JDOE) database provided on the release tape. If, for example, you want to try an example that deletes lines in RID 2B, be sure to use a duplicate report so you do not change RID 2B in the demonstration database.

Keyboard Key Names

The key names used in this manual are based on the UTS family of keyboards. If you need definitions for keys other than those on a UTS keyboard, refer to your terminal documentation.

Uppercase Letters

These items appear in uppercase letters:

- MAPPER functions and run statement calls (for example, CHG)
- MAPPER runs (for example, FCC run, RUNA run)
- MAPPER files (for example, MAPER1 file)
- Reserved words (for example, INPUT$)

Italics

Italics 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, and so on).
- Other italicized letters (for example, *x* and *y*) indicate user-supplied variables.
- In date formats, *y* or *Y* means year; *m* or *M* means month; and *d* or *D* means day.
- Words in ***bold italics*** are key terms that are defined in the glossary.

Color

Run statement formats and information that you are instructed to type are shown in color.

Enter and Resume

The term "enter" means type the necessary information and press **XMIT**.

"Resume" means press **F1** (you must use the **UPPER FUNCTION** key at the same time) or enter rsm.

ORGANIZATION

This manual contains seven sections and six appendixes:

Section 1. Introduction contains a list of new run design features and provides a brief description of MAPPER runs.

Section 2. Using the Data Directory describes the Data Directory and how to use it.

Section 3. Formulating Run Statements introduces you to run statement formats and guidelines. It also explains how to use labels and special characters in run statements.

Section 4. Variables and Reserved Words introduces you to variables and reserved words and explains how to use them in runs.

Section 5. Using Online Runs describes how to use HELP and other online run design aids.

Section 6. Designing and Debugging Runs discusses how to handle reports and results and how to design, register, and debug runs.

Section 7. Run Statements presents the MAPPER run statements in alphabetical order, along with their formats, field descriptions, and examples.

A. Summaries: Statements and Options lists all MAPPER run statement formats and field abbreviations.

B. Reserved Words provides a table of all reserved words, a detailed example using reserved words, and a discussion of how to use reserved words directly in run statements.

C. Sample Runs: DEMO, EDIT, and MARK provides three detailed sample runs.

D. Efficient Run Techniques offers suggestions for making your runs efficient.

E. **Character Sets** lists the Limited Character Set and the Full Character Set.

F. **Data Transfer Module** explains how to use the DTM interface through the QSND, QSNR, QRSP, QREL, and QCTL run statements.

RELATED PRODUCT INFORMATION

This manual is part of the MAPPER software level 34R1 library, which contains documents that you may find helpful while using MAPPER software. The following list provides the exact title of each document in the library, followed by its short title in parentheses and its previous title. The documents are listed in the order a new MAPPER site might use them. A separate list describes how to order copies of the MAPPER software level 34R1 manuals. In addition, there are several documents of related Unisys products that are referenced throughout this manual; these are listed under "Related Unisys Documents."

Documents in This Library

- *OS 1100 MAPPER Software Installation Guide*, UP-10786.9 **(Installation Guide)**

  Previous titles: *MAPPER Software Level 33R1, Installation Guide*
  MAPPER Software Level 33R1, Release Description

  This guide is for systems analysts who install and maintain MAPPER systems. It contains information previously found in the Release Description, such as new features for this software level, compatibility with other software and hardware, any restrictions that apply, and information about product support. It also lists the contents of the release tape and procedures used to install, configure, verify, start, and generate MAPPER software level 34R1.

- *OS 1100 MAPPER Software Operations Guide, Vol. 1: Coordinators*, UP-9194.6 **(Coordinator's Guide)**

  Previous title: *MAPPER Software Level 33R1, Coordinator's Reference*

  This guide is for MAPPER system coordinators. It describes their responsibilities and gives examples of the reports they use to establish and monitor a MAPPER system.

- *OS 1100 MAPPER Software Operations Reference Card: Coordinators*, UP-14074 **(Coordinator's Reference Card)**

  This reference card lists the most commonly used runs, functions, run statements, and reserved words for coordinators.

- *OS 1100 MAPPER Software Operations Guide, Vol. 2: Operators*, UP-9195.6 **(Operator's Guide)**

  Previous title: *MAPPER Software Level 33R1, Operator's Reference*

  This guide is for MAPPER system operators. It describes and gives examples of all operator tasks, including starting the MAPPER system, maintaining the system database, and creating recovery and history tapes.

- *OS 1100 MAPPER Software Operations Reference Card: Operators*, UP-14073 **(Operator's Reference Card)**

  This reference card lists directives used by MAPPER system operators.

- *OS 1100 MAPPER Manual Functions Operations Reference Manual*, UP-9193.6 **(Manual Functions Reference)**

  Previous title: *MAPPER Software Level 33R1, Software Reference*

  This manual is for users who have a basic knowledge of the MAPPER system. It provides comprehensive descriptions and examples of the MAPPER functions. It also includes an overview of form type design and run design.

- *OS 1100 MAPPER Manual Functions Operations Reference Card,* UP-9196.7 **(Manual Functions Reference Card)**

  This reference card provides a summary of formats for MAPPER functions and runs, along with the available options.

- *OS 1100 MAPPER Run Design Operations Reference Manual,* UP-9662.5 **(Run Design Reference)**

  Previous title: *MAPPER Software Level 33R1, Run Designer's Reference*

  This manual is for MAPPER run designers. It provides complete descriptions, formats, and examples for all run statements, as well as instructions for designing runs and getting them registered.

- *OS 1100 MAPPER Run Design Operations Reference Card,* UP-12999.1 **(Run Design Reference Card)**

  This reference card provides all MAPPER run statement formats, along with their field definitions and available options. It also lists reserved words used for run design.

- *OS 1100 MAPPER Word Processing Operations Guide,* UP-11619.1 **(Word Processing Guide)**

  Previous title: *MAPPER Software Level 33R1, Word Processing Guide*

  This guide is for all users of MAPPER word processing. It provides complete descriptions and examples of MAPPER word processing functions.

- *OS 1100 MAPPER Word Processing Operations Reference Card,* UP-13019 **(Word Processing Reference Card)**

  This reference card lists and describes all word processing control parameters, control characters, and commands.

- *OS 1100 MAPPER Color Graphics Operations Guide*, UP-11615.1 **(Color Graphics Guide)**

  Previous title: *MAPPER Software Level 33R1, Color Graphics Guide*

  This guide is for all users of MAPPER color graphics. It provides complete descriptions and examples of MAPPER color graphics functions, runs, and graphics codes.

- *OS 1100 MAPPER Color Graphics Operations Reference Card*, UP-13020 **(Color Graphics Reference Card)**

  This reference card lists all graphics runs, functions, primitive graphics code commands, and expanded syntax commands. It also contains GOC (Generate Organization Chart) commands and tables for colors, marker symbols, line patterns, and fill patterns.

- *OS 1100 MAPPER SCHDLR Interface Programming Reference Manual*, UP-11616.1 **(SCHDLR Reference)**

  Previous title: *MAPPER Software Level 33R1, SCHDLR Programmer's Reference*

  This manual is for COBOL programmers who want to use the SCHDLR interface. It provides the procedures and coding needed for a COBOL program to interface with the MAPPER system. This manual assumes you have knowledge of COBOL and Transaction Processing (TIP).

Optional Documents

These documents are not part of the standard MAPPER library and must be ordered separately.

- ☐ *OS 1100 MAPPER Manual Functions Operations Training Guide,* UP-13964 **(Manual Functions Training Guide)**

  Previous title: *A Guide to Using MAPPER Software*

  This guide helps beginners use MAPPER software productively. It provides an overview of what MAPPER software is and how it can be used, and it introduces the most commonly used MAPPER functions. For complete details on all MAPPER functions, see the Manual Functions Reference.

- ☐ *OS 1100 MAPPER Run Design Operations Training Guide,* UP-13965 **(Run Design Training Guide)**

  Previous title: *A Guide to Creating MAPPER Software Runs*

  This guide is for users who have never written a run. It covers only basic information and should be read and followed, step by step, at a MAPPER terminal. When you need more details than are given in this manual, see the Run Design Reference.

- ☐ *OS 1100 MAPPER Software Operations Quick-Reference Guide,* UP-11628.1. **(Quick-Reference Guide)**

  This guide is a handy, durable summary for all users of MAPPER systems. It contains MAPPER functions, run statements, function and run statement options, and reserved words. It also lists color graphics and word processing information, commands for the MAPPER system coordinator, and directives for the operator.

- *OS 1100 MAPPER Software Operations Guide, Vol. 3: Using an IBM®* 3270 Terminal, UP-11632.1 **(Using an IBM 3270 Terminal Guide)**

  Previous title: *MAPPER Software Level 33R1, Using an IBM 3270 Terminal*

  This guide explains how to use MAPPER software from an IBM 3270 series terminal (or equivalent). It shows examples for signing on to the MAPPER system and using MAPPER software in one of the two modes of operation, native or UTS emulation. It also lists considerations for MAPPER run design and word processing.

- *OS 1100 MAPPER New Features Operations Reference Manual,* UP-11631.1 **(New Features Reference)**

  Previous title: *MAPPER Software Level 33R1, Summarizing Level 33R1 Features*

  This manual provides an overview of MAPPER software level 34R1. It is a handy summary of new features for users who have previous experience with MAPPER software.

Ordering MAPPER Software Level 34R1 Documents

The MAPPER documentation is ordered by PL, PK, and UP numbers:

- Use the PL (Product Library) number to order an entire standard library. Note that this does not include the optional MAPPER documentation.

- Use the PK (Package) number to order an individual manual with its binder.

- Use the UP number to order documents that do not have binders.

IBM is a registered trademark of International Business Machines Corporation.

Use the following PK, PL, and UP numbers to order MAPPER documentation.

Standard Library

Reference cards are included in the binders of the corresponding manuals. To order additional copies of the reference cards, order them by the UP number.

PL-0284	MAPPER Software Level 34R1 Standard Library
PK-1328	Manual Functions Reference
UP-9196.7	Manual Functions Reference Card
PK-1329	Run Design Reference
UP-12999.1	Run Design Reference Card
PK-1330	Word Processing Guide
UP-13019	Word Processing Reference Card
PK-1331	Color Graphics Guide
UP-13020	Color Graphics Reference Card
PK-1333	Coordinator's Guide
UP-14074	Coordinator's Reference Card
PK-1332	Operator's Guide
UP-14073	Operator's Reference Card
PK-1334	SCHDLR Reference
PK-1335	Installation Guide

Optional Documents

These are optional documents that do not come with the standard library and must be ordered separately:

PK-1834	Using an IBM 3270 Terminal Guide
PK-1336	New Features Reference
PK-1835	Manual Functions Training Guide
PK-1836	Run Design Training Guide
UP-11628.1	Quick-Reference Guide

Related Unisys Documents

These Unisys manuals are referred to in this documentation, but they are not part of the MAPPER library. Use the version that corresponds to the level of software in use at your site.

- ☐ *OS 1100 Distributed Data Processing (DDP-PPC/DDP-FJT) Messages Reference Manual,* UP-13510 (**DDP-PPC/DDP-FJT Messages Reference Manual**)

  This reference contains information about all messages and error codes for DDP-PPC and DDP-FJT.

- ☐ *OS 1100 Distributed Data Processing File and Job Transfer (DDP-FJT) Operations Guide, Vol. 1: IPF Interface,* UP-9740.3 (**DDP-FJT Operations Guide, Vol. 1: IPF Interface**)

  This guide describes how to transfer files and jobs from terminals in a DDP network.

Contents

Appendixes

Tables

1. Introduction

This manual contains reference information about new and existing run statements. By using this manual, you can write and update complete runs, obtain quick access to statement syntax, and learn about efficiency techniques by following examples.

This section contains:

- New Run Design Features
- What Is a MAPPER Run?

New Run Design Features

MAPPER software level 34R1 contains many new run statements, enhancements to existing run statements, reserved words, and other new run design features. Following are brief descriptions of these new features.

NEW RUN STATEMENTS

BRG (Break Graphics)
: Packs data, such as primitive graphics code, in the output area and places it into a result.

DCR (Decode Report)
: Transforms an encoded report into a readable report by specifying the key. See ECR.

DSM (Display Message)
: Allows you to display your own one-line message at the top of the screen.

ECR (Encode Report)
: Encodes a report, changing the data from readable text into code that can be read only if a key is specified.

RDB (Run Debug)
: Halts the run and enters into debug mode, which allows you to interactively examine the contents of variables, reserved words, and renamed results.

SC (Screen Control)
: Allows you to create menus and other input screens or to edit text already on the screen. You can also use it to overlay existing OUT or DSP screens.

STN (Station Information)
Provides you with information about a specific station number. Use it when sending data to another terminal via a DSG, OUT, or SC statement.

USE (Use Named Variables)
Allows you to assign a variable name to a specific variable number.

ENHANCEMENTS TO EXISTING STATEMENTS

ADR Allows you to specify the RID number to add in a particular mode and type. You can also enter the label or relative line number to go to in case of an error.

DEF Added the V option to define the variable name that is assigned to a specific variable number.

DLR Allows you to delete results as well as reports.

DSG Allows you to send DSG displays to other terminals. You can specify a label or line number to go to in case of error.

DUP Allows you to specify a different mode and type to duplicate a report into.

LDV Allows you to specify only the receiving variable when loading a variable with its own contents.

MAU Added the A option to allow match updates on all line types.

MCH Added the A option to allow matching all line types.

OUT Allows you to send OUT displays to other terminals. You can specify a label or relative line number to go to in case of error.

REP Allows you to specify a particular receiving report to be created by the REP statement.

SRH Added the *vrid* variable to capture the RID number where the find is made when doing a range search. This is useful when using the B(*n*) option.

TYP Added the *vrlmt* variable to capture the highest RID number allowed in the form type; the *vllmt* variable to capture the highest number of lines allowed per report in the form type; and the *vrids* variable to capture the total number of reports in the form type.

NEW RESERVED WORDS

ELINE$ Contains the line number currently being executed in a run control report.

MAXTYP$ Contains the maximum octal form type available on your MAPPER system.

MSEC$ Contains the current number of milliseconds since midnight.

OTHER NEW FEATURES

- The RDB (Run Debug) function and run statement allow you to execute a run interactively and examine the contents of variables, reserved words, and renamed results. You can step through the run one line or command at a time; or you can set a breakpoint to halt the run at a specific line number, label, run statement, or variable.
- Type A variables now allow up to sixteen characters rather than twelve.
- The new LIMITS run displays the highest RID number and lines per report that are allowed for the mode and type you are currently in.
- The SCHEDULE run can now be called from a run control report using the LNK (Link to Another Run) statement.

- Chart runs can now be called from a run control report using the LNK (Link to Another Run) statement.

- Variables can now be assigned meaningful names rather than numbers. These variable names are enclosed in greater than and less than signs; for example, you can use <Finds> rather than V1.

- The DEMO, EDIT, and MARK sample runs have been updated to show the use of named variables.

- The BVT (Build Variable Table) run allows you to build or rebuild a variable table displaying the location of variables and convert variables in your run control report. The BVT run replaces the VAL run, which is no longer supported.

- The RUN run now includes function keys that allow you to control the operation of runs and a screen that allows you to display, execute, and register results or exit the run.

- You can use zeros in substring variables to specify two kinds of trailing substrings. The first kind is a ***known trailing substring***, in which you indicate the starting character position and use the remaining characters in the field. The second kind is an ***unknown trailing substring***, in which you don't know the starting character position, but indicate the number of ending characters to use.

- Run statements now check the specified report to determine whether the requested column exists. If the column number does not exist, the run errs rather than returning the message "No more finds." This enhancement does not apply to four run statements: LFN (Load Field Name), RDC (Read Continuous), RDL (Read Line), and RLN (Read Line Next).

- The HELP run contains a new option for run designers, HELP *@rfc*, where *rfc* is the run function call. When this is entered with a run control report on display, the format of the specified run function call is displayed on the control line. You can obtain additional help about that run statement by pressing **F1** or entering rsm.

- The FKEY$ reserved word now allows you to capture the number of up to 22 function keys.

DISCONTINUED RUNS

- The FORMD run has been discontinued. However, you can now use the new HELP *@rfc* to display a run statement format.

- The VAL run has been discontinued and is replaced by the new BVT run.

What Is a MAPPER Run?

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

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

You can make logical decisions based on variables or results (for example, jumping, branching, and decisions based on data content). You can also design MAPPER runs that run users can interrupt to display data on the screen or enter information to be captured and processed by the run.

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

2. Using the Data Directory

The ***Data Directory*** is a run design tool that allows you to dynamically identify fields, modes, form types, and reports in run statements by naming them.

Database naming has its own special syntax; however, it does not replace standard run syntax. You can use both conventions in the same run, even within the same run statement.

This section includes:

- ☐ Naming Fields
- ☐ Naming Modes, Form Types, and Reports
- ☐ Naming Data Using Reserved Words

Naming Fields

You can process fields in reports by name. The names are derived either from the headers of the report, or from RID 0 if you're processing the entire form type. Field names, therefore, are an integral part of the database.

For example, in standard run syntax, this statement searches the report for customer code AMCO:

```
@SRH,0,D,1 '' 26-4 □,AMCO .
```

Using named fields, this statement does the same thing:

```
@SRH,0,D,1 '' 'CUST CODE' □,AMCO .
```

You don't have to specify the starting column number or the field size; only the field name is required.

If a field moves within a report or if the size of the field changes, you don't have to change the reference to it.

REPORT HEADERS AND THE HEADER-DIVIDER LINE

When a run reads the first named field in a statement, it scans the report for a header-divider line (*=). The report must contain a header-divider line within the first 16 lines of the report.

The run derives starting columns and field sizes from the grouping of equal signs on the header-divider line, extracting field names from up to two asterisk lines immediately preceding it. If the report has more than two asterisk lines, the run recognizes only the last two.

Examples

This example shows the first five fields in RID 2B in mode 0:

```
*ST.STATUS.BY. PRODUCT .SERIAL.
*CD. DATE .IN.  TYPE   .NUMBER.
*==.======.==.========.======.
```

From these fields, the following field names are derived:

ST CD (2-2)
STATUS DATE (5-6)
BY IN (12-2)
PRODUCT TYPE (15-9)
SERIAL NUMBER (25-6)

This example shows the first fields of a report with three asterisk lines containing field headers:

```
* MONTHLY. ANNUAL .
*INTEREST.DISCOUNT.
*  RATE  . STATUS .
*========.========.
```

The field names are:

INTEREST RATE (2-8)
DISCOUNT STATUS (11-8)

Therefore, if you're using named fields for reports with three or more asterisk type header lines, keep the important information in the two header lines immediately preceding the header-divider line.

FIELD NAMES

Enclose field names within apostrophes ('). Names can be either uppercase or lowercase and can contain from 1 to 32 characters. Field names must be unique within the report header; if duplicate names are present, the run uses the leftmost one.

You can use any characters in your field names. However, the run considers only alphanumeric characters (A to Z and 0 to 9) when comparing the field name to the report header; it ignores any other characters, such as spaces or special characters. For example, 'CUSTCODE', 'CUST-CODE', and 'CUST CODE' are all acceptable field names.

You can also abbreviate field names by omitting trailing characters, as long as the characters specified are unique to that field.

For example, you can specify a search of the CUST CODE field like this:

```
@SRH,0,D,1 '' 'CUST' □,AMCO .
```

This makes the run statement shorter, which generally makes it more efficient.

FIELD NAMES IN VARIABLES

Enclose the name of a variable within apostrophes if it contains a field name. Do not place any other characters, such as spaces, within apostrophes. You can use any variable type, including a variable-variable designation. However, you cannot use substrings of variables.

This example searches the CUST CODE field for AMCO; notice that there are no spaces or other characters between the apostrophes in 'V1':

```
@LDV V1H18='CUST CODE' .
@SRH,0,D,1 '' 'V1' □,AMCO .
```

NAMING PARTIAL FIELDS

You can also specify a partial field. For example, you may want to scan a field for specific starting or ending characters.

To name a partial field, enclose the relative starting column position and number of characters of the partial field in parentheses after the field name.

For example, this statement searches the first character of the PRODUCT TYPE field:

```
@SRH,0,D,1 '' 'PRODUCT TYPE(1-1)' □,B .
```

To process a named field from any column to the end of the field, specify the starting column and define the number of characters as zero.

For example, this statement searches the PRODUCT TYPE field starting at the sixth column in the field for the remainder of the field:

```
@SRH,0,D,1 '' 'PRODUCT TYPE(6-0)' □,BOX1 .
```

To name the trailing portion of a named field, specify a starting column of zero and the number of characters to process.

This example searches the last character of the ORDER NUMBER field:

```
@SRH,0,D,1 '' 'ORDER NUMBER(0-1)' □,S .
```

FIELD ORDER

You can list multiple named fields in any order; they don't have to be in the same order in which they appear in the report, as long as the parameters and named fields are in the same order.

For example, these two statements perform the same search and have identical results:

```
@SRH,0,D '' 'ST CD','CUST CODE' □,OR,AMCO .

@SRH,0,D '' 'CUST CODE','ST CD' □,AMCO,OR .
```

FIELD SIZE VARIABLE DEFINITION

Whenever an input parameter is used to process a report field, the variable used for input must match the field size. The input field size on a screen may also need to match a corresponding report field.

You can use the DVS (Define Variable Size) statement to create variables equal to the size of the report fields (see DVS in Section 7). When the run executes, it defines the size of the variable; any input parameter or screen using that variable dynamically adjusts to a change in the size of the field.

You can also define the variable to a field size in these statements that load variables with data from report fields: RDC (Read Continuous), RDL (Read Line), RLN (Read Line Next), and SUB (Subtotal). See RDC, RDL, RLN, and SUB in Section 7.

SELECTING FIELDS TO DISPLAY

Use the FMT (Format) statement to select which fields you want to display in a following DSP, OUT, or OUM statement (see FMT in Section 7).

CONVERTING TO FIELD NAMES

Use the LFN (Load Field Name) statement to translate standard column number syntax into field names (see LFN in Section 7). You can use it to convert existing run statements to use field names, or to translate data from the OUM statement into field names (see OUM in Section 7).

EFFICIENCY CONSIDERATIONS

When a run encounters a field name, it must read the report header, which requires one additional I/O access. However, it does not read the header for other field names in the same run statement. Also, succeeding run statements that specify the same report, or a result derived from it, do not cause the run to read the report header again.

In this example, the SRH statement causes the run to read the headers; the SOR and TOT statements do not.

```
@SRH,0,D,1 '' 'ST CD' □,OR .
@SOR,0,D,-0 '' 'ORDER NUMBER' □,1 .
@TOT,0,D,-0 S 'ORDER NUMBER','ORD QTY' □,S,+ .
```

You typically reprocess the result of a previous function. This distributes the overhead of reading report headers and minimizes its impact on any individual statement.

Naming Modes, Form Types, and Reports

You can identify a mode, a form type, or a report by name in a run statement.

In standard run syntax, for example, this statement searches mode 0, RID 1D:

```
@SRH,0,D,1 '' 'CUST CODE' □,AMCO .
```

Using a named report, this statement does the same thing:

```
@SRH,'ORDER STATUS' '' 'CUST CODE' □,AMCO .
```

You don't have to specify the mode, form type, and report number — only the name of the report.

Mode, form type, and report names are defined in the System Directory, which you can access and update using the NAME run. (See "NAME — Updating the System Directory" in this section.) Before using a named mode, form type, or report, you must enter it in the System Directory.

Depending on how it is defined, this name replaces one or more of the mode, type, and RID (*m,t,r*) subfields in a run statement. If the name defines a mode, it replaces only the first subfield. A name that defines a report replaces all three subfields.

You can also assign a name to a range of reports in a form type. With BFN (Binary Find), FND (Find), and SRH (Search), the system automatically adds the R option to the run statement if you use a name that defines a range of reports.

MODE, FORM TYPE, AND REPORT NAMES

Enclose the names of modes, form types, and reports within apostrophes ('). Names must start with an alphabetic character (A to Z), can be either uppercase or lowercase, and can contain from 1 to 16 characters.

You can use any characters in your data names. However, the run considers only alphanumeric characters (A to Z and 0 to 9) when comparing the name to the System Directory; it ignores any other characters, such as spaces. For example, 'ORDERSTATUS', 'ORDER-STATUS', and 'ORDER STATUS' are all acceptable names.

NAMES IN VARIABLES

Enclose the name of a variable in apostrophes if it contains a mode, form type, or report name. Do not place any other characters, such as spaces, within the apostrophes. You can use any variable type, including a variable-variable designation. You cannot use substrings of variables, however.

This example searches the ORDER STATUS report for AMCO in the CUST CODE field; notice that there are no spaces or other characters between the apostrophes in 'V1'.

```
@LDV V1H18='ORDER STATUS' .
@SRH,'V1' '' 'CUST CODE' □,AMCO .
```

NAMING RESULTS

Wherever you specify the current result (-0) or a renamed result (-1 to -4), you can omit the mode and type (*m* and *t*) fields.

For example, instead of the following statement:

```
@DSP,0,B,-0 .
```

you can use this statement to do the same thing:

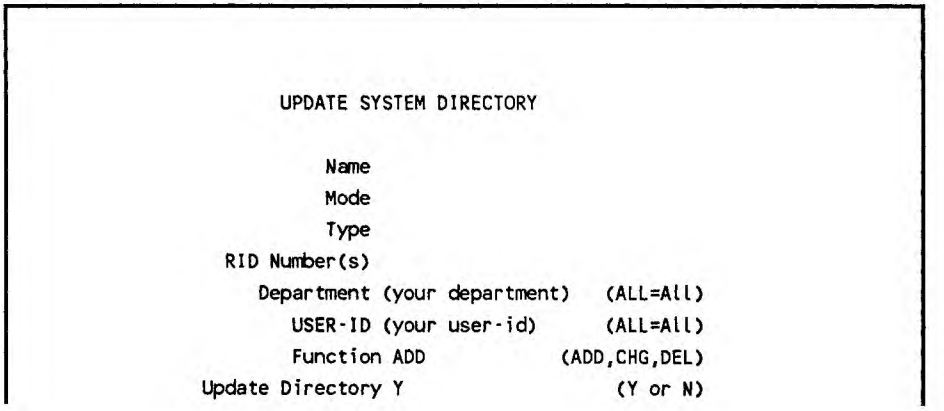
```
@DSP,-0 .
```

NAME — UPDATING THE SYSTEM DIRECTORY

The ***System Directory*** is a report that contains data names for modes, form types, and reports. You can use data names in run statements to define modes, types, and reports to process.

The NAME run updates the System Directory. You can enter a new name, delete an existing name, or change the definition of an existing name.

To use the NAME run, enter **name** to display this screen:



In field:	Enter:
Name	the data name (up to 16 characters, beginning with an alphabetic character). Only alphanumeric characters are stored in the System Directory; any other characters entered in this field are ignored.
Mode*	the mode number.
Type*	the form type (A to I).
RID Number(s)*	the RID number, or a range of RIDs defined by a lower-higher designation (1-12, 500-999, and so forth).
Department	the department number qualifier. (Default = user department.) If you want to allow all departments to use the name, enter ALL.
USER-ID	the sign-on. (Default = your user-id.) If you want to allow all users to use the name, enter ALL.
Function	the type of update to the System Directory, where ADD = add new item (default), CHG = change the definition of an existing item, and DEL = delete an existing item. Note that only the user who entered a name can change or delete it.
Update Directory	a Y to include the name in the System Directory. If you are naming several reports, it is more efficient to enter N for all but the last report. You must enter a Y for the last report to include all reports that you named.

* To name a mode, leave the Type and RID Number fields blank. To name a form type, leave only the RID Number field blank. When deleting a name, you can leave the Mode, Type, and RID Number fields blank.

To use the fast-access method, use this format:

NAME *name*[*,m,t,r,dept,user,func,update?*]

In field:	Enter:
name	the data name.
m,t,r	the mode, form type, and RID number. (Default = current -0.)
dept	the department number. (Default = user's department.)
user	the user's sign-on. (Default = your user-id.) Enter ALL to allow all users to access the name.
func	the function, where ADD = add (default), CHG = change, DEL = delete.
update?	a Y to include the name in the System Directory. (Default = Y.)

SYSTEM DIRECTORY INFORMATION

The DIR (Directory Information) statement allows you to retrieve information about a particular data name from the System Directory. (See DIR in Section 7.)

Naming Data Using Reserved Words

You can use reserved words directly in run statements, which means you don't have to load them in variables beforehand. Because many reserved words represent data entities, such as form types, this is also a form of data naming.

For example, instead of the following sequence:

```
@CHG V114 MODE$ .
@CHG V216 TYPE$ .
@CHG V314 RID$ .
@DSP,V1,V2,V3 .
```

you can use just one statement to do the same thing:

```
@DSP,MODE$,TYPE$,RID$ .
```

In addition, you can use reserved words that represent numeric form types to replace the mode and type (*m* and *t*) fields of run statements.

For example, this statement can replace the preceding example:

```
@DSP,TYPE$,RID$ .
```

3. Formulating Run Statements

MAPPER run statements begin with an at sign (@) and follow a specific format. The next page shows a typical run statement format. In addition, run statements often contain labels and special characters.

This section includes:

- ☐ Run Statement Format
- ☐ Labels
- ☐ Special Characters

Run Statement Format

Here's a typical MAPPER run statement format:

```
@label:function-call,mode,type,rid options column-characters
△line-type,parameters variables . comments
```

Note the space-period-space before the comment. Your runs execute more efficiently if you type a space-period at the end of each statement and a space before typing a comment.

This SRH (Search) statement uses data from mode 0, type C, report 1:

```
@7:SRH,0,C,1 D 2-9 □,BLACKBOX7 V115,V216 .
```

where:

@	control character
7:	label and separator (optional)
SRH	function call
0	mode number
C	form type of report
1	report identifier (RID) number
D	D option (deletes search information lines)
2-9	column-character positions to process: column 2 for nine character positions
□	line type: process tab lines (you can enclose the invisible tab character in apostrophes if you want to remind yourself that it is there)
BLACKBOX7	parameter: search for BLACKBOX7s

V1I5	five-digit variable named V1, which captures the number of finds
V2I6	six-digit variable named V2, which captures the 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 run statement line; the line must have a valid statement. If the system finds something invalid in a line, it responds with an error message like one of these:

⌜UNABLE TO FIND ALL THE FIELDS REQUIRED⌝

⌜THIS CONTROL WORD IS NOT VALID⌝

FORMULATING RUN STATEMENTS

Follow these guidelines when formulating run statements:

- Type an @ in column 1.
- Enter multiple run statements on one line and separate them with spaces. Use just one @ per line.
- Terminate a line with a space-period-space (△.△). Beginning a line with an at sign-period (@.) or an at sign-label-period (*@label* .) also terminates the line. 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 system ignores statements that follow them on the same line): DSP, ESR, LNK, OUT, RDB, RRN, RSR, RTN, RUN, SC, and WAT.

- You can specify fields to process in the *cc* (column-characters) subfield in any order, regardless of their order in the report, as long as you list the parameters in the same order. For example, these two run statements work the same:

```
@SRH,0,C,1 '' 2-5,16-1 □,BLACK,A .

@SRH,0,C,1 '' 16-1,2-5 □,A,BLACK .
```

- Define columns only once in the *cc* (column-characters) field.
- Terminate fields with spaces. A comma does not terminate a field.
- Separate subfields with commas. This includes 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 they correspond to.
- You can use variables in fields and subfields.
- Never exceed 19 variables in a field or subfield. In some cases, the maximum number of variables allowed is fewer than 19 and noted as such.
- Enclose fields or subfields that require significant spaces within apostrophes.
- The required fields and subfields for a run statement vary from statement to statement. Include all required fields. Enter two apostrophes ('') if you're not entering options in the *o* (options) field.
- To get the literal representation of variables in the output area, enclose them within apostrophes (for example, **'V1'**).
- Do not use an at sign or a colon (@ or :) in the first character position of any line in the output area.

- Whenever you use a comma for something other than a subfield separator, enclose it within apostrophes; for example, to include a comma in the replacement string of an LCH (Locate and Change) statement, use `'change, and'`.

- In run statements that require an issuing and a receiving report, designate the issuing report first and the receiving report second.

- Run statements that access or lock reports (IDU, LOK, RDL, SOR, SRH) cannot access their own run control report.

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

Labels

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

- At the start of a run statement to match a label specified in the *lab* subfield of another run statement
- In logical IF GTO statements
- When you want to identify sections in a run

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

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

Format

@n: (followed by a statement)

or:

@n .

where *n* is a line label number from 1 to 199 (or 1 to 399 if your system is set up to handle up to 399 labels; check with your coordinator).

Each label must be unique; duplicates are not allowed.

LABEL TABLE DEFINITION LINES

Label table definition lines predefine label locations (that is, they tell the run which lines have labels).

Use label table definition lines in larger, more stable production runs, but not in runs that change frequently.

Here's an example of a label table:

```
:L 22=13,33=26,44=39
```

Label 22 is on line 13, label 33 is on line 26, and label 44 is on line 39.

Format

:L *label-number=line-number, . . .*

where:

label-number	is the label number.
line-number	is the line number in the run control report where this label is located.

Use BLT to build label tables. See BLT in Section 7. See also the BLT function in the Manual Functions Reference.

Special Characters

Use these special characters in MAPPER runs:

- Semicolon (;) as a field delimiter
- Slant (/) to indicate multiple parameters (for example, in a range search)
- Reverse slant (\ ) for continuing a run statement on the next line
- Apostrophe (') to specify literal data in parameters fields of run statements

SEMICOLON — FIELD DELIMITER

In an ART or CAL statement, use a semicolon (;) to separate expressions. For example:

```
@ART V2+V3;V4*4 V10|12,V11|12 .
```

and:

```
@CAL,0,C,1 L 50-5,56-8,65-15 □,A,B,C \
C=A*B;AVRG=VAVG(A) .
```

In an IF statement, use a semicolon to control more than one decision on the same line. For example:

```
@IF V1 = 3 GTO 9 ; IF V1 = 6 GTO 8 ; GTO END .
```

See GTO and IF in Section 7 for more examples.

SLANT — MULTIPLE PARAMETERS

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

```
@SRH,0,B,2 D 2-2 □,OR/□,SC V113,V213 .
```

REVERSE SLANT — CONTINUE STATEMENT

Use a reverse slant (\) whenever a run statement is too long for one line. For example:

```
@CAL,0,C,1,,,99 L 25-7,33-8,65-15 □,A,B+,C+ \
MAXA=VMAX(A);MAXB=VMAX(B);C=B-A V119,V219,\
V319,V419 .
```

The reverse slant at the end of the first line tells the system that this statement continues on the second line.

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.

You can use up to 640 characters in a run statement on multiple lines. The system counts all characters in the last line, including unused spaces.

APOSTROPHE — LITERAL DATA

Normally, characters in the parameters fields of run statements need not be enclosed in apostrophes. The following characters, however, must be enclosed in apostrophes:

- Spaces; for example:

```
@SRH,0,D,1 D 'CUSTOMER' □,'DIGITAL CORP'.
```

- Slants; for example:

```
@SRH,0,D,1 / 'CUSTOMER' □,'UNION STEEL/SULFR' .
```

- Commas; for example:

```
@SRH,0,C,1 F 'PRODUC COST' □,'13,500' .
```

4. Variables and Reserved Words

Variables and reserved words are important aspects of run design, so you should know how to use them properly and efficiently. This section defines variables and reserved words and tells why you use them in run design. It also presents two runs (VARIABLE and BVT) that you use with variables.

This section includes:

- ☐ Variables — Names, Types, and Sizes
- ☐ Initializing and Redefining Variables
- ☐ Changing the Contents of Variables
- ☐ Using Exponential Notation with Variables
- ☐ Examples Using Variables
- ☐ Loading Variables with Screen Input and Initial Input Parameters
- ☐ VARIABLE Run — Testing Contents of Variables
- ☐ BVT Run — Building Variable Tables and Converting Variables
- ☐ Reserved Words

Variables—Names, Types, and Sizes

A ***variable*** is a labeled entity that can assume different values. These values are assigned by you or by the system.

When you use a variable, you must first initialize it. This means that you assign it a name, a variable type, a size, and an initial value.

NAMING VARIABLES

You can name variables in two ways:

- You can use the traditional naming conventions, in which you name a variable with a V followed by a number from 1 to 199. (You can use up to 399 variables if your system is set up to handle more than 199. Ask your coordinator for the maximum number of variables allowed at your site.) You refer to the variable with the letter V and its number (for example, V10).

- You can use the variable-naming method, in which you assign a meaningful name for a variable rather than a number. The variable name can be no greater than twelve characters, it must begin with an alphabetic character (A-Z), and contain only aphanumeric characters (A-Z and 0-9). You enclose the name in less than (<) and greater than (>) signs (for example, <NAME>).

  You can use these named variables anywhere you would use numbered variables in a run. When a named variable is defined, the system assigns it the lowest unused variable number. Therefore, the first variable in a run is assigned to V1, the second is assigned to V2, and so on. After a name is assigned to a variable, it can no longer be referenced by its variable number.

Mixing named and numbered variables in the same run is not recommended. However, if it is necessary to do so, you can use the USE run statement, which allows you to assign a name to a specific variable number. See USE (Use Variable Name) in Section 7.

NOTE: Named variables are slightly less efficient than numbered variables. Although the difference is small, you may want to consider it when using logic-intensive runs.

ASSIGNING VARIABLE TYPES AND SIZES

When you use a variable for the first time, you must assign it a type and a size. There are six types of variables:

A	Alphanumeric
F	Fraction
H	Hollerith (any characters)
I	Integer (whole numbers)
O	Octal
S	String

You specify the size by using a number (or another variable) to indicate the number of characters the variable can have.
When you initialize a variable, you specify its name, its type, and its size.

For example, this means that V9 is an integer variable with three characters:

```
V9I3
```

Using the new variable-naming conventions, this means that <PHONE> is an alphanumeric variable with eight characters:

```
<PHONE>A8
```

Table 4-1 shows types and sizes of variables used in ART, CHG, and IF statements.

USING VARIABLES

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, this means start at character position 1 for three characters:

```
V10(1-3)        or        <PHONE>(1-3)
```

NOTE: Substrings of H, I, and S type variables are treated as A type variables. See Table 4-1 for information about types, sizes, and limitations of variables.

You can also use zeros in substrings to specify ***known trailing substrings*** and ***unknown trailing substrings***. To use a known trailing substring, you specify the starting character position, then a zero to indicate the remaining characters of the field. For example, this means start at character position 3 for the remaining characters in V1:

```
V1(3-0)
```

To specify an unknown trailing substring, you specify zero because you don't know the starting character position; then specify the number of ending characters to use. For example, this means to use the last two characters of V1:

```
V1(0-2)
```

You can use another variable that has a number from 1 through 199 to name a variable. For example, if V1 contains the value 2, this references variable V2:

```
VV1
```

You can also use the new variable-naming conventions to reference a variable with another variable. You specify this by enclosing the variable name within two less than and greater than signs. For example, if variable <ONE> contains the value TWO, this references variable <TWO>:

```
<<ONE>>
```

You can use another variable that contains 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 statement syntax and special commands, you are better able to determine where a variable might be useful.

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

- In any field or subfield in run statements.
- In the run's output area. See BRK in Section 7.
- As counters and checks for logical decisions. See IF in Section 7.

Table 4-1 lists the maximum size, examples, and contents of each type of variable. It also describes the contents of each type when used with ART, CHG, and IF statements.

Table 4-1. Variables: Types, Sizes, Limitations

Type A (Alphanumeric):

Maximum Size:	16
Examples:	V10a16 <PHONE>a8
Contents:	Alphanumeric and special characters.
ART:	Must have exclusively numeric characters.
CHG:	For arithmetic, numeric characters produce numeric answers and change letters and special characters to the next letter in the character set.
IF:	Treats spaces and some special characters (. + -) as zeros.

Type F (Fraction):

Maximum Size:	18
Examples:	V10f18.10 <TOTAL>f18.10
Contents:	Fractional numbers, positive and negative. Positive numbers may be unsigned; that is, they don't need a plus sign (+). The maximum size of 18 includes the sign and decimal point. The fractional portion may be up to ten characters. In the above example, 18 characters are allowed — 7 before the decimal point, the decimal point, and 10 after the decimal point. If more positive numbers are added to the left of the decimal point, the decimal portion is truncated.
	ART, CHG, and IF are allowed.

(continued)

Table 4-1. Variables: Types, Sizes, Limitations (cont.)

Type H (Hollerith):	
Maximum Size:	18
Examples:	V10h16 <CODE>h18
Contents:	Hollerith, any characters.
ART:	Not allowed.
CHG:	Changes all characters to the next character in the set, including numbers.
IF:	Characters must be identical to satisfy a true condition.

Type I (Integer):	
Maximum Size:	16
Examples:	V10i12 <FINDS>i3
Contents:	Integer (whole numbers), positive and negative. Positive numbers may be unsigned; that is, they don't need a plus sign (+). Include the sign in the variable size — it is a significant character.
	ART, CHG, and IF are allowed.

(continued)

Table 4-1. Variables: Types, Sizes, Limitations (cont.)

Type O (Octal):

Maximum Size:	12
Examples:	V5o12 <TYPE>o12
Contents:	Numbers 0 through 7.
ART:	Not allowed.
CHG:	For arithmetic, produces octal answers.
IF:	Considers the number's decimal value, not its octal representation.

Type S (String):

Maximum Size:	132
Examples:	v10s132 <ADDRESS>s40
Contents:	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.

Full redefinition of string variables, of both size and type, is allowed.

Initializing and Redefining Variables

You can initialize and redefine variables in these ways:

- ☐ With an LDV statement
- ☐ With a CHG statement
- ☐ By way of another statement

NOTE: You can also initialize and redefine variables with a colon. However, this method is no longer recommended because it is less efficient and more difficult to use.

USING AN LDV STATEMENT

Using an LDV statement is the most efficient way to initialize and load a variable, as in these examples. The first LDV statement initializes V1 to 1; the second redefines V1 to A and initializes V2 to 10:

```
@LDV V1I2=1 .

@LDV V1A1=A,V2I2=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 .
```

This example shows how to name a variable with the contents of another variable. If V2 contains 10, the following initializes V10 to XYZ:

```
@CHG VV2A3 XYZ .
```

This example also names a variable with the contents of another variable. If <ONE> contains the value TWO, the following statement initializes variable <TWO> with 2:

```
@CHG <<ONE>>I2 2 .
```

Redefined variables lose their previous content. In the following example, the first two CHG statements initialize V10 and V11, and the last two CHG statements redefine them:

```
@CHG V10A3 AAA CHG V11A1 A .

@CHG V10F4.2 1.23 CHG V11I4 1234 .
```

INITIALIZING VARIABLES WITH OTHER STATEMENTS

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

This example shows how to use an RDL statement to place the data in column 5 for six characters in <DATE> and the data in column 71 for five characters in <ORDER>:

```
@RDL,0,B,2,<LINE>I2,99 5-6,71-5 <DATE>I,<ORDER>H .
```

This example uses a FND statement to place the RID number of the report where the find was made in <RID> and the line number in <LINE>:

```
@FND,0,B '' 'ST-CD' □,IP <RID>I6,<LINE>I6 .
```

This example uses a TOT statement to place the number of lines processed in <LINES> and the sum of the values totalized in <SUM>:

```
@TOT,0,C,1 E 42-7 □,+ <LINES>I4,<SUM>I7 .
```

Changing the Contents of Variables

You can change a variable's content several ways, as in these examples. Note that these examples are shown as if in sequential order in a run.

```
@LDV V10A3=AAA . INITIALIZE V10 TO AAA

@LDV V10=BBB . CHANGE V10 TO BBB

@LDV V11I3=123 . INITIALIZE V11 TO 123

@INC V11 . INCREASE V11 BY 1

@LDV V12S15=0123456789ABCDE .
```

The following statement loads the first three character positions of V12 with V10:

```
@LDV V12(1-3)=V10 . V12 = BBB3456789ABCDE
```

Note that this INS statement does the same as the preceding LDV statement, but it is slower and less efficient:

```
@INS V10 V12(1-3) .
```

The following statement loads two characters of V12 starting in column 14 with two characters of V11 starting in column 2:

```
@LDV V12(14-2)=V11(2-2) . V12 = BBB3456789ABC24
```

This LDV statement initializes V13 to 4 and changes V10 to equal the three character positions starting in column V13 of V12:

```
@LDV V13I1=4,V10=V12(V13-3) . V10 = 345
```

Using Exponential Notation with Variables

Type A, type I, and type F variables can 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 system changes the value to exponential notation if variables are defined as A, I, or F type. For example:

```
@CHG V2I8 12345678 * 10 . V2 EQUALS 1.234E+8
```

Examples Using Variables

This DSP statement:

```
@DSP,V1,V2,V3 .
```

is requesting that the following be displayed:

- Mode V1 (that is, the mode number in V1)
- Type V2 (that is, the form type in V2)
- Report V3 (that is, the report number in V3)

Variables V1, V2, and V3 could be initialized in any number of ways. Here are some possibilities:

- You could load the variables earlier in the run so that if the data you want to display is moved to another mode or report, you 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 statement in the run.
- You could write the run to pick up information entered on the screen by the run user. For example:

```
ENTER THE LOCATION OF THE REPORT YOU WANT & TRANSMIT
    MODE □    , ALPHABETIC TYPE □ ,  RID □    ,
@BRK OUT,-0,3,3,1,1,Y,,,P .
@CHG INPUT$ V113,V2A1,V314 DSP,V1,V2,V3 .
```

  In this example, V1 is initialized as the mode entered, V2 as the type entered, and V3 as the report entered on the screen. The MAPPER system displays the requested report on the screen.

- Certain statements load variables automatically with pertinent information. For example, this statement executes a find across all reports in mode 2, type D, and loads V3 with the report number of the first find:

```
@LDV V1I3=0,V2A1=D .
@FND,V1,V2 '' 22-3 □,'  1' V3I5 .
```

- You could also load 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 0,TYPE B)
                    STATUS CODE□  ,   RID□   ,
@BRK OUT,-0,3,3,1,1,Y,,,P .
@CHG INPUT$ V4A2,V3I4 LDV V1I1=0,V2A1=B .
@FND,V1,V2,V3 '' 2-2 □,V4 ,V5I5 .
@DSP,V1,V2,V3,V5 .
```

In the last statement, V5, which is the line number of the find, is the line the display is started on.

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

```
@LDV V1|3=0,V2A1=D,V3|4=1,V4|5=6 .
@LZR,V1,V2,V3 V5|5 . V5=NR.LINES
@FND,V1,V2,V3,V4,196 '' 22-3 □,'  2' ,V4 .
@DSP,V1,V2,V3,V4 . V4=LINE
@INC V4 IF V4 NOT > V5 GTO LIN -2 .
@196: .
     No More Finds
@GTO END .
```

In this example, V4 is the counter for the line to start the find on, 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 reports and display the correct report at the appropriate line by using more variables and checks.

Loading Variables with Screen Input and Initial Input Parameters

You can load variables with the contents of these reserved words to capture initial input (run call) parameters or information the user has entered on the screen:

INPUT$	Captures user input from the screen or initial input parameters (18 characters maximum for input fields)
INSTR$	Captures user input from entire lines on the screen
INVAR$	Captures user input from fields on the screen
INVR1$	Captures user input from fields on the screen (terminated by the size of the variable rather than the size of the field)
ICVAR$	Captures input the user has entered at the control line (used only with the CHD [Command Handler] run statement)
FKEY$	Captures the number of the function key the user pressed

Here are some guidelines for using these reserved words:

- To resume the run after the output is displayed, press **XMIT** with the cursor below the control line. You can also press **F1** or enter rsm to resume the run, but any data you entered on the screen is not captured.
- If you specify Y in the *interim?* subfield of an OUT (Output) statement or use the I option of the SC (Screen Control) statement, the run continues automatically and you cannot load variables with these reserved words.
- If you specify Y in the *fxmit?* subfield of an OUT statement or use the X option of the SC statement, the run continues automatically also, but you can load variables with these reserved words.

NOTE: All input captured from the screen using these reserved words is in the character set of the run control report.

USING INPUT$ TO CAPTURE DATA FROM THE SCREEN

When using INPUT$ to capture data from the screen, remember these guidelines:

- String variables are not allowed.
- Specify @CHG INPUT$ *after* an OUT or SC 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, and so on.
- The number of characters loaded from a field depends on the defined length of the variable.

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

```
1. @BRK,0,A .
2. ENTER APPROPRIATE DATA AND TRANSMIT .
3. □       ,ENTER START DATE IN FORMAT YYMMDD
4. □       ,ENTER END DATE IN FORMAT YYMMDD
5. PLACE CURSOR HERE ->□ , AND TRANSMIT .
6. @BRK OUT,-0,2,6,1,1,Y .
7. @CHG INPUT$ <STDATE>16,<ENDATE>16 .
```

Here are descriptions of the lines in this example:

1. The first BRK statement defines the next output area (that is, the lines that follow) as mode 0, type A. (See "Handling Reports and Results" in Section 6 for an explanation of the output area.)

2-5. Place these lines in the output area.

6. The second BRK statement places the preceding lines from the output area into the -0 result. The OUT statement displays the new -0 result on the screen.

7. CHG INPUT$ loads <STDATE> with the start date and <ENDATE> with the end date that is entered.

USING INPUT$ TO CAPTURE INITIAL INPUT PARAMETERS

Here are some examples of initial input parameters:

- Information the user enters after the run name (for example, **runname,ab,1234,99.99)**
- RUN (Run Start) statement (See RUN in Section 7.)
- RRN (Remote Run) statement (See RRN in Section 7.)
- LNK (Link to Another Run) statement (See LNK in Section 7.)
- BR (Background Run) statement (See BR in Section 7.)
- BPRUN$ command in a batch runstream

When using INPUT$ to capture initial input parameters, remember these guidelines:

- You can capture up to 40 variables.
- Check the maximum size of the variable (see Table 4-1).
- String variables are not allowed.

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

```
1.  @CHG INPUT$ V1H2,V2I4,V3F5.2 .
      .
      .  (other run statements)
      .
2.  @BRK,0,A .
3.  ENTER PART NUMBER □   ,
4.  ENTER QUANTITY □      ,
5.  @BRK OUT,-0,2,2,1,1,Y .
6.  @CHG INPUT$ V1H14,V2I8 .
```

Here are descriptions of the lines in this example:

1. The first statement in the run captures three variables from initial input parameters.

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 the output area.

4. Place this line in the output area.

5. The second BRK statement places the output area in a result (-0); the OUT statement displays the new -0 result on the screen.

6. CHG INPUT$ loads the value of the part number into V1 and the value of the quantity into V2.

USING INSTR$ TO CAPTURE DATA FROM THE SCREEN

When using INSTR$ to capture data from the screen, remember these guidelines:

- ☐ You can use string variables.
- ☐ No leading tabs are required.
- ☐ Input variables are terminated by the end of the line.
- ☐ The maximum usable number of variables is the vertical screen size.
- ☐ Specify @CHG INSTR$ *before* the OUT or SC statement.
- ☐ Do not use INSTR$ with formatted or protected output screens, where the data received depends on the terminal type.

This example loads variables with two lines of data on the screen:

```
@CHG INSTR$ V1S80,V2S80 .
```

V1 contains up to 80 characters, starting with the first character following the SOE. If there is no SOE on the screen, it contains the first 80 characters starting with the home position. V2 contains up to 80 characters starting with the first character on the next line.

USING INVAR$ TO CAPTURE DATA FROM THE SCREEN

When using INVAR$ to capture data from the screen, remember these guidelines:

- ☐ You can enter up to 40 variables.
- ☐ You can use string variables.
- ☐ Specify @CHG INVAR$ *before* the OUT or SC 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, and so on.

- The length of an input field varies; it depends on the variable's defined length.

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

```
@CHG INVAR$ V1S17,V2I3,V3I6 .
@BRK .
ENTER DESCRIPTION □                    ,
ENTER QUANTITY □    ,
ENTER DATE IN FORMAT YYMMDD □        ,
@BRK OUT,-0,2,3,1,1,Y .
```

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

USING INVR1$ TO CAPTURE DATA FROM THE SCREEN

INVR1$ is similar to INVAR$, but it allows you to load several fields into one variable. When using INVR1$ to enter data from the screen, remember these guidelines:

- You can capture up to 40 variables.

- You can use string variables.

- Specify @CHG INVR1$ *before* the OUT or SC statement.

- Do not use INVR1$ with formatted or protected output screens, where the data received depends on the terminal type.

- The variable's length determines how many characters, including intervening tab characters, are loaded into each variable. Characters beyond the current screen line are not loaded.

In this example, after the user enters the solicited information, the run continues at the statement that follows the OUT statement. V1 contains both the first and last name the user entered. Note that protected format is not used.

```
@CHG INVR1$ V1S25 .

            First          Last

ENTER NAME □               □
@BRK OUT,-0,2,4,1,1,Y .
```

USING ICVAR$ TO CAPTURE DATA FROM THE CONTROL LINE

When using ICVAR$ to capture data from the control line, remember these guidelines:

- □ ICVAR$ captures information only when the user presses **XMIT** from the control line and you have specified a CHD statement.
- □ Specify @CHG ICVAR$ *before* the DSP, OUT, or SC statement.
- □ No leading tabs are required.
- □ You can use string variables.

This example displays a report and loads V1 with input the user transmitted from the control line:

```
@CHD 100 .
@CHG ICVAR$ V1S80 .
@DSP,0,B,2 .
```

USING FKEY$ TO CAPTURE FUNCTION KEY INPUT

In this example, the KEY statement requests function key input, the DSP statement displays RID 2B in mode 0, and the LDV statement loads V1 with the contents of FKEY$. V1 can then be tested for its contents, and the run can be processed accordingly.

```
@KEY .
@DSP,0,B,2 .
@LDV,W V112=FKEY$ .
```

VARIABLE Run – Testing Contents of Variables

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

To execute the VARIABLE run, enter:

```
variable
```

You get a message prompting you to enter a value. If you enter a value, you get a result. You can resume and continue testing values as long as you want. You can print the results you want to keep.

This example shows the Variable screen and a value to test:

```
Enter a value ▶ 100.00      (up to 6 characters)


          The VARIABLE run displays the contents of variables
            after initialization and arithmetic operations
```

This is the result:

```
line▶ 2     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶    ▶  <<<RESULT>>> ▶

Variable values for entry (100.00)

 @CHG INPUT$ V1      @CHG INVAR$ V2       CHG V3 V1 + 1          @INC V1
=================   =================   =================   =================
V1A6   = (100.00)   V2A6   = (100.00)   V3A6   = (   101)   V1A6   = (   101)
V1I6   = (   100)   V2I6   = (   100)   V3I6   = (   101)   V1I6   = (   101)
V1F6.3 = (100.00)   V2F6.3 = (100.00)   V3F6.3 = (101.00)   V1F6.3 = (101.00)
V1O6   = (100.00)   V2O6   = (   100)   V3O6   = (N/A   )   V1O6   = (100.00)
V1H6   = (100.00)   V2H6   = (100.00)   V3H6   = (100.01)   V1H6   = (   101)

                      .. Press F1 to enter another value
```

If you want to test another value, press **F1**.

BVT Run—Building Variable Tables and Converting Variables

Use the BVT (Build Variable Table) run to build or rebuild a table that displays the location of all the variables in your run control report. You can also use it to name variables so you can easily convert to or from named variables. (See "Variables—Names, Types, and Sizes" in this section for information about named variables.) The variable table result is displayed at the end of your run control report.

To use the BVT run, display your run control report and enter one of these requests:

BVT The Build Variable Table request builds a variable table and displays it as a result at the end of your run control report.

BVT,Q The Quick build lists only the variables that are defined (for example, V1A3 rather than V1).

CVT The Convert request converts V-type variables (such as V1) to named variables (such as <name>) using a previously built variable table from the end of your run control report.

CVT,N The Convert from Named Variables request converts all named variables to V-type variables.

CVZ This request converts all V-type variables to three-character variables (for example, V1 to V001).

Note that each of the previous requests also calls the BLT (Build Label Table) function, which builds or rebuilds the label table.

When you rebuild a variable table, the new table is matched with the existing table to preserve any user-defined names or comments.

Building or rebuilding a variable table produces this information as a result at the end of your run control report:

```
.VARIABLE TABLE
*    Name    .Vnum.Sq.            Line Numbers            .     Comment
*============.====.==.====================================.================
```

Field	Description
Name	The variable name (<name>) to equate to the V-type number. (Default = N000.)
Vnum	The V-type number to equate to the variable name (<name>).
Sq	The sequence number used to match and save user comments.
Line Numbers	The line number where the variables are located. The line numbers containing defined variables are flagged with asterisks.
Comment	Any user-supplied comments.

Here are some additional points to remember:

- When you use BVT or BVT,Q to build or rebuild a variable table locating a named variable, the run first checks for an existing table to determine a V-type variable to associate the named variable with. The run does not read USE run statements to determine a V-type variable (see USE in Section 7). For example, @USE NAME=V199 does not necessarily associate <NAME> with V199.

- When you use CVT or CVT,N to convert variables, you are not notified when a variable/variable is converted. For example, if you convert VV199 to <<name>>, it may not execute correctly.

- When you use CVT, CVT,N, or CVZ, and the size of the new variable causes the run statement to extend beyond the end of the line, the original report is displayed at that line number. No changes are made to the run control report until you change that run statement line.

Example

The run control report for the MARK Run, shown in Appendix C, is in RID 3E of mode 0. After the BVT run is executed against this run control report, this information is displayed at the end of the run control report:

```
.VARIABLE TABLE
*    NAME     .VNUM.SQ.          LINE  NUMBERS                    COMMENT
*============.====.==.=====================================.====================
 QTY          V001 01 7*,13,18*,19                          ORDER QUANTITY
 RETAIL       V002 01 7*,8,13,18*,19                        RETAIL $$$$
 CUST         V003 01 18*,19                                CUSTOMER NAME
                      ..... END REPORT .....
```

Reserved Words

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

You can initialize a variable with the value of a reserved word using CHG and LDV statements. For example, this CHG statement initializes V2 as an SOE character:

```
@CHG V2H1 SOE$ .
```

This LDV statement initializes <MODE> to contain the mode number, <TYPE> to contain the numeric form type number, and <RID> to contain the report number of the last report or result processed or on display:

```
@LDV,W <MODE>I4=MODE$,<TYPE>I6=TYPE$,<RID>I4=RID$ .
```

See also CHG and LDV in Section 7.

You can also use reserved words directly — where you might otherwise use a variable — in these subfields of a run statement:

m	Mode
t	Form type
r	RID number
l	Line number
f	Format
p	Parameters

NOTE: You cannot use reserved words in the output area, where MAPPER software reads them literally. You can use variables in the output area, however. (See BRK and OUT in Section 7 and "Handling Reports and Results" in Section 6 for more details.)

See Appendix B for a list of all reserved words and more examples.

5. Using Online Runs

This section contains information about online help and MAPPER runs you can use to help you write your own runs. For more information about the runs in this section, enter help,run,aid.

This section includes:

- HELP Run
- LIMITS Run — Displaying Report and Line Limits
- CC Run — Displaying Horizontal Column Count Positions
- FCC Run — Examining Report Fields
- FORM Run — Displaying Statement Fields and Subfields
- FORMC Run — Creating Statements for Functions That Use Function Masks
- MARS Run — Creating Statements in Run Control Report
- RUN Run — Automatically Generating and Registering Runs
- RUNA Run — Analyzing Your Run

HELP Run

The HELP run displays information on your screen about individual run statements and other items related to run design.

To execute the HELP run, enter:

```
help run
```

You get the RUN Information screen.

System Response

```
                              RUN Information

      ROLL▶ ▮                                (Tab to Selection or enter TARGET)

ADD       ▶  Append report
ADR       ▶  Add report
AID       ▶  Run design aids
ART       ▶  Arithmetic calculator
AUX       ▶  Auxiliary device control
BFN       ▶  Binary find in reports
BLT       ▶  Build label tables
```

This screen lists run targets. A ***target*** is an item on the screen that you can get information about. Notice that the cursor is located after **ROLL**. Just press **XMIT** to roll through the target list.

Here's how to get more information about a target from the list:

- ☐ Enter the name of the target after **ROLL**.
- ☐ Or tab to the target and press **XMIT**.

If you want to roll backwards through the list, enter .

If you know the target you want help with, you can enter the target on your help call:

```
help run,target
```

where *target* is either the actual function call or statement, or an abbreviation for other kinds of information.

For example, press **CURSOR TO HOME** and enter:

```
help run,add
```

This takes you directly to the online help for the ADD statement and tells you how to append one report to another.

USING HELP FOR RUN STATEMENT FORMATS

You can now use HELP to display a run statement format on the control line while you are writing a run. With a run control report on display, enter:

```
help @rfc
```

where *rfc* is the run function call you want the format for. You can use this format as a reference while supplying the fields for your own run statement. If you need more information about the run statement, press **F1** or enter **rsm** to display detailed HELP information. Press **F1** or enter **rsm** again to return to your run control report.

USING HELP WITH ERROR MESSAGES

If an error message appears in the control line while you're executing a run and you would like more detailed information about the error, press **CURSOR TO HOME** and enter:

```
help
```

After reading the information, press **F1** or enter **rsm**. The system returns you to the line where your error occurred.

LIMITS Run – Displaying Report and Line Limits

Use the LIMITS run to display the highest RID number and the maximum number of lines per report allowed for the current mode and type. This information is displayed on the first line of the report, overlaying the control line.

To execute the LIMITS run, display a report and enter:

```
limits
```

System Response

This example shows the system's response if you execute the LIMITS run with mode 0, RID 2B on display:

```
                Highest Report = 2000   Lines/Report = 131071
.DATE 15 FEB 88  08:27:51  RID      2B   22 JAN 86  JDOER
.@991231          CORPORATE PRODUCTION STATUS                          B000002
*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 831224 LS BLACKBOX1 436767        84389 AMCO 831223 831224
 IP 831225 LS BLACKBOX1 436768        84390 AMCO 831223 831225
 IP 831219 LS BLACKBOX2 637071        84353 INTR 831218 831219
```

CC Run—Displaying Horizontal Column Count Positions

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

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.

To execute the CC run, display a report and enter:

```
cc
```

System Response

This example shows the system's response if you execute the CC run with mode 0, RID 2B on display:

```
▮
         1111111111222222222233333333334444444444555555555566666666667777777778
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 831224 LS BLACKBOX1 436767         84389 AMCO 831223 831224
 IP 831225 LS BLACKBOX1 436768         84390 AMCO 831223 831225
 IP 831219 LS BLACKBOX2 637071         84353 INTR 831218 831219
```

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.

To redisplay the original report, press **F1**.

FCC Run – Examining Report Fields

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

To execute the FCC run, display a report and enter:

```
fcc
```

System Response

This example shows the system's response if you execute the FCC run with mode 0, RID 2B on display:

```
line▶ 1     fmt▶   rl▶ -     shft▶     hld chrs▶    hld ln▶    ▶ <<<RESULT>>> ▶
.DATE 15 FEB 88  08:18:52      REPORT GENERATION       JDOER
.MODE (0) TYPE (B) RID (2) CHARACTERS (80)
*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. If you want, print the result for future reference.

To redisplay the original report, press **F1**.

FORM Run – Displaying Statement Fields and Subfields

The FORM run displays the format of run statements (fields and subfields). You get the @ control character, the function call, and the abbreviated fields and subfields. It fills in all open function calls in the report.

To execute the FORM run, type the function calls you want the formats for in your run control report, and enter:

```
form
```

Example

Here's an example of a request to execute the FORM run with some function calls entered:

```
LINE▶ form  FMT▶  RL▶      SHFT▶    HLD CHRS▶   HLD LN▶   ▶     fcs     ▶
.DATE 15 FEB 88 08:22:20  RID     75    15 FEB 88
*RUN FUNCTION DATA: EXAMPLE OF USE OF THE RUN DESIGN AID 'FORM'          E0210
*=============================================================================.
@SRH
@SOR
@MCH
@DSP

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

System Response

```
LINE▶ 1     FMT▶   RL▶ -     SHFT▶     HLD CHRS▶    HLD LN▶    ▶     fcs      ▶
.DATE 15 FEB 88 08:24:20  RID     75E   15 FEB 88  JDOER
*RUN FUNCTION DATA: EXAMPLE OF USE OF THE RUN DESIGN AID 'FORM'          E0210
*=============================================================================.
@SRH,m,t(,r,l,q,lab)  o  cc  ltyp,p  (vlines,vls,vrid) .
@SOR,m,t,r  o  cc  ltyp,p .
@MCH,im,it,ir,rm,rt,rr(,lab)  o  icc  iltyp,ip  rcc  rltyp,rp .
@DSP,m,t,r(,l,tabp,f,int?,hold,msg80)  .


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

FORMC – Creating Statements for Functions that Use Function Masks

The FORMC run creates run statements for functions that use function masks, as well as for RDC, RDL, and WRL.

To execute the FORMC run:

- ☐ Enter the function call in a run control report including all necessary modes, form types, and report numbers (for example, **@MCH,0,B,1,0,C,1** .).
- ☐ Roll the line with the desired statement to the top of the screen.
- ☐ Enter formc in the control line.

The function mask or masks appear on your screen. Fill in the options and parameters and modify the mask, if necessary, just as you would for the equivalent manual function.

For RDC, RDL, and WRL, fill in the variables in the desired fields.

The system writes the statements in your run control report. Modify the statements as needed.

MARS – Creating Statements in Run Control Report

The MARS run creates MAPPER run 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.

The MARS run is especially useful for capturing functions you use repeatedly in a run control report. You can name the run and execute it at any time. Call your coordinator for more information.

The MARS run prompts you for the information it needs to write the run statements. It writes the statements in the run control report as it creates them.

To execute the MARS run, enter:

```
mars[,rt]
```

or

```
mars*[,rt]
```

where *rt* is the RID number and form type (if you already have a run control report), and * means use Directory field names instead of column-character positions in affected statements.

You get a menu of functions. You can do either of the following:

- ☐ Tab to the function you want and press **XMIT**.
- ☐ Enter the function call at the top of the menu after **Enter call**.

If you need help, enter:

```
mars,help
```

If you have the MARS run menu on your screen and you want help with any function, tab to the call you want help with and enter a ?.

RUN Run – Automatically Generating and Registering Runs

The RUN run automatically generates a run as you perform the manual functions. The manual functions appear to execute normally, but the RUN run converts them to run statements and accumulates them in a result.

When you finish typing the sequence of manual functions, you can register the run statements accumulated in the result as a separate run or you can append them to an existing run.

To execute the RUN run, enter:

```
run   or  run*
```

where * means use report field names instead of column-character positions in affected statements.

You can now begin entering manual functions in the sequence in which you want them saved as a run. Each function is converted into a run statement until you exit the RUN run.

CONTROLLING THE RUN GENERATION

You use these function keys to control execution of the run:

F1 Displays the generated run result.

F2 Displays the report currently being processed.

F3 Displays the Automatic Run Generation Logo.

F4 Terminates automatic run generation.

When you press **F4** to end the automatic run generation, this screen is displayed:

```
                              RUN

                  Automatic Run Generation Terminated

                      Display the result
                             Execute it
                    Register it in Type     (B-I)
                                   Exit

                       (You are in mode nnn)
```

Place the cursor in one of the fields and press **XMIT**, or if you want to register your run, enter the form type (B-1) in which to register it.

Field	Description
Display the result	Display the run as a result. If you want to return to this menu, press **F1**.
Execute it	Execute the run that has been generated. Note that you must press **F1** for each DSP in the result. After the run has finished executing, the menu is redisplayed.
Register it in Type	Place the generated result in the form type you specify and register it as a run. The run name is your user-id and only you can execute it. Because the system deletes any runs previously generated with the same name, you may want to ask your coordinator to change the name.*
Exit	Exit Automatic Run Generation.

* Note that the RUN run may be restricted on some systems; contact your coordinator if you have problems registering the run.

These functions generate run statements and terminate automatic run generation:

X	Generates an XIT statement. When the XIT is encountered in the run, the user is signed off.
REL	Generates a REL statement. When the REL is encountered in the run, the screen is released.

You can use most manual functions, except these:

A	Arithmetic
AL	Alarm
CALL	Interactive message switching
CUT (PASTE)	Cut (Paste)
LANG	Language
LZ	Line Zero
PC	Phrase Change
PL	Phrase Locate
PUNCH	Punch
PSW	Password
RPSW	Read Password
RSI	Remote Symbiont Interface (demand mode)
SP	Spelling Check
SS	Station-to-Station Message
WP	Word Process

You also cannot use:

OK	Acknowledge a message

or these auxiliary device directives:

SI	Activate a screen bypass to an offline terminal
SQ	Resend a report to an auxiliary device
SR	Reactivate printing at the specified page number
SX	Terminate printing on an auxiliary device

These functions work when you are entering them as manual functions, but do not generate run statements:

FUN	Function
L	Line Control
PNT	Paint
RSM	Resume
T	Type

DISPLAYING A REPORT OR RESULT

While you are executing manual functions within automatic run generation, the RUN run generates a DSP (Display Report) statement each time you display a report and when you terminate automatic run generation with a report or result on display. If you don't want the final DSP statement, press **F3** to get the Automatic Run Generation logo before pressing **F4**.

The RUN run does not generate a DSP statement for each function result. For example, if you execute the S (Search), SORT (Sort), and TOT (Totalize) functions, the generated run statements are entered into your run and only the final result is displayed. If you want your run to display each result, enter **d -** after each result is displayed.

You can also display a result at a specified line number and in any format. To display the result at a specific line, roll the line to the top of the screen. To display the result in a different format, enter the format in the control line. Remember to enter **d -** if you want your run to display this result.

LIMITATIONS

Here are some limitations of the RUN run:

- Entering L for line control is the same as pressing **F2** or entering pnt.

- You can use only the first three control line fields: **LINE**, **FMT**, and **RL**. The system ignores the others.

- You cannot reuse a function mask after an error message appears on the control line. Instead, you must reenter the function, get the function mask a second time, and execute the function again.

- To update displayed reports or results, you must use the CHG (Change) function or an updating function such as SU (Search Update) followed by an UPD, or use a line change function such as ADD LINE (]+*n*). However, when the generated run is executing, you can update in the usual way.

- The system does not save the report number produced by the AR (Add Report) or XR (Duplicate Report) function. The report becomes the current result (-0) for further processing.

  NOTE: Report numbers will probably be different at execution time from what they were at generation time.

- You cannot generate run statements that have no manual counterparts (such as IF and LDV). See "Functions and Statements" in Section 7 for a list of statements with no manual function counterpart.

- You cannot generate output screens or obtain user input from any other source (for example, by using the reserved words INPUT$ or INVAR$).

RUNA Run – Analyzing Your Run

The RUNA run identifies certain inefficient run design techniques. The RUNA run is not an absolute or total test of run design quality, but it does give you a good indication whether your run is acceptable. Carefully review any indications of inefficient techniques identified in the analysis and try to correct them. If you have a problem correcting something, call your coordinator. Your coordinator must give final approval of your run for production.

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

The RUNA run has its own built-in online HELP. To read it, enter:

```
runa
```

or

```
runa,help
```

(Note that if you enter `runa` without a log list on display, you get the RUNA run's online HELP.)

Before executing the RUNA run against your own run, do the following:

- ☐ Make sure you have a LOG statement (see LOG) at the beginning of the run (but after the label table, if there is one).
- ☐ Replace the REL statement (if used) with a GTO END statement to keep the log list intact after the run completes.
- ☐ Execute your run.

Follow these steps after your run completes:

1. Wait a few seconds; then press **F1**.

2. When the log list appears on your screen (almost immediately), enter runa to get the RUNA analysis result.

   At this point, you can print the result, or go to the next step.

3. If you want even more details, press **F1**.

   The RUNA run appends a detailed explanation of recommended corrective guidelines to the end of the result.

   - ☐ Print the result if you want to refer to it in the future.
   - ☐ If you don't want to print the result, you can press **F1** to return to the log list.

6. Designing and Debugging Runs

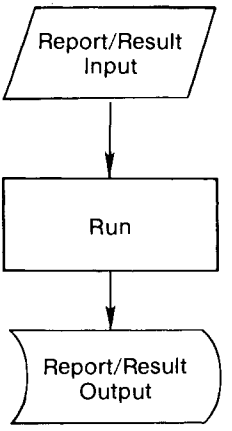
Before writing a run, it's important to know how to handle reports and results. In addition, you should follow a step-by-step procedure in creating your run. Finally, you need to know some debugging methods in order to create an error-free run.

This section includes:

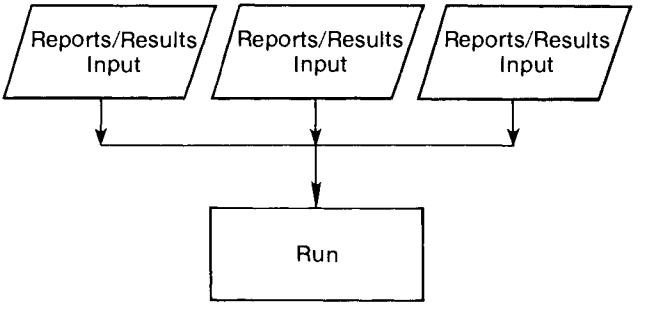
- ☐ Handling Reports and Results
- ☐ Designing and Registering a Run
- ☐ Debugging Your Run
- ☐ Using RDB (Run Debug)

Handling Reports and Results

You can process several reports in a single MAPPER run. The simplest run acts on a report or result from a previously executed function and produces a result or updated report, as shown in this figure:



You can use several reports and results from different modes for input, provided your coordinator has registered the modes for access. This figure shows a run using reports and results from different modes:



THE OUTPUT AREA

The ***output area*** is a temporary scratch area that you build in your run control report to hold information. This information is composed of ***output lines***, which are lines of data that do not have @ signs or colons in column one (and are not continuations of run statement lines). They may be, for example, messages or special screen displays you create that you want to display later in the run.

To examine the output area at any time during the run, enter a GTO END statement. This displays the contents of the output area as a result.

You can use output area data as a result at any time by executing a BRK statement (see BRK in Section 7). The BRK statement places the output area into the current result and clears the output area. You can then use the DSP or OUT statement to display the result. The output area is now empty, so you can place new data in it.

RESULTS

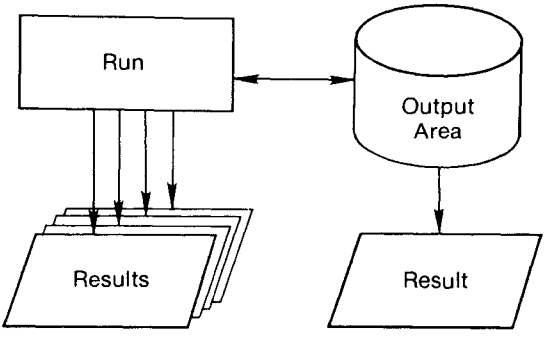
A ***result*** is a temporary copy of data obtained by executing a MAPPER function or run statement. The ***current result*** is the latest result, and its report number is always -0. Only one current result (result -0) exists at any moment. In addition to the current result, a MAPPER run can save up to four results for subsequent access. To save up to four results, rename them with an RNM statement (see RNM in Section 7).

To access results, specify the result identifier (the renamed result, such as -1, or the current result, -0) in the appropriate subfields of a MAPPER run statement. To access a report or result on display before the run started, refer to it as -0 until your run creates another result.

THE RELATIONSHIP BETWEEN OUTPUT AREA AND RESULTS

Do not confuse the output area with a result. You create a result using a function or run statement, such as SRH or TOT. You create the output area by adding output lines to your run control report. You can create an output area without affecting the current result or previously renamed results.

The following figure shows how runs use the output area and results:

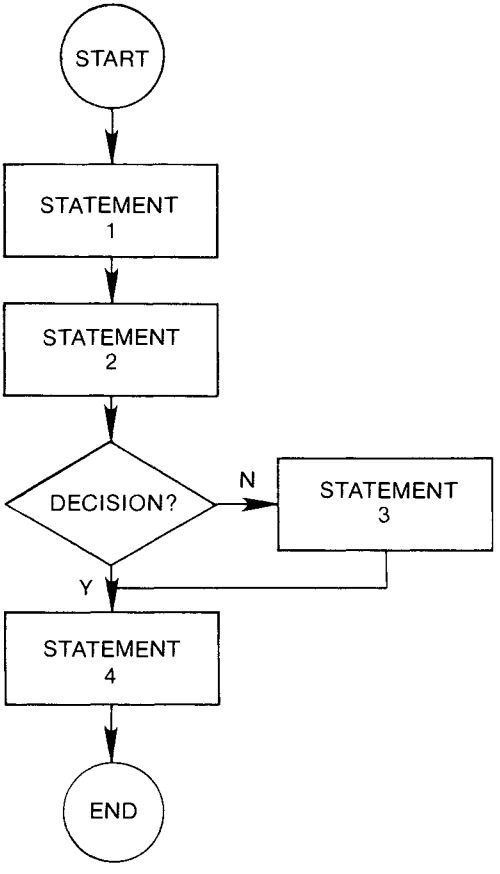


For more detailed information about the output area and results, see the Run Design Training Guide.

Designing and Registering a Run

When you're ready to write a MAPPER run, follow this logical step-by-step procedure:

1. Plan the run. Determine which statements you want to execute and whether you're going to use any logical decisions, paths, or loops. It may be helpful to draw a flow chart, as shown here, to map out the processing steps:



If you're unsure what effect a run statement has in a run, you can usually test the statement by running it separately. If it uses a manually executable processing function, test it manually first.

2. Register your run control report through your MAPPER system coordinator. First, execute the T (Type) function to see which form type is available in your mode for MAPPER runs. Next, use the AR (Add Report) function to add a report in that form type. Give your coordinator such information as report number, form type number, modes to be used, and your proposed run name. Explain the general plan of the run to your coordinator to assess its impact on the MAPPER system. If everything is acceptable, your coordinator registers the run for online debugging.

3. Enter the required run statements in the run control report. With manual updating, natural pauses between transmissions disperse the processing load; but in a run, where the statements are executed rapidly one after the other, virtually no pauses occur between the execution of statements. A run that executes several individual functions may put a severe load on the MAPPER system. You should consider loading effects of the run and adjust run user response expectations accordingly.

4. When the run is designed and ready for use, execute it in a production mode with logging (see LOG in Section 7) turned on. Have your coordinator assess your run's impact on the MAPPER system by examining the log list. Your coordinator may suggest improvements for your run.

   After accepting your run and obtaining a final log list, your coordinator registers it by its name. The coordinator may restrict user accessibility, mode accessibility, time of execution, input/output quantity, logic line count, and station numbers for your run.

   If you change your run significantly, your coordinator must reanalyze it.

5. You should give the run control report a report password and a save flag in this format:

   .@yymmdd

   The save flag must start in the first column of line 2 (the line below the date line) and must be a period line (notice the period beginning in column 1).

Debugging Your Run

There are several methods you can use to debug your run:

- ☐ Interactive debugging
- ☐ HELP run
- ☐ Checkpoint displays
- ☐ RDB statement and function
- ☐ RAR and RER run statements

INTERACTIVE DEBUGGING

When your run is halted because of an error, you see a one-line error message and the erring run statement line at the top of the screen. You can interactively correct the error instead of returning to the run control report. This technique works well for simple errors, such as an incorrect form type or variable name.

With the halted run on display, correct the erring line, move the cursor to the end of the line and press **XMIT**. The change is added to your run control report. To test the corrected run, execute it again.

THE HELP RUN

You can also use the HELP run to debug your run. When the run stops because of an error, an error is displayed on the control line. If you need more information, enter help to show a screen with a detailed explanation. Press **F1** or enter rsm when you are ready to continue. The screen shows the line where the error occurred.

CHECKPOINT DISPLAYS

If your run has several stages of processing, add DSP (Display Report) run statements to display intermediate results. You can check the results to see if the previous run statements have executed properly. If you write your run in modules, where each module performs a specific task, you can easily run each module with a checkpoint display to test it. As you debug the run, take the checkpoint displays out.

RDB STATEMENT OR FUNCTION

You can use the RDB (Run Debug) function or run statement to debug your run while it executes. This is different from interactive debugging because you can do the following:

- ☐ Display or change the value of any variable
- ☐ Display a window of the run control report or another report or result
- ☐ Examine a specific run statement line
- ☐ Stop the run when it comes to a specific variable

See "Using RDB (Run Debug)" in this section for more information.

RAR AND RER STATEMENTS

Additional bugs may be found while the run is being used in production. The RAR and RER run statements help track down the type of errors that occur so you can correct the run and produce a better version.

Sometimes a run stops because a user presses **MSG WAIT**, or a run statement error causes a problem. You can add the RAR statement so that your run jumps to a subroutine in case a user presses **MSG WAIT**. You can use the RER statement to jump to a subroutine if a run statement error occurs. See RAR and RER in Section 7 for more information.

Using RDB (Run Debug)

The RDB utility is a powerful run design tool. Use it to debug your run interactively, examining the contents of variables, reserved words, and renamed reports and results—all as the run is being executed. You can step through the run one line or command at a time, or you can set a breakpoint to halt the run at a specific line number, label, run command, or variable.

You can use RDB manually or in a run. In either case, you must be a registered run designer and the last person to update the run control report; otherwise, the RDB request is ignored.

Manual Function Format

To use the manual function, this is the format to enter on the control line:

RDB *run*[,*v*,...,*v*]

In field:	Enter:
run	the name of the run to debug.
v,...,*v*	any input variables supplied to the run.

Run Statement Format

@RDB .

If you use the manual function, the run halts before executing line 3 of the run control report and displays a screen. If you use the run statement, the run halts when it encounters the RDB statement and displays a screen. The following example illustrates the lines from the screen that is displayed:

```
RDB▶
RUN=testrun  MODE=0  RID=2E

     3▶@SRH,0,D,1 D 'Cust Code' *,AMCO .
```

Each line on the screen serves a specific purpose:

- The first line is the RDB prompt line; you enter RDB commands after the SOE (▶) on this line.
- The second line is the RDB Status Line. It displays the run name, mode number, report name, and other information.
- The third line is blank.
- The fourth line contains the line in the run control report to execute. It contains the line number and up to 70 characters of the line (up to 122 characters on a 132-character terminal). If you used the manual function, this line contains the first line (line 3) of the run control report. If you used the RDB run statement, this line contains the line in the run control report following the RDB statement.

The lines that follow the fourth line of the screen contain subsequent run statements as they are executed. Whenever a run halts, the next statement to execute is always the bottom line on the screen.

RDB COMMANDS

You execute RDB commands by entering information on the RDB prompt line or by pressing function keys. Table 6-1 describes the RDB commands.

Table 6-1. RDB Commands

Help	Enter a ? to display a summary of RDB commands. Press **F1** or enter rsm to return to the run.
Exit	Enter a ^ to terminate the run and release the screen.
Throttle	Use function keys **F1** through **F4** to advance the run:

F1 (Line Step) executes an entire line of logic, even if it contains more than one run statement. The run halts and displays the next line to execute.

F2 (Command Step) executes only a single run command. The run halts and displays the line starting at the column of the next run command to execute. (Note that a space-period-space terminator is considered a run command.)

F3 (Normal Speed) returns the run to normal execution speed. Pressing **F3** a second time halts the run.

F4 (Stream) causes continuous step execution of the run. Each line is displayed before it is executed (or, each command is displayed if the most recent step command was **F2**). Pressing **F4** a second time halts the run.

NOTE: The run always halts and displays the RDB prompt when resumed after a display or output command.

(continued)

Table 6-1. RDB Commands (cont.)

Display	Enter one of the following commands to display a variable, reserved word, the run control report, or a renamed report or result:
	V*n* displays variable number *n*. The variable size and type are displayed, followed by its contents enclosed in slant (/) characters. If your run uses named variables, you can display the variable by entering the name (for example, <name>).
	$*** displays reserved word ***$ (for example, USER$ or LLP$). The reserved word name is displayed, followed by its value enclosed in slant (/) characters.
	R displays the run control report at the line currently being executed. You can use any control line commands (such as ROLL, SHFT, and so forth) to manipulate the run control report. However, you cannot use manual functions such as S (Search) or LOC (Locate). Press **F1** or enter **rsm** to return to the run.
	-*n* displays renamed report or result number *n*. You can use any control line commands (such as ROLL, SHFT, and so on) to manipulate the renamed report or result, or you can update it with a line change or SOE Update. However, you cannot use manual functions such as S (Search) or LOC (Locate). Press **F1** or enter **rsm** to return to the run.

(continued)

Table 6-1. RDB Commands (cont.)

Breakpoint	You can set a breakpoint to cause the run to halt at a specified item. Only one breakpoint can be active at a time; setting a new breakpoint clears any previous one. When you set a breakpoint, it is displayed in the RDB Status Line. To set a breakpoint, enter one of these commands:
	B *n* — breakpoints at line number *n*. The run halts before executing line *n*.
	B L*n* — breakpoints at label number *n*. The run halts before executing the line at label *n*.
	B V*n* — breakpoints after variable number *n*. The run halts after executing any run statement that references variable *n*.
	B @*x* — breakpoints at run command *x* (for example, SRH, OUT, and so forth.). The run halts before executing run command *x*.
	B — clears the breakpoint.
Monitor	You can monitor a variable or reserved word by displaying its contents whenever the run halts (between line or command steps). Only one item can be monitored at a time; setting a new monitor clears the previous one. When you set a monitor, it is displayed on the RDB Status Line. To set a monitor, enter one of these commands:
	M V*n* — monitors variable number *n*. The size, type, and contents of variable *n* are displayed whenever the run halts.

(continued)

Table 6-1. RDB Commands (cont.)

M *$*	monitors reserved word *$* (for example, USER$, LLP$, and so forth). The contents of reserved word *$* are displayed whenever the run halts.
M	clears the monitor.

Execute — You can dynamically insert a temporary run statement to execute. Enter the statement in the RDB prompt line. For example:

```
RDB▶@LDV V1I4=1234 .
```

RDB ERROR MODE

If a run errs while under RDB control, press **F1** or enter rsm to activate RDB Error Mode. While in RDB Error Mode, you can examine variables, reserved words, and renamed reports and results as they existed when the run erred. You can execute only the RDB Display commands; the other commands (Breakpoint, Monitor, and Execute) are not allowed.

7. Run Statements

This section presents the MAPPER run statements in alphabetical order by the abbreviated call name.

The first subsection contains a reference list of all run statements in alphabetical order by the complete run statement name. The second subsection lists all run statements that have no manual counterpart. The rest of this section lists all MAPPER run statements; each subsection includes the run statement, its format, and brief examples.

Here's an overview of this section:

- List of Statements by Name
- Statements with No Corresponding Manual Function
- All Run Statements

List of Statements by Name

Name	Call
Accounting Log	LOG
Acknowledge Message	OK
Add Report	ADR
Append Report	ADD
Arithmetic	ART
Auxiliary	AUX
Background Run	BR
Batch Start	STR
Binary Find	BFN
Break	BRK
Break Graphics	BRG
Build Label Tables	BLT
Calculate	CAL
Calculate Update	CAU
Change	CHG
Clear Abort Routine	CAR
Clear Error Routine	CER
Clear Label Tables	CLT
Clear Link	CLK
Clear Subroutine	CSR
Clear Variables	CLV
Command Handler	CHD
Commit Updates	CMU
Conditional	IF
Copy	CPY
Create Result Copy	RSL
Date	DAT
Date Calculator	DC
DDP Copy	DCPY
DDP Create	DCRE
DDP Purge	DPUR
Decode Report	DCR
Decommit Updates	DCU
Decrement Variables	DEC
Defer Updates	DFU
Define	DEF

Define Variable Size	DVS
Delete	DEL
Delete Report	DLR
Device	DEV
Directory Information	DIR
Diskette	DIS
Display Graphics	DSG
Display Message	DSM
Display Report	DSP
Downline Load	DLL
Duplicate Report	DUP
Element	ELT
Element Delete	EL-
Encode Report	ECR
Exchange Variables	XCH
Execute Run Statement	XQT
Exit Subroutine	ESR
Extract	EXT
Find	FND
Find and Read	FDR
Form Type	TYP
Format	FMT
Function Key Input	KEY
Generate Organization Chart	GOC
Go To	GTO
Graphics Scalar	GS
Increment Variables	INC
Index	IND
Index User	IDU
Insert	INS
Justify Variables	JUV
Last Line Number	LLN
Line Add	LN+
Line Delete	LN-
Line Duplicate	LNX
Line Insert	LNI
Line Move	LNM
Line Zero	LZR
Link to Another Run	LNK
List Merge	LMG
Load Field Name	LFN

Load Format Characters	LFC
Load System Message	LSM
Load Variables	LDV
Locate	LOC
Locate and Change	LCH
Locate/Change Variable	LCV
Match	MCH
Match Update	MAU
Message to Console	MSG
Mode	MOD
Output	OUT
Output Mask	OUM
Peek Variables	PEK
Poke Variables	POK
Pop Variables	POP
Print	PRT
Push Variables	PSH
Read Continuous	RDC
Read Line	RDL
Read Line Next	RLN
Read Password	RPW
Reformat Report	RFM
Register Abort Routine	RAR
Register Error Routine	RER
Release Display	REL
Remote Run	RRN
Remote Symbiont Interface	RSI
Remove Variables	RMV
Rename	RNM
Replace Report	REP
Retrieve File	RET
Retrieve from History	REH
Return Remote	RTN
Run Debug	RDB
Run Start	RUN
Run Status	RS
Run Subroutine	RSR
Screen Control	SC
Search	SRH
Search Update	SRU
Send Report	SEN

Set Format Characters	SFC
Sign-Off	XIT
Sort	SOR
Station Information	STN
Subtotal	SUB
Tape Cassette	TCS
Totalize	TOT
Unlock	ULK
Update	UPD
Update Lock	LOK
Use Variable Name	USE
Wait	WAT
Word Change	WDC
Word Locate	WDL
Word Process	WPR
Write Line	WRL

Statements with No Corresponding Manual Function

Most MAPPER run statement calls have corresponding manual function counterparts. These statements, however, cannot be done manually:

BRG	Break Graphics
BRK	Break
CAR	Clear Abort Routine
CER	Clear Error Routine
CHD	Command Handler
CHG	Change
CLK	Clear Link
CLV	Clear Variables
CMU	Commit Updates
CSR	Clear Subroutine
DCPY	DDP Copy
DCRE	DDP Create
DCU	Decommit Updates
DEC	Decrement Variables
DEF	Define
DFU	Defer Updates
DIR	Directory Information
DPUR	DDP Purge
DSM	Display Message
DVS	Define Variable Size
ESR	Exit Subroutine
FDR	Find and Read
FMT	Format
GTO	Go To
IF	Conditional
INC	Increment Variables
INS	Insert
JUV	Justify Variables
KEY	Function Key Input
LCV	Locate/Change Variable
LDV	Load Variables
LFC	Load Format Characters
LFN	Load Field Name
LLN	Last Line Number

LNK	Link to Another Run
LOG	Accounting Log
LOK	Update Lock
LSM	Load System Message
MSG	Message to Console
OUM	Output Mask
OUT	Output
PEK	Peek Variables
POK	Poke Variables
POP	Pop Variables
PSH	Push Variables
RAR	Register Abort Routine
RDC	Read Continuous
RDL	Read Line
RER	Register Error Routine
RLN	Read Line Next
RMV	Remove Variables
RNM	Rename
RPW	Read Password
RRN	Remote Run
RSR	Run Subroutine
RTN	Return Remote
SC	Screen Control
SFC	Save Format Characters
STN	Station Information
TYP	Form Type
ULK	Unlock
USE	Use Variable Name
WAT	Wait
WRL	Write Line
XCH	Exchange Variables
XQT	Execute Run Statement

Add (Append Report)

The ADD statement appends the issuing report to the end of the receiving report and creates a result.

The ADD statement is equivalent to either the ADD ON or ADD TO manual function, depending on the order in which you specify the issuing and receiving reports.

If the receiving report has longer lines than the issuing report, the system fills the lines in the issuing report with spaces. If the receiving report has shorter lines, the system truncates the lines in the issuing report.

Format

@ADD,*im,it,ir,rm,rt,rr* .

In field:	Enter:
im,it,ir	the mode, type, and RID number of the issuing report.
rm,rt,rr	the mode, type, and RID number of the receiving report.

Example

This statement appends RID 2B to RID 1B, both in mode 0:

```
@ADD,0,B,2,0,B,1 .
```

Refer to the result as -0. For example:

```
@DSP,-0 .
```

The ADR statement adds a new report in a specified form type. You can specify the number of the report to be added. If you specify a report number, that report is added, provided that it does not already exist. If it does exist, the run continues at the label or relative line number specified; if you did not specify a label or line number, the run errs.

If you don't specify a report number, the MAPPER system selects the next available report number in the form type and puts the RID number in RID$.

The added report becomes the current -0.

NOTE: You can create a permanent report to use as a scratch area during run execution, but only if you intend to use it later as a place to permanently store the results of the run. It is inefficient to use an added report as a scratch area, then delete it at the end of a run. See also Appendix D.

Format

@ADR,*m,t*[,*r,lab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to be added.
lab	the label or relative line number to go to if the RID number specified already exists or the form type specified exceeds the maximum number of reports.

Reserved Word

Reserved word: RID$	
Word	Content
RID$	RID number of the report you just added

Example 1: Adding a New Report

This example uses an ADR statement to add a new report in mode 0, type B. The LDV statement captures the new RID number:

```
@ADR,0,B .
@LDV,PW <RID>13=RID$ .
```

Example 2: Adding a New Report to Specified RID Number

The following example adds RID 10B to mode 0. The run goes to label 99 if 10B already exists or if type B is full.

```
@ADR,0,B,10,99 .
```

The ART statement performs arithmetic operations on variables or constants. Variables capture the numbers resulting from these operations.

Use the ART statement primarily for solving complex operations, including arithmetic and trigonometric functions, which are discussed in this subsection. Use a CHG statement for simple computations.

Format

@ART exp vrslts .

In field:	Enter:
exp	an arithmetic expression or expressions.
vrslts	variables to capture the results of the expressions. (Initialize variables for the number of results you want to capture.) Use type A, type F, and type I variables only. (See Table 4-1 for a description of variable types.)

OPERATORS

Table 7-1 lists ***arithmetic operators*** that you can combine to form an expression.

Table 7-1. ART: Arithmetic Operators

Operator	Operation	Expression	Gives
+	Addition	a+b	value a plus value b.
-	Subtraction	a-b	value a minus value b.
/	Division	a/b	value a divided by value b.
//	Integer Division	a//b	value a integer divided by value b.
*	Multiplication	a*b	value a times value b.
**	Exponentiation	a**b	value of a raised to the power of value b.
-	Unary Minus	-a	negative the value of a.

NOTE: Values a and b are real integers and numbers and can include decimal fractions or expressions composed of such numbers.

FORMULATING ARITHMETIC EXPRESSIONS

When formulating expressions, specify arithmetic operators for every operation; for example, enter the operation *a* times *b* as a*b. Forms such as (a)(b) or ab are not valid.

Table 7-2 shows the priority by which the FORTRAN-based calculator of the MAPPER system performs arithmetic operations.

Table 7-2. ART: Priority of Arithmetic Operations

Priority	Operator	Operation
First	-	Unary minus
Second	**	Exponentiation
Third	*,/,//	Multiplication, division, integer division
Fourth	+,-	Addition, Subtraction

Example

This statement raises 3 to the 4th power, divides the result into 2, and places the answer in V1:

```
@ART 2/3**4 V1F6.2 .
```

Don't precede or follow operators with spaces.

MULTIPLE EXPRESSIONS

Evaluating multiple expressions in a single statement is more efficient than using a separate ART statement for each expression. This example adds V33 to V34 and puts the answer in V36I3. It then subtracts V34 from V33 and puts the answer in V37I3. Note that a semicolon (;) separates the expressions.

```
@ART V33+V34;V33-V34 V36I3,V37I3 .
```

INTERNAL COMPUTATION

You can refer to variables that are created internally by an ART statement (A, B, etc.), then use these variables for computing expressions within the same ART statement, as in this example:

```
@ART V1+V2;A*V3;B+5 ,V4I3,V5I3 .
```

In this example, the addition of V1 to V2 produces answer A, which is used in the second expression. The second expression, A*V3, produces answer B, which is used in the third expression. The first subfield in the variables field is skipped and a comma is used in its place, so the answer from the first expression (A) is not captured in a variable. Variable V4 contains the answer to the second expression, and V5 contains the answer to the third expression.

NEGATIVE NUMBERS

If it's possible that a variable used in an arithmetic expression has a negative number, place the variable in parentheses; otherwise, the calculator reads it as part of an expression and the run errs. Place all negative numbers in arithmetic expressions in parentheses. For example, in this statement, V5 contains a negative number:

```
@ART 3+(V5) V6I3 .
```

CHANGING THE HIERARCHY OF EXPRESSIONS

Use parentheses to change the hierarchy of expressions. In this example, 2 is divided by 3, the product is raised to the power of 4, and the answer is placed in V1:

```
@ART (2/3)**4 V1F6.2 .
```

ARITHMETIC AND TRIGONOMETRIC FUNCTIONS

You can perform the arithmetic and trigonometric functions shown in Table 7-3 using an ART statement. Note that *x* is a numeric value (whole or fraction) or an arithmetic expression.

Table 7-3. ART: Arithmetic and Trigonometric Functions

Function	Description
ABS(*x*)	absolute value or magnitude of *x*
ACOS(*x*)	arc cosine: angle in radians that has a cosine of *x*
ASIN(*x*)	arc sine: angle in radians that has a sine of *x*
ATAN(*x*)	arc tangent: angle in radians that has a tangent of *x*
CBRT(*x*)	cube root of *x*
COS(*x*)	cosine of *x* radians
CTN(*x*)	cotangent of *x* radians
DEG(*x*)	*x* radians expressed in degrees
EXP(*x*)	exponent: natural number "e" raised to power *x*
FRAC(*x*)	fractional portion of *x*
HCOS(*x*)	hyperbolic cosine of *x*
HSIN(*x*)	hyperbolic sine of *x*
HTAN(*x*)	hyperbolic tangent of *x*
INT(*x*)	integer portion of *x*
LOG(*x*)	logarithm of *x* in base "e"
LOG10(*x*)	logarithm of *x* in base 10
PI	pi (π)
RAD(*x*)	*x* degrees in radians
SIN(*x*)	sine of *x* radians
SQRT(*x*)	square root of *x*
TAN(*x*)	tangent of *x* radians

The calculator uses double-precision arithmetic. This produces a greater number of digits, resulting in greater accuracy.

AUX (Auxiliary)

The AUX statement sends reports to auxiliary devices — usually auxiliary printers — connected to display terminals. The AUX statement does not create a result (no -0 result exists).

Format

@AUX,m,t,r,sn,dev[,dlnos?,f,tabs?,dhdrs?,dlchar?,lsp,transp?,unit,sl,spcc] .

In field:	Enter:
m,t,r	the mode, type, and report number of the report to send.
sn	the station number where the auxiliary device is connected.
dev	the auxiliary device type reference: COP Communications Output Printer CQP Correspondence Quality Printer TC1 Cassette 1 TC2 Cassette 2 TD1 Diskette 1 TD2 Diskette 2
dlnos?	a Y to delete line numbers. Use Y if the report is 132 characters and you're printing basic format. (Default = N.)
f	the report format (1 to 6). (Default = basic.)
tabs?	a Y to include tab characters in the output or an N to change them to spaces. (Default = N.)

(continued)

(continued)

In field:	Enter:
dhdrs?	a Y to delete report headers in the output. (Default = N.)
dlchar?	a Y to delete the first character of each line in the output. (Default = N.)
lsp	1, 2, or 3 for the line spacing. (Default = 1.)
transp?	a Y to write lines exceeding 80 characters. (Default = N.) Applicable only for cassettes/diskettes.
unit	the unit on which to locate data in subsequent search operations. Applicable only for cassettes/diskettes.
sl	the station letter, which is on all station numbers. Use it only to access the station through the Communications Management System (CMS 1100). Call your coordinator if you do not know the station letter.
spcc	a ~ to delete special print control characters ~B, ~H, ~U, ~V, and ~X; also, if the printer is able, it can bold, underline, etc. (Default = blank; does not delete control characters before printing.) See the Word Processing Guide for more information on special print control characters.

Example

This statement prints RID 2B, mode 0, at station 123, on device name COP, with deleted line sequence numbers, deleted headers, and double spacing:

```
@AUX,0,B,2,123,COP,Y,,,Y,,2 .
```

BFN (Binary Find)

The BFN statement finds items very quickly in a sorted list. It does so by sampling the data at midpoint in the report or series of reports to determine whether or not the data to find precedes or follows this point (thus, the term "binary"). BFN ignores blank lines within the report. It continues sampling, dividing, and sampling until it finds the first occurrence of the data. With the N or O option, it creates a result.

NOTE: If a find is made, the report in which the find is made becomes the current -0. Any previous -0 result is destroyed.

The data in a report must be sorted on the fields the find is to be done on. If you want to perform a binary find on a range of reports, make sure the data is sorted across all reports.

If BFN detects a sort order discrepancy, doesn't find data, or doesn't find blanks when looking for them, it gives control to the label specified in the *lab* subfield.

If you specify a range of reports, make sure there are no empty reports (containing no valid data or headers only) within the range. If the BFN statement encounters an empty report, then finds the data that matches the search criteria in a subsequent report, it ignores any reports preceding the empty report. In this case, no error message is supplied because a valid find is made.

Format

@BFN,m,t[,r,l,lab] o cc ltyp,p [vrid,vlno] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to scan.
l	the line number where the binary find starts. If you have an idea where the data is located in your report, you can speed up the binary find process by designating a line number slightly before that location. If the data is actually located before the line number you specify, BFN searches the entire report and still finds the data. If you want to locate data after a specific line only (disregarding the previous lines of the report), use a FND statement instead.
lab	the label or relative line number to go to if no finds are made or in case of error.
o	options: A Process all line types. B Build an index containing the first data line of each report. Use an index to speed up the find operations across multiple reports. Enter a K in the *p* (parameter) field for the target field or fields. If you use more than one field, they must be in order from left to right. Be sure the targets do not cross reports. With the B option, BFN creates a result that you must replace into the report immediately preceding the first report of the specified range.

(continued)

(continued)

In field:	Enter:
	If you would like the RID number placed in a field, enter an equal sign in the corresponding parameter field.
C(*x*)	Alter normal character set processing: C(F) Full character set C(L) Limited character set C(S) Strict character set of report See also Appendix E.
E	Display last item only if item appears more than once.
I[*n*]	Index in type at report *n*. Enter only I for the default report (report 2); BFN scans report 2 or report *n* to determine where to find items, assuming that the reports having data to scan follow the index report.
K	Verify that reports are sorted in ascending order. Enter a K in the corresponding parameter field. Note that when you specify the K option, BFN tests each line of every report specified, so use it with discretion on large databases.

(continued)

(continued)

In field:	Enter:
N	Create a result containing a separate line for each item with the number of times the item appears. Enter a K in the corresponding parameter field to compare. Place an equal sign in the parameter field if you want to store the number of times the item appears in the database. To subtotal a field, enter =N in the parameter field. The =N subtotals only fields containing integer values. Note that you cannot use the = and =N parameters at the same time.
O	Create a result containing the items found, including their trailer lines. (Also, you can capture the information in four variables instead of just two. See the end of this table.) You cannot use the O option with the N option.
P	Include trailer lines (all line types other than the target line type) from the last valid find in the result. Valid only with the N option.
Q	Quick-find an item that appears only once in the report. Use the Q option if you know that the item appears only once in the report. When used with the O option, no trailer lines are included.

(continued)

(continued)

In field:	Enter:
	R*x-y* Scan a range of reports from report *x* through report *y*. Normally, when you process all reports in a type, report numbers 1 and 2 are skipped. You can use the R option to specify a range of reports, including reports 1 and 2. Note that you must specify a range; it cannot be a series of selected reports such as R3,5-10.
	S Scan each report separately.
	U Set an update lock on the report in which the find is made (or would have been made).
	@ Find blank lines or fields at the end of the report. Use this option in runs that write blank lines at the end of the report. When a blank field is found, it is the first blank field after the last line with data in it. If lines with blank fields are interspersed with lines containing data, the blank fields are not found. When using the @ option, enter a K in the corresponding parameter field.
	/ Find slant as data.
cc	the column-character positions or names of the fields in which to scan.
ltyp	the line type to scan. (If you specify the A option, leave this subfield blank.)

(continued)

(continued)

In field:	Enter:
	p one of these four kinds of parameters: the find parameters the K parameter if you are using the N, K, B, or @ option the = parameter if you are using the B or N option the =N parameter if you are using the N option **NOTE:** If no finds are made, and if the statement contains a label to go to, you can use *vrid* and *vlno*.
vrid	a variable to capture the RID number where the find would have been made.
vlno	a variable to capture the line number where the find would have been made. **NOTE:** With the O option, you can capture information in four variables (which replace *vrid* and *vlno*). (Note that if no finds are made and a label is specified, variables 3 and 4 are loaded with 0.)
variable 1	a variable to capture the number of finds.
variable 2	a variable to capture the number of lines in the result.
variable 3	a variable to capture the RID number of the first find.
variable 4	a variable to capture the line number of the first find.

Reserved Word

Reserved word: STAT1$ (error codes)	
Code	Error
1	Data not found
2	All lines with space fields filled (@ option)
3	Data not sorted

Example 1: Finding All Occurrences of an Item

```
@BFN,0,C,1 O 'PRODUCT-TYPE' □,BLACKBOX4 .
```

where:

0,C,1	Search in RID 1C in mode 0.
O	Use the O option to create a result.
'PRODUCT-TYPE'	Look in the PRODUCT TYPE field.
□	Process tab lines.
BLACKBOX4	Find BLACKBOX4s.

Example 2: Finding the Only Occurrence of an Item

The following example uses the Q option and captures the line number where the find was made in <LINE>:

```
@BFN,0,C,1,,99 Q 2-9 □,BLACKBOX4 ,<LINE>13 .
```

where:

0,C,1	Search in RID 1C in mode 0.
99	Go to label 99 in case of no finds or an error.
Q	Use the Q option to find it quickly.
2-9	Look in the PRODUCT TYPE field (column 2 for nine characters).
□	Process tab lines.
BLACKBOX4	Find the only BLACKBOX4.
<LINE>	Capture the line number where the find was made in <LINE>.

BLT (Build Label Tables)

Use a BLT statement to build label tables in a run control report and create a result.

At the start of a run, label tables indicate on which lines labels are located in the run control report. When the run reaches a statement that directs it to a label, the run already knows where the label is. This saves on overhead.

You can use the BLT statement to implement label tables in a database made up entirely of runs. Label tables increase the efficiency of runs with GTO statements that branch to labels (see GTO).

A new BLT statement in a report clears any old label tables before creating new ones.

NOTE: You will most often use the manual BLT function rather than the run statement. To use the manual function, simply display your run control report, enter **blt**, and replace the result. See the Manual Functions Reference for more details.

Format

@BLT,*m,t,r*[,*lab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the run control report to build label tables in.
lab	the label or relative line number to go to if an error condition exists (such as a duplicate or invalid label).

Example

```
@BLT,0,E,10,99 .
```

where:

0,E,10	Build a label table in RID 10E, mode 0.
99	Go to label 99 in case of error.

See also CLT.

BR (Background Run)

Use a BR statement to start a background run.

When you execute a BR statement, the system removes the current result (-0) from your run, if one exists, and passes it to the background run. Your original run continues executing.

Background runs cannot contain DSP, OUM, REL, RSI, or XIT statements; they can contain DSG, SC, or OUT statements only if the output is sent to another terminal. Also, a BR statement cannot be executed from a background run.

Format

@BR[,*sn*,*lab*] *run*[,*vld*] .

In field:	Enter:
sn	(optional) the station number to be notified when the background run completes. If omitted, the station is not notified.
lab	the label or relative line number to go to if the number of active background runs has already reached the maximum allowed at your site. If you omit the label, the run stalls until the background run is able to start.
run	the name of the background run to start. The run must be registered for execution as a background run by the coordinator.
vld	variables, literal data, reserved words, or any combination of these, to be picked up by the background run via INPUT$.

Reserved Word

Reserved word: ORSTAN$	
Word	Content
ORSTAN$	Station number from the BR statement that started the run, or originating station number if not supplied in the BR statement. ORSTAN$ always equals zero for nonbackground runs.

Example

This statement starts the background run KISMET, picking up input from USER$, and notifies station 123 when the run completes. If the number of active background runs has already reached the maximum allowed at this site, the run goes to label 99.

```
@BR,123,99 KISMET,USER$ .
```

BRG (Break Graphics)

Use a BRG statement to pack data in the output area and place it into a result. It is particularly useful for packing primitive graphics code and processing the result with MAPPER chart runs. The BRG statement packs leading and trailing spaces from a single line, but does not pack spaces embedded within the line.

The BRG statement is similar to a BRK statement (see BRK). When the run encounters a BRG statement, it places all data in the output area into a scratch result (-0) and clears the output area.

If you don't specify a mode and type for the output area, they are the same as those of the run control report. You can change the mode and type of the output area with a BRG statement without affecting the current result. You do, however, change the mode and type of the output area for subsequent results created by another BRG statement.

Format

@BRG[,*m*,*t*,*q*] .

In field:	Enter:
m,*t*	the mode number and form type of the output area. The *t* field is required if you specify a mode.
q	how many output lines (quantity). For output exceeding 500 lines, estimating improves efficiency by substantially reducing the I/Os. If you don't specify a mode and type, skip only one subfield (for example, @BRG,,2500).

Example 1: Packing Primitive Graphics Code

In this example, the first BRG statement places the current output area into a scratch result. (Note that a BRK could also be used here.) The primitive graphics code then becomes the current output area in the same mode and type as the run control report. The second BRG statement packs the primitive code and places it into a result that can be processed by MAPPER chart runs.

```
@BRG .

    Primitive Graphics Code

@BRG .
```

Example 2: Estimating Subsequent Output Lines

This statement places the output area into a result and estimates that the following output area will be 2,500 lines:

```
@BRG,,2500 .
```

Example 3: Specifying Next Output Area

This statement places the output area into a result. The next output area will reside in mode 2, type B.

```
@BRG,2,B .
```

BRK (Break)

Use a BRK statement to place the run's output area into a result. The run builds the output area automatically (see Section 6 for more details).

When the run encounters a BRK statement, it places all data in the output area into a scratch result (-0) and clears the output area.

If you don't specify the mode and type for the output area, they are the same as those of the run control report. You can change the mode and form type of the output area with a BRK statement without affecting the current result. You do, however, change the mode and form type of the output area for subsequent results created by another BRK statement.

Format

@BRK[,*m*,*t*,*q*] .

In field:	Enter:
m,*t*	the mode number and form type of the output area. The *t* field is required if you specify a mode.
q	how many output lines (quantity). For output exceeding 500 lines, estimating improves efficiency by substantially reducing the I/Os. If you don't specify a mode and type, skip only one subfield (for example, @BRK,,2500).

Example

This example uses multiple BRK statements:

1. `@BRK .`

   DATA1

2. `@BRK,0,B .`

   DATA2

3. `@BRK,,2500 .`

   DATA3

4. `@BRK .`

Here is an explanation of each BRK statement:

1. The output area that follows resides in the same mode and type as the run control report.

2. *DATA1* is now the -0 in the same mode and form type as the run control report; the output area that follows resides in mode 0, type B.

3. *DATA2* is now the -0 result in mode 0, type B.

4. *DATA3* is now the -0 result, at an estimated 2,500 lines, in mode 0, type B.

CAL (Calculate)

The CAL statement performs arithmetic computations and conditional evaluations on reports and results, and creates a result (except when the O option is used).

Format

@CAL,m,t,r[,l,q,lab] o cc ltyp,p eq [vrslts] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to process.
l	the line number at which to start processing.
q	how many lines (quantity) to process.
lab	the label or relative line number to go to if no data exists. Be sure to use this field if there is any possibility that some reports will not contain data; otherwise, no result is created.
o	options: A Process all line types. C Conditionally display specific result lines. After all equations are processed, include only those lines that meet a true condition based on the last IF statement in the result. For example, if the last IF statement is IF:A=0, include only those lines where field A is equal to zero in the result.

(continued)

(continued)

In field:	Enter:
E	Erase fields (fill with spaces) if the answer equals zero.
I	Produce integer results: Truncate any fractional part (that is, any numbers on the right side of the decimal point) in the result.
J(x)	Justify result value, where x is the justification:
	C Insert commas in the integer portion every three digits; eliminate nonsignificant zeros; place the resulting value in the leftmost portion of the field.
	L Left-justify, eliminate nonsignificant zeros, and place the resulting value in the leftmost portion of the field.
	R Right-justify, eliminate nonsignificant zeros, and place the resulting value in the rightmost portion of the field.
	X Expand: Eliminate nonsignificant zeros, place the resulting value in the leftmost column, and fill the remaining fields with zeros.
	Z Eliminate nonsignificant zeros, right-justify, and fill preceding fields with zeros.
Kn	Initialize a value label to n. Default value of value label = zero.
L	List all value label names and their final values at the end of the result.

(continued)

(continued)

In field:	Enter:
N*n*	Substitute the numeric value *n* for nonnumeric fields. The default value of nonnumeric fields is zero; a nonnumeric field has either no data (all spaces or tab characters) or data that has a nonnumeric character in its leftmost significant position.
O	Omit data lines. Include only header lines and all value label names and their final values.
R*n*	Round results to the nearest *n*: R.0000000000000001 through R100000 (nearest 100,000 units). **NOTE:** To control rounding equation by equation, use the R option as an equation option.
S(*x*)	Set character set interpretation to *x*. (The system compares limited character set strings to limited character set internal codes, and full character set strings to full character set codes; it interprets uppercase and lowercase alphabetic characters in the same way.) *x* may be: F Use full character set internal codes (use only when processing Limited Character Set [LCS] reports). L Use limited character set internal codes (use only when processing Full Character Set [FCS] or Full Character Set Upper [FCSU] reports).

(continued)

(continued)

In field:	Enter:
	S Use strict character set internal codes to differentiate between uppercase and lowercase alphabetic characters (use only when processing FCS reports).
T	Include both processed and unprocessed lines in the result. Do not use the T option if you want to include only the line type being processed.
V	Process only those equations whose result values are calculated from valid data (invalid data is either nonnumeric data or an invalid date). Note that a nonnumeric field has either no data (all spaces or tab characters) or data that has a nonnumeric character in its leftmost significant position. Skip equations with an invalid value for a result. Do not alter the receiving label.
X	Exclude invalid values in minimum, maximum, sum, and average computations (MIN, MAX, SUM, AVG, VMIN, VMAX, VSUM, VAVG) and in functions specified by vertical operators. (Invalid values include field labels that represent nonnumeric data and labels whose value was calculated from nonnumeric data or from an invalid date.)
*	Flag all invalid results with asterisks (*) after the value. Invalid results are values or labels calculated from nonnumeric report data, or values calculated from invalid dates. Note that a nonnumeric field has either no data (all spaces or tab characters) or data that has a nonnumeric character in its leftmost significant position.

(continued)

(continued)

In field:	Enter:
cc	the column-character positions or names of the fields in which to calculate.
ltyp	the line type to process. (If you specify the A option, leave this subfield blank.)
p	the parameters, which include: Alphabetic field labels Vertical operators (+, /,< , and >)
eq	the equations in this format: *receiving-label* [,*options*]=*expression*[; . . .] where: *receiving-label* may have: Alphabetic field labels One- to six-character value labels (beginning with alphabetic character, and containing both alphanumeric characters and characters $, %, !, and ?). Constant labels PI, LINE, LT, and RLINE

(continued)

(continued)

In field:	Enter:
	Note that ***receiving-label*** can be a partial label in the format: *partial-receiving-label=receiving-label(x-y)* where is the starting column and *y* is the number of characters. *options* Equation options that override statement options, if entered. Options E, I, J(*x*), K*n*, N*n*, R*n*, and * are valid (see statement options field *o*). Enter the option followed by a minus sign to override the statement option (for example, N-). If you use the J option, the minus sign must be enclosed in parentheses. *expression*[; . . .] Combination of one or more operands and zero or more operators regarded as a single numeric value.
vrslts	the variables to capture the final results of vertical operations and final values of value labels in the order initialized. (Initialize variables for the number of results and final values you want to capture.)

Reserved Words

Reserved words: STAT1$ and STAT2$	
Word	Content
STAT1$	Number of lines processed (i.e., number of lines of line type specified)
STAT2$	Total number of lines in report (excluding header lines)
Reserved word: TIC$	
If you use apostrophes in a CAL statement, use TIC$ if the run control report is in Limited Character Set (LCS). Use TIC$ or quotation marks (") for FCS and FCSU reports.	

Table 7-4 shows the priority by which the calculator performs arithmetic operations.

Table 7-4. CAL: Priority of Arithmetic Operations

Priority	Operator	Operation
First	-	Unary Minus
Second	**	Exponentiation
Third	*,/,//	Multiplication, Division, Integer Division
Fourth	+,-	Addition, Subtraction

Table 7-5 shows the priority by which the MAPPER system evaluates relational operations.

Table 7-5. CAL: Priority of Relational Operations

Priority	Operator	Relational Operation
First	=	Compare equal to
	<> or ><	Compare not equal to
	>	Compare greater than
	<= or =<	Compare not greater than (less than or equal to)
	<	Compare less than
	>= or =>	Compare not less than (greater than or equal to)
Second	&	AND (Boolean)
Third	,	OR (Boolean)

NOTE: The operators ampersand and comma (& and ,) don't perform a numeric comparison, but are based on a true/false concept.

Table 7-6 shows the result of all possible true/false conditions. *value1* and *value2* are usually relational expressions (such as a>1000).

Table 7-6. CAL: AND and OR True/False Conditions

AND Operation (value1) & (value2)			OR Operation (value1) , (value2)		
value1	value2	result	value1	value2	result
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Equations can include internal arithmetic and trigonometric functions, as shown in Table 7-7.

Table 7-7. CAL: Internal Arithmetic/Trigonometric Functions

Function	Description
ABS(x)	Absolute value or magnitude of x
ACOS(x)	Arc cosine: angle in radians that has a cosine of x
ASIN(x)	Arc sine: angle in radians that has a sine of x
ATAN(x)	Arc tangent: angle in radians that has a tangent of x
AVG($x1$,...,xn)	Average value of all subfields specified
CBRT(x)	Cube root of x
COS(x)	Cosine of x radians
CTN(x)	Cotangent of x radians

(continued)

(continued)

Function	Description
DEG(*x*)	*x* radians expressed in degrees
EXP(*x*)	Exponent: natural number "e" raised to power *x*
FRAC(*x*)	Fractional portion of *x*
HCOS(*x*)	Hyperbolic cosine of *x*
HSIN(*x*)	Hyperbolic sine of *x*
HTAN(*x*)	Hyperbolic tangent of *x*
INT(*x*)	Integer portion of *x*
LOG(*x*)	Logarithm of *x* in base "e"
LOG10(*x*)	Logarithm of *x* in base 10
MAX(*x1,...,xn*)	Maximum value of all subfields specified
MIN(*x1,...,xn*)	Minimum value of all subfields specified
MOD(*x,y*)	Modulus: remainder value of *x//y*
RAD(*x*)	*x* degrees in radians
RAN(*x*)	Random integer value in range *x* to *y*
SIN(*x*)	Sine of *x* radians
SQRT(*x*)	Square root of *x*
SUM(*x1,...,xn*	Sum: total value of all subfields specified
TAN(*x*)	Tangent of *x* radians
VAVG(*x1,...,xn*)	Vertical average of all subfields
VMAX(*x1,...,xn*)	Vertical maximum of all subfields
VMIN(*x1,...,xn*)	Vertical minimum of all subfields
VSUM(*x1,...,xn*)	Vertical sum of all subfields

Equations can also include DEF statements in the format:

DEF(*field-label*)

to produce a numeric value that defines the contents of a report field (see Table 7-8).

Table 7-8. CAL: DEF Statement Report Fields/Values

Contents of Report Field (field-label)*	Value
All tab characters or spaces or both	0
All numeric characters	1
All alphabetic characters	2
Alphabetic and numeric characters	3
All special characters	4
Special and numeric characters	5
Special and alphabetic characters	6
Special, numeric, and alphabetic characters	7

*The contents of a report field can include either just the characters indicated or both the characters indicated and spaces.

Equations can include these conditional statements:

IF:*expression*[; . . .]

and:

THEN:*equation*[; . . .]

and:

ELSE:*equation*[; . . .]

DATE AND TIME PROCESSING

You can use the CAL statement to perform computations on dates and times. CAL converts the date and time data to numeric values and processes them with equations.

The results of these computations represent numbers of days or hours. You can then convert these numbers to a specific format. See Table 7-9 for the available formats.

Using the DATE functions, you can process dates from January 1, 1944, through December 31, 2043.

Using the TIME functions, the CAL statement translates times into numbers of hours relative to midnight. You can also process minutes and times in computations; just convert them into hours by dividing minutes by 60 and seconds by 3,600. All times are based on a 24-hour clock.

There are two constant labels, which you can use directly in equations when you want to perform calculations or comparisons with the current date or time:

TODAY contains the current date expressed in the number of days relative to January 1, 1944.

TIME contains the current time expressed in the number of hours relative to midnight.

Input Functions

DATE INPUT functions translate dates in any of several formats into a number of days relative to January 1, 1944. You can use this number to compare dates or create a new date by adding or subtracting a number of days. Its format is:

D*n*(*x*)

where:

n is a number (0 through 8) identifying the date format.

x is a numeric date value or any type of label except a multiple field label other than the receiving label.

TIME INPUT functions translate times in any of several formats into a number of hours relative to midnight. You can use this number to compare other times or create a new time by adding or subtracting a number of hours. Its format is:

T*n*(*x*)

where:

n is a number (0 through 3) identifying the time format.

x is a numeric time value or any type of label except a multiple field label other than the receiving label.

Output Functions

The DATE OUTPUT functions translate the number of days relative to January 1, 1944, into one of several formats. You enter a DATE OUTPUT function as an equation option on an individual equation, such as:

receiving-label,D(*x*)=*expression*

where:

receiving-label	is a field label or value label.
(*x*)	is the output format the number of days is to be translated into. The format must be in parentheses.
expression	is a date calculation, usually the addition or subtraction of days from a date.

TIME OUTPUT functions translate the number of hours relative to midnight into one of several output formats. You enter a TIME OUTPUT function as an equation option on an individual equation, such as:

receiving-label,T(*x*)=*expression*

where:

receiving-label	is a field label or value label.
(*x*)	is the output format the number of hours is to be translated into. The format must be in parentheses.
expression	is a time calculation, usually the addition or subtraction of hours from a time.

When you specify a DATE or TIME OUTPUT function in an equation, the E, I, J, and R function and equation options are disabled since they also specify a type of output format.

If you specify a receiving label that is not large enough to contain the output format or if the result value of the expression isn't a valid time, CAL fills the field with asterisks (*) and assigns it a value of zero.

The system always right-justifies numeric formats and left-justifies alpha formats in the output field. It fills any unused columns in the output field with spaces.

DATE AND TIME FORMATS

Table 7-9 shows all available input and output formats for dates and times. The Min Size field specifies the minimum number of columns required to display that specific format.

Table 7-9. DATE and TIME Formats

Format	Input	Output	Min Size
DATE0$ (YMMDD)	D0(x)	D(0)	5
DATE1$ (YYMMDD)	D1(x)	D(1)	6
DATE2$ (DD MMM YY)	D2(x)	D(2)	9
DATE3$ (YDAY)	D3(x)	D(3)	4
DATE4$ (YYDAY)	D4(x)	D(4)	5
DATE5$ (DDMMYY)	D5(x)	D(5)	6
DATE6$ (MM/DD/YY)	D6(x)	D(6)	8
DATE7$ (Month DD, YYYY)	D7(x)	D(7)	18*
DATE8$ (MMDDYY)	D8(x)	D(8)	6
TIME0$ (HH:MM:SS)	T0(x)	T(0)	8
TIME1$ (HH:MM)	T1(x)	T(1)	5
TIME2$ (HHMMSS)	T2(x)	T(2)	6
TIME3$ (HHMM)	T3(x)	T(3)	4
More Output Time Formats			
Hour number		T(H)	2
Minute number		T(M)	2
Second number		T(S)	2
More Output Date Formats			
Month name		D(C)	1**
Day		D(D)	2
Julian day		D(J)	3
Month		D(M)	2
Day number		D(N)	1
Day name		D(W)	1**
Year		D(Y)	2

* DATE7$ fields must be exactly 18 characters; fewer displays all asterisks.

** The number of characters for output formats C and W depends on the field size of the receiving label.

In addition to the date and time equation options (D*n* and T*n*), you can use one of two W options in your equations:

W*n* specifies the number of days in a work week, from 1 to 6. (Default = 7.) A work week is considered to start on Monday.

W- overrides the W option.

The last example in this subsection shows date processing.

USING THE ICAL RUN TO CREATE A CAL STATEMENT

You can use the ICAL run to create a CAL statement. For information on how to use the ICAL run, see the Manual Functions Reference.

To create a CAL run statement equivalent of an equation set, use the ICAL run to process the equation set, press **F4**, and select the **Display Run Statement** option. This displays a freeform Full Character Set (FCS) type A result containing the CAL run statement. Note that you can't use the ICAL run to create a CAL run statement if your site is using a Limited Character Set (LCS) type for the freeform type A reports.

The Manual Functions Reference shows a detailed example using ICAL. If you try that example, pressing **F4** and selecting the **Display Run Statement** option, you receive the following statement:

```
@CAL,0,C,2 'R.01' 25-7,33-8,42-7,56-8 □,A,B,C,D \
C=B-A;D=C/B*100 .
```

CAL EXAMPLES

NOTE: All CAL examples except the last one process tab lines in RID 1C, mode 0.

Example 1: Multiplying Two Fields

```
@CAL,0,C,1 '' 'SPACE-REQ','DEMO-QUAN',\
'DEMO-RESULTS' □,A,B,C C=A*B .
```

where:

''	Use no options.
'SPACE-REQ' A	Label the SPACE REQ field: Field A.
'DEMO-QUAN' B	Label the DEMO QUANTITY field: Field B.
'DEMO-RESULTS' C	Label the DEMO RESULTS field: Field C.
C=A*B	Multiply the SPACE REQ field (A) by the DEMO QUANTITY field (B) and place the product in the DEMO RESULTS field (C).

Example 2: Moving Values into Fields Based on Values

The following example moves all values that exceed 24,000 from one field into another field:

```
@CAL,0,C,1 '' 'RETAIL $$$$','DEMO-RESULTS' \
□,A,B IF:A>24000;THEN:B=A .
```

where:

''	Use no options.
'RETAIL $$$$' A	Label the RETAIL $$$$ field: Field A.
'DEMO-RESULTS' B	Label the DEMO RESULTS field: Field B.

IF:A>24000; THEN:B=A	Put all RETAIL $$$$ field (A) values that exceed 24,000 into the DEMO RESULTS field (B).

Example 3: Averaging Items in a Field

```
@CAL,0,C,1 R.01 42-7,65-15 □,A,B B=VAVG(A) .
```

where:

R.01	Use the R option. Round result to nearest one hundredth.
42-7	Label the SALES COMMIS field (column 42 for 7 characters):
A	Field A.
65-15	Label the DEMO RESULTS field (column 65 for 15 characters):
B	Field B.
B=VAVG(A)	Put the cumulative vertical average of the SALES COMMIS field (A) into the DEMO RESULTS field (B).

Example 4: Using Apostrophes and Quotation Marks

The following example uses CAL in an LCS run control report. (Notice the apostrophes around the string BLACKBOX; these are necessary for the MAPPER system to distinguish between the string and the reserved word TIC$):

```
@CAL,0,C,1,,,99 C 2-8 □,A IF:A=TIC$'BLACKBOX'TIC$ .
```

Here is the same example using an FCS/FCSU run control report. (Note that if the string contained a space, it would need to be enclosed in apostrophes as well as quotation marks.)

```
@CAL,0,C,1,,,99 C 2-8 □,A IF:A="BLACKBOX" .
```

where:

99	Go to label 99 if no data exists.
C	Use the C option to include in the result only those lines that meet a true condition based on the last IF statement.
2-8	Label the first eight characters of the PRODUCT TYPE field (column 2 for 8 characters):
A	Field A.
IF:A=TIC$'BLACKBOX'TIC$ IF:A="BLACKBOX"	Include all lines with BLACKBOX in the first eight characters of the PRODUCT TYPE field.

Example 5: Calculating Vertical Totals and Maximum Values

```
@CAL,0,C,1,,,99 L 25-7,33-8,65-15 □,A,B+,C+ \
MAXA=VMAX(A);MAXB=VMAX(B);C=B-A V1I9,V2I9,\
V3I9,V4I9 .
```

where:

99	Go to label 99 if no data exists.
L	Use the L option to list all value label names and their final values at the end of the result.
25-7,33-8,65-15	Label the WHOLE SALE$ field (column 25 for 7 characters), the RETAIL $$$$ field (column 33 for 8 characters), and the DEMO RESULTS field (column 65 for 15 characters):
A,B+,C+	Fields A, B, and C, with the vertical operator + on fields B and C.
MAXA=VMAX(A);	Put the highest value found in the WHOLE SALE$ field (A) in the value label MAXA.
MAXB=VMAX(B);	Put the highest value found in the RETAIL $$$$ field (B) in the value label MAXB.
C=B-A	Subtract field A from field B and put the difference in field C.
V1I9,V2I9, V3I9,V4I9	Capture the vertical total of RETAIL $$$$ (field B) in V1, the vertical total of DEMO RESULTS (field C) in V2, the contents of the MAXA value label in V3, and the contents of the MAXB value label in V4.

Example 6: Processing Dates

Here's an example of processing dates using CAL. It adds a constant number of days to a date and specifies the same input and output date format:

```
@CAL,0,B,2 V 50-6 □,A A,D(1)=D1(A)+30 .
```

where:

0,B,2	Process RID 2B in mode 0.
V	Use the V option to process only valid data.
50-6	Label the PRODUC PLAN field (column 50 for six characters):
A	Field A.
A,D(1)= D1(A)+3	Add 30 days to the date in the PRODUC PLAN field (A); both the input and output dates are in DATE1$ format.

CAR (Clear Abort Routine)

Use a CAR statement to cancel an abort routine previously registered by an RAR statement (see RAR).

Format

@CAR .

The CAU statement performs arithmetic computations and conditional evaluations on reports and creates an updatable result.

You can do one of the following:

- Follow the CAU statement with a DEL statement to delete the found lines from the report (see DEL).
- Follow the CAU statement with an EXT statement to extract the found lines from the report and create a result (see EXT).
- Make changes to the updatable result and follow the CAU statement with a UPD statement to blend the updated lines from the updatable result back into the report (see UPD).

You cannot execute a CAU statement against a result.

See CAL; the same information applies to CAU.

Format

@CAU,m,t,r[,l,q,lab] o cc ltyp,p eq [vrslts] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to process.
l	the line number at which to start processing.
q	how many lines (quantity) to process.
lab	the label or relative line number to go to if no data exists.

(continued)

(continued)

In field:	Enter:
o	options (use CALCULATE options from the CAL statement).
cc	the column-character positions or names of the fields in which to calculate.
ltyp	the line type to process. (If you specify the A option, leave this subfield blank.)
p	the parameters, which include: Alphabetic field labels Vertical operators (+, /,< , and >)
eq	equations in the format: *receiving-label* [,*options*]=*expression*[; . . .] where: *receiving-label* may have: Alphabetic field labels One- to six-character value labels (beginning with an alphabetic character, and having both alphanumeric characters and characters $, %, !, and ?) Constant labels PI, LINE, and RLINE

(continued)

(continued)

In field:	Enter:
	options
	Equation options that override statement options, if entered. Options E, I, J(*x*), K*n*, N*n*, R*n*, and * are valid (see statement options field *o*). Enter the option followed by a minus sign to override statement option (for example, N-). If you use the J option, the minus sign must be enclosed in parentheses.
	***expression*[; . . .]**
	Combination of one or more operands and zero or more operators regarded as a single numeric value.
vrslts	the variables to capture the final results of vertical operations and final values of value labels in order initialized. Initialize variables for the number of results and final values you want to capture.

Reserved Words

Reserved words: STAT1$ and STAT2$	
Word	Content
STAT1$	Number of lines processed (i.e., number of lines of line type specified)
STAT2$	Total number of lines in report (excluding header lines)

Example

```
@CAU,0,D,2 C 'ORDER(1-2)' □,A IF:A=96 EXT .
```

where:

0,D,2	Process RID 2D in mode 0.
C	Use the C option to include in the result only those lines that meet a true condition based on the last IF statement.
'ORDER(1-2)'	Label the first two characters of the ORDER NUMBER field:
A	Field A.
□	Process tab lines.
IF:A=96 EXT	If the first two characters of ORDER NUMBER (A) equal 96, delete the lines from RID 2D and place the lines in the result.

CER (Clear Error Routine)

Use a CER statement to cancel an error routine previously registered by an RER statement (see RER).

Format

```
@CER .
```

CHD (Command Handler)

Use a CHD statement to register a routine to be executed whenever the user of the run enters information in the control line after a DSP, OUT, or SC statement.

The command handler routine can be in the same run control report (internal) or in another run control report (external). External command handler routines must be in the same character set type as the calling run control report.

NOTE: Don't use the CHD statement unless you're an advanced run writer. Its use is intended primarily for intercepting and interpreting commands that normally go to MAPPER software.

Without a CHD statement in the run, MAPPER software interprets all entries made in the control line. With a CHD statement in the run, however, the run itself interprets and processes input from the control line, except for the RELEASE function (a caret); MAPPER software continues to perform the RELEASE function unless you choose to give control to the run.

Whenever a run user transmits with the cursor on the control line, the run continues at the label for the report specified in the CHD statement and cancels any active subroutines. For example, if a subroutine contains a DSP, OUT, or SC statement, an ESR statement won't work (see ESR).

Format

@CHD[,*m,t,r,rel?*] *lab* .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report that contains an external command handler routine.
rel?	a Y to transfer release control (^) to the run. If you do not specify a mode, type, and report, skip only two subfields (for example, @CHD,,,Y 99 .).
lab	the label where the command handler routine starts (use 0 to cancel the currently registered command handler routine). You can also put a line number in the *lab* field. For external routines, use the absolute line number (LIN *n*, where *n* is 1 or greater); for internal routines, use the relative line number (LIN *n*| LIN -*n*).

Reserved Words

Reserved words: ICVAR$ and FKEY$	
ICVAR$ works only with a CHD statement; FKEY$ works with a CHD or KEY statement	
Word	Content
ICVAR$	Captures user input from the control line. Remember to put ICVAR$ before the variable in the CHG statement, and put the CHG statement before the DSP, OUT, or SC statement. If you specify Y in the forced transmit (fxmit?) subfield of an OUT statement, it has the same effect as pressing XMIT. If the cursor is on the control line, it also affects ICVAR$ input. You should use string variables (type S), but you can use type A or type H variables if the data fits in them. The system copies all input (except leading tab characters) into the specified variable up to the end of the variable. Other reserved words used to capture input (INMSV$, INPUT$, INSTR$, INVAR$, and INVR1$) are not affected if the user transmits with the cursor below the control line.
FKEY$	You can find out which key the user pressed with the reserved word FKEY$, whose value is always 0 if a CHD or KEY statement hasn't been used. Whenever the user presses a function key, the run continues executing at the statement following the DSP, OUT, or SC statement--not in the CHD routine--and FKEY$ contains a number indicating the key pressed: -1 MSG WAIT (not applicable with the KEY statement) 0 XMIT (with cursor below control line) 1 F1 or RSM 2 F2 or PNT 3-22 F3-F22

Examples

If the user presses XMIT from the control line, go to label 52 in this run control report:

```
@CHD 52 .
```

If the user presses XMIT from the control line or enters a release character, go to label 100:

```
@CHD,,,Y 100 .
```

If the user presses XMIT from the control line, go to the current line plus 3 in this run control report:

```
@CHD LIN +3 .
```

Cancel the currently registered command handler routine:

```
@CHD 0 .
```

Here is an example using ICVAR$, INVAR$, and FKEY$:

```
@CHD 10 .
@CHG ICVAR$ V1S80 .
@CHG INVAR$ V2A12,V3A8 .
@OUT,0,A,-0,2,1,,,Y .
@CHG V412 FKEY$ .
@IF V4 = -1,(20),0,(30),1,(40),2,(50),3,(60),4,(70) .
```

Processing continues at different points in the run, depending on which key the user presses:

- If the user presses XMIT from the control line, the run goes to label 10, with V1 capturing command input (entire control line).

  If the user presses MSG WAIT, the run goes to label 20.

- If the user presses XMIT with the cursor below the control line, the run goes to label 30, with V2 or V3 capturing data input.

- If the user presses **F1**, the run goes to label 40.
- If the user presses **F2**, the run goes to label 50.
- If the user presses **F3**, the run goes to label 60.
- If the user presses **F4**, the run goes to label 70.

Use a CHG statement to initialize or redefine the contents of variables (see Section 4 for more information about initializing and redefining variables). You can initialize or redefine all variable types except string variables with a CHG statement.

Also use a CHG statement for simple arithmetic computations. You can add, subtract, multiply, divide, and integer divide numbers or the contents of variables.

NOTE: To simply increase or decrease the numeric value of a variable, use the INC or DEC statement.

Format

@CHG *v vld* .

In field:	Enter:
v	the variable and, optionally, the variable type.
vld	variables, literal data, arithmetic expressions, reserved words, or any combination of these, to load the variable with.

Examples

This statement initializes the variable <CHAR> as an SOE character:

```
@CHG <CHAR>H1 SOE$ .
```

The following statement creates V3 from the computation. Note that the CHG statement evaluates the computations from left to right.

```
@CHG V3I6 V1 + V10 * V12 - 3 .
```

In this statement, V3 contains 3 (2.5 rounded to 3):

```
@CHG V3I6 5 / 2 .
```

In this statement, V3 contains 2 (the integer portion of the answer):

```
@CHG V3I6 5 // 2 .
```

In this statement, V3 contains 2.50:

```
@CHG V3F5.2 5 / 2 .
```

NOTE: Precede and follow each operator (+, -, *, /, //) with a space.

USING CHG STATEMENTS WITH RESERVED WORDS

Use a CHG statement to place the contents of reserved words in variables. Put the reserved word in the *vld* field, as in this example:

```
@CHG <RID>I3 RID$ .
```

Put reserved words that capture user input *before* the variable. These reserved words capture user input:

ICVAR$	(see CHD)
INMSV$	(see OUM)
INPUT$	(see OUT)
INSTR$	(see OUT)
INVAR$	(see OUT)
INVR1$	(see OUT)

Here's an example:

```
@CHG INPUT$ V1I3,V2A5 .
```

PROCESSING TYPE O VARIABLES

Type O variables have these additional operators:

A	logical AND
O	logical OR
X	exclusive OR
L	left shift
R	right shift
C	circular shift

NOTE: Right shift = +; left shift = -.

A shift of a type O variable is actually a bit-count shift. If the data being loaded is a literal value, MAPPER software assumes an octal value. If the data resides in another variable, MAPPER software assumes a decimal value and converts it to octal before processing.

In this example, V1 is changed to an octal variable with four characters, with an initial value of 2:

```
@CHG V1O4 2 .
```

The following statement performs a logical AND of the contents of V1 with 7:

```
@CHG V1 V1 A 7 .
```

The following statement performs a circular shift on the contents of V1 to the left one bit:

```
@CHG V1 V1 C -1 .
```

CLK (Clear Link)

Use a CLK statement in a run that was started by a LNK statement to clear the link to the original run (see LNK). Note that after you clear the link, you can link to another run.

Format

```
@CLK .
```

CLT (Clear Label Tables)

Use a CLT statement to delete label tables in a run control report and create a result.

A CLT statement is especially useful for deleting existing label tables in a run control report for easier run development and testing.

NOTE: You will most often use the manual CLT function rather than the run statement. To use the manual function, simply display your run control report, enter **clt**, and replace the result. See the Manual Functions Reference for more details.

Format

@CLT,*m,t,r*[,*lab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the run control report in which to clear label tables.
lab	the label or relative line number to go to if there is an error.

Example

```
@CLT,0,E,10,99 .
```

where:

0,E,10	Clear label table in RID 10E, mode 0.
99	Go to label 99 if no labels are found in the run control report.

See BLT for information on building label tables in a run control report.

Use a CLV statement to clear the definition and contents of a range of variables.

Typically, you use a CLV statement to release string variable space, where only a limited amount is available. After releasing the string space, you can use it for other variables.

Format

@CLV[,*stvno*,*q*] .

In field:	Enter:
stvno	the starting variable number to clear (*q* is assumed if *stvno* is not specified).
q	how many variables (quantity) to clear. All variables are cleared if this field is not specified.

Examples

This statement clears variables 1 through 10:

```
@CLV,1,10 .
```

This statement clears all variables:

```
@CLV .
```

See also PSH.

CMU (Commit Updates)

Use a CMU statement to make updates that were previously deferred in a report.

Use a CMU statement after a DFU statement if you want to commit the updates made since the DFU statement executed.

Format

@CMU .

The CPY statement copies an OS 1100 program file or element, or a data file, from one site to another through the remote run link. The file must be a sector-formatted file with no read or write keys. A remote run link (see RRN) must exist between the two sites.

If the receiving file does not exist, the Executive System (Exec) creates it with a maximum granularity of 6400 tracks.

Format

@CPY,o ls,lq,lfn[,le,lv] rms[,rmq,rmfn,rme,rmv] rmu,rmd,rmpw[,l] .

In field:	Enter:
o	options: A Absolute element O Omnibus element R Relocatable element S Symbolic element
ls	the local site number.
lq	the local qualifier.
lfn	the local file name.
le	the local element name.
lv	the local version.
rms	the remote site number.

(continued)

(continued)

In field:	Enter:
rmq	the remote qualifier.
rmfn	the remote file name.
rme	the remote element name.
rmv	the remote version.
rmu	the user-id registered at the remote site.
rmd	the user department number registered at the remote site.
rmpw	the user sign-on password at the remote site.
l	the line number on the screen where progress messages start.

Example

This statement copies qualifier MAPPER, file 2, absolute element THREE, version 1, to site number 2. Since the rest of the receiving subfields are blank, the copied element has the same identification — qualifier MAPPER, file 2, absolute element THREE, version 1.

JDOE is the user-id and 7 is the department number at the receiving site:

```
@CPY,A 1,MAPPER,2,THREE,1 2 JDOE,7 .
```

Use a CSR statement in an external subroutine to clear the return path to the calling run control report.

Once you use a CSR statement, you can no longer return to the calling run control report. You must use a CSR statement in the external subroutine run control report before calling another external subroutine. (See also RSR.)

NOTE: Use a CSR statement only in conjunction with an equivalent ESR statement. Use an ESR statement only in conjunction with an RSR statement. Call your coordinator for further details.

Register run control reports that have external subroutines with associated calling runs.

Format

```
@CSR .
```

See RSR for examples using subroutines.

DAT (Date)

The DAT statement performs computations on dates (from 1964 to 2063) within reports and results, and creates a result.

Format

@DAT,m,t,r o cc ltyp,p .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to perform computations in.
o	options:

A Process all line types.

T Convert the time in a field designated with a plus sign (+) to decimal hours, and place the converted time in the field with an equal sign (=). If you don't specify an equal sign, the result overlays the data in the time field.

NOTE: The only valid parameters are the plus and equal signs. The input data must be in the format *hh:mm:ss* or *hhmm*.

W Determine the day of the week.

n Specify the number of workdays in a week for computations, where *n* is the number of workdays.

(continued)

(continued)

In field:	Enter:
cc	the column-character positions or names of the fields where dates are located in the report.
ltyp	the line type to process. (If you specify the A option, leave this subfield blank.)
p	the parameters:

date format:

A	*ymmdd*
B	*yymmdd*
C	*dd mmm yy*
D	*yddd*
E	*yyddd*
F	*ddmmyy*
G	*mm/dd/yy*
I	*mmddyy*

arithmetic:

+	add
-	subtract
=	move result to this field
K	constant (integer)
:	place day of week here

The parameters subfield tells you at a glance which operations are performed in a DAT statement, how many date fields are used, their format, and which arithmetic functions are performed on each one. Always specify a result date field, such as B= or =.

If you don't specify a format in either the + or - field, format B is assumed in both fields.

If you specify a format for either the + or - field (but not both), the specified format is assumed for both fields.

If you don't specify a format for the = field, the format specified for the + field (if one exists) is assumed; otherwise, the format specified for the - field is assumed.

Example 1: Subtracting Dates

```
@DAT,0,B,2 '' 'PRODUC-ACTUAL','PRODUC-PLAN',\
'SHIP-DATE' □,-,+,= .
```

or:

```
@DAT,0,B,2 '' 57-6,50-6,64-6 □,-,+,= .
```

where:

0,B,2	Process RID 2B in mode 0.
''	Use no options.
'PRODUC-ACTUAL' 57-6	Subtract the date in the PRODUC ACTUAL field (column 57 for 6 characters)
'PRODUC-PLAN' 50-6 +	from the PRODUC PLAN field (column 50 for 6 characters)
'SHIP-DATE' 64-6 =	and put the difference in the SHIP DATE field (column 64 for 6 characters).
□	Process tab lines.

Example 2: Converting Dates to a Different Format

```
@DAT,0,B,2 '' 50-6,32-6 □,B,F= .
```

where:

0,B,2	Process RID 2B in mode 0.
”	Use no options.
50-6 B	Convert the format B date in the PRODUC PLAN field (column 50 for 6 characters)
32-6 F=	to a format F date, and place it in the PRODUC COST field (column 32 for 6 characters).
□	Process tab lines.

DC (Date Calculator)

The DC statement performs arithmetic computations on dates (from 1944 to 2043) or times that reside in variables, or on literal dates or times.

Format

@DC eq vrslts .

In field:	Enter:
eq	equations. Separate equations with a semicolon.
vrslts	variables to capture results of the equations in the order presented.

For each date you use in an equation, specify its format first. Enter dates in this format:

D*n* (*x*)

where:

n	is the number (0 through 8) identifying the date format.
x	is the variable containing the date, or the actual date.

Enter times in this format:

T*n* (*x*)

where:

n is the number (0 through 3) identifying the time format.

x is the variable containing the time, or the actual time.

NOTE: If you add 1 to the time, it appears in the hours position.

Use these formats:

D0(*x*)	**DATE0$**	***YMMDD***	
D1(*x*)	**DATE1$**	***YYMMDD***	
D2(*x*)	**DATE2$**	***DD MMM YY****	
D3(*x*)	**DATE3$**	***YDDD***	
D4(*x*)	**DATE4$**	***YYDDD***	
D5(*x*)	**DATE5$**	***DDMMYY***	
D6(*x*)	**DATE6$**	***MM/DD/YY***	
D7(*x*)	**DATE7$**	***MONTH DD, YYYY****	(format of constant label TODAY)
D8(*x*)	**DATE8$**	***MMDDYY***	
T0(*x*)	**TIME0$**	***HH:MM:SS***	(format of constant label TIME)
T1(*x*)	**TIME1$**	***HH:MM***	
T2(*x*)	**TIME2$**	***HHMMSS***	
T3(*x*)	**TIME3$**	***HHMM***	

* Since these formats have spaces, enclose actual dates in apostrophes. Don't enclose variables that have these formats in apostrophes.

Each equation in a DC statement creates a label (A through Z, consecutively) that you can use in subsequent equations in the same statement. You don't need to specify the format of a date represented by a label—it stays the same as the answer it represents and the system remembers it. You cannot name your own labels (as you can with an ART statement). However, two constant labels are available for calculations based on current date and time (TODAY and TIME).

Your answer appears in the same format as the date you use in the equation, unless you specifically change the format.

HOW TO CHANGE FORMATS

Change formats with these equations:

{D|T}{*n*|*w*}=*expression*

where:

D	is the date.
T	is the time.
n	is the format number.
w	changes the answer of an expression to the day-of-the-week name of the resulting date.

Examples

V1 equals today's date plus 90 days in DATE7$ format:

```
@DC TODAY+90 V1H18 .
```

V1 equals today's date in DATE1$ format; V2 equals current time in TIME0$ format:

```
@DC D1=TODAY;TIME V116,V218 .
```

V1 equals the day of the week that January 25, 1980, was; V2 equals 850125 plus five days, changed to DATE7$ format; V3 equals the number of days difference between the date in V2 (label B) and today's date:

```
@DC DW=D7('JANUARY 25, 1980');D7=D1(850125)+5;\
TODAY-B V1H10,V2H18,V3I4 .
```

V1 equals 831007 (V5) plus 60 days in DATE1$ format; V2 equals the day of the week that the date in V1 (A) is; V3 equals the date in V1 (A) in DATE7$ format; V4 equals the date in V3 (C) minus two days in DATE7$ format:

```
@LDV V5I6=831007 .
@DC D1(V5)+60;DW=A;D7=A;C-2 V1I6,V2H10,V3H18,V4H18 .
```

DCPY (DDP Copy)

Use a DCPY statement to copy an OS 1100 program file or element, or data file, from one host to another using DDP 1100.

The MAPPER system implements the DDP file transfer capability asynchronously. When the DCPY statement is executed, the MAPPER system transfers the copy request to DDP. DDP then does preliminary error checking and returns a status to the MAPPER system indicating that the copy is in progress. If the preliminary error checking does not pass, however, no error condition is returned. In either case, control is returned to the MAPPER system. Because the MAPPER system does not wait for the copy to complete, there are no reserved words that determine the status of the copy; you must determine the status outside of your MAPPER run.

Format

@DCPY[,lab] host,fn,rhost,fn,type[,pos,transl] .

In field:	Enter:
lab	the label or relative line number to go to if an error status is returned by DDP 1100.
host	the name of the host configured in DDP 1100 that contains the file to be copied from. Specify the host here as you specify it in the IPF environment.
fn	the name of the file contained on the host. If the file name contains special characters, enclose the field in single quotation marks.
rhost	the name of the host configured in DDP 1100 that is to receive the file.

(continued)

(continued)

In field:	Enter:
fn	the name of the file on the receiving host. If the file name contains special characters, enclose the field in single quotation marks.
type	the type of file or element to copy. If you need more than one type, separate them with commas and enclose the field in single quotation marks. These are valid types: DDP Copy the entire file ALL Copy all elements of the program file SYM Copy a symbolic element REL Copy a relocatable element ABS Copy an absolute element OMN Copy an omnibus element
pos	**ADD** to add to the end of a file, or **REP** to replace the existing file or element. (Note that you cannot use ADD if you are copying to an OS 1100 file.)
transl	a data translation entry. Files transferred between two OS 1100 sites must use transparent (TRA), which is also the default.

(continued)

(continued)

In field:	Enter:

These are the valid entries (you need to specify only the first three characters):

TRAnsparent	Do not translate the file or element.
ASCii	Translate the file or element to ASCII.
EBCdic	Translate the file or element to EBCDIC.

Please refer to the DDP-FJT Operations Guide, Vol. 1: IPF Interface for more information about each field of the DCPY run statement.

Reserved Words

The reserved words STAT1$, STAT2$, and STAT3$ contain error codes. These codes are listed in the DDP-PPC/DDP-FJT Messages Reference Manual.

Reserved words: STAT1$, STAT2$, and STAT3$ (error codes)

Word	Content
STAT1$	Interface error
STAT2$	CLASS-CODE
STAT3$	DETAIL-STATUS code

Example

This statement copies all symbolic elements from the file A*B. located on host RSL2 to the file D*E. on host TOC. The elements replace any existing elements on host TOC. If there was an error in the statement or DDP configuration, STAT1$, STAT2$, and STAT3$ contain the status code and the label 1 exit is taken.

When the DCPY is in progress, the file cannot be completely copied before control is returned. MAPPER software waits up to one minute for a status from the other host. If the status is not received, a COPY IN PROGRESS error is returned (STAT2$=1 and STAT3$=15). Depending upon the system usage, it may take some time for the copy to complete.

```
@DCPY,1 RSL2,'A*B.',TOC,'D*E.',SYM,REP .
```

DCR (Decode Report)

Use a DCR statement to decode a report that has been encoded with the ENCODE function or ECR run statement (see ECR). The decoded report becomes the current (-0) result.

Use caution when encoding and decoding reports. Here are some important things to remember:

- Don't forget your key. A report cannot be decoded if you lose or forget the key because no record of it is kept by the MAPPER system. Your coordinator cannot tell you what it is.

- Invalid characters and corresponding error messages are produced in these cases:

  - You specified the wrong key.
  - The encoded report has been corrupted.
  - The report is not encoded.

- If you are using a normal ASCII terminal, you may not be able to decode a report that was encoded from a National Character Set (NCS) terminal. In addition, you may not be able to decode a report that contains special NCS characters from a normal ASCII terminal.

- Because of high processing overhead, you may impact system performance if you use the DCR statement excessively.

Format

@DCR,m,t,r,key .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to decode.
key	the one- to eight-character key that was used to encode the report (A-Z and 0-9).

Examples

This statement decodes RID 10B of mode 0, which was encoded using the key JMT6077:

```
@DCR,0,B,10,JMT6077 .
```

DCRE (DDP Create)

Use a DCRE statement to create a file on a DDP 1100 host. Before a file is copied with DCPY, it must already be created on the remote host with DCRE.

Format

@DCRE[,*lab*] *host,fn*[,*dev,init,gran,maxfz,vol,access*] .

In field:	Enter:
lab	the label or relative line number to go to if an error status is returned by DDP 1100.
host	the name of the host configured in DDP 1100 where the file is to be created or allocated. Specify the host here as you specify it in the IPF environment.
fn	the name of the file to be created or allocated. If the file name contains special characters, enclose the field in single quotation marks.
dev	the name of the device where the file is to be created. (See the DDP-FJT Operations Guide, Vol. 1: IPF Interface for a list of valid device types.)
init	the initial file size in bytes. (Default = 0.) Note that on an 1100 host, this number multiplied by the granularity and divided by 7168 gives the number of tracks to allocate.
gran	the granularity of the file that specifies the size (in 9-bit bytes) of the units used in the *init* and *maxfz* subfields. (Default = 7168 bytes, which is one track.)

(continued)

(continued)

In field:	Enter:
maxfz	the maximum file size in bytes. (Default = system default size.)
vol	the volume-id for a removable or system disk pack. (Default = system fixed mass storage volume.)
access	the access type for the file (you need to specify only the first three characters): PUBlic This is a public (shared) file. PRIvate This is private (nonshared) file (default).

Please refer to the DDP-FJT Operations Guide, Vol. 1: IPF Interface for more information about each field of the DCRE run statement.

Reserved Words

The reserved words STAT1$, STAT2$, and STAT3$ contain error codes. These error codes are listed in the DDP-PPC/DDP-FJT Messages Reference Manual.

Reserved words: STAT1$, STAT2$, and STAT3$ (error codes)

Word	Content
STAT1$	Interface error
STAT2$	CLASS-CODE
STAT3$	DETAIL-STATUS code

Example

This statement creates the file A*B in host RSL2 on an 8450 disk with a movable head. The file is public and has an initial size of 10 tracks with a maximum of 500 tracks.

```
@DCRE,1 RSL2,'A*B.',F50M,10,,500,,PUB .
```

This Exec control statement creates the same file in a batch or demand program:

```
@CAT,P A*B.,F50M/10/TRK/500 .
```

DCU (Decommit Updates)

Use a DCU statement to decommit deferred updates and restore the reports to their state before a DFU statement (see DFU).

Format

```
@DCU .
```

DEC (Decrement Variables)

Use a DEC statement to decrease the numeric value of variables.

You cannot use a DEC statement with string (type S) variables. Also, if the variable you're trying to change contains alphabetic or special characters, the variable remains unchanged. A DEC statement is more efficient than a CHG statement, and it requires fewer characters.

Format

@DEC[,*n*] *v* [,*v*,...*v*] .

In field:	Enter:
n	the floating-point or integer number by which you want to decrease the value. (Default = 1.)
v	the variables you want to decrease by *n*.

Example

To subtract 1 from the numeric value of V3, use:

```
@DEC V3 .
```

instead of:

```
@CHG V3 V3 - 1 .
```

To subtract 5 from the numeric values in <COST> and <TOTAL>, use:

```
@DEC,5 <COST>,<TOTAL> .
```

Use a DEF statement to do the following:

- ☐ Test a variable, variable substring, or reserved word for its current contents or characteristics
- ☐ Set another variable to a value based on the option used

Format

@DEF[*,o,lab*] {*setv,testv* | *setv,testv*} .

In field:	Enter:
o	an option. If you use an option, use only one. These are the options: A Change the numeric form type in the test variable (which must be a valid, positive octal number and have a valid numeric form type number) to its equivalent alphabetic designation. The variable being set must be a type A or H variable. C Set the variable to contain the number of significant characters in the test variable. A significant character is any nonspace character or tab character.

(continued)

(continued)

In field:	Enter:

I Set the variable to contain a number based on the initialized type of test variable, as follows:

Test type:	Set to:
A	1
F	2
S	3
I	4
H	5
O	6

K If the test variable has Kanji characters, set the variable to 8. Otherwise, set it to 0 through 7 (see the *testv* and *setv* values at the end of the options in this table).

M Change the numeric form type in the test variable to its equivalent mode designation. A test variable must have a valid octal number and a valid numeric form type number. Set the variable to contain the mode number in which its numeric form type resides. For example, if the test variable's numeric type is 1 (form type A), set the variable to equal 0 (mode zero).

N Define the numeric form type of the mode number and alphabetic type in the test variable. The test variable must contain *modetype* (for example, 24C).

P Set the variable to contain the packed size of the test variable, as if it had been packed. (Packed size equals the number of nonspace characters and any intervening spaces.)

(continued)

(continued)

In field:	Enter:

S Set the variable to contain the size of the test variable.

T Set the variable to contain the number of tab characters in the test variable.

V Set the variable to contain the variable name that is assigned to the test variable number. (If *testv* does not have a variable name assigned, *setv* is loaded with spaces.)

If you don't specify an option, the DEF statement sets the *setv* variable to a value from 0 to 7, depending on the contents of the *testv* variable:

Contents of *testv*	Value of *setv*
All tab characters or spaces or both	0
All numeric characters	1
All alphabetic characters	2
Alphabetic and numeric characters	3
All special characters	4
Special and numeric characters	5
Special and alphabetic characters	6
Special, numeric, and alphabetic characters	7

Note that the contents of *testv* can include either just the characters indicated or both the characters and spaces.

lab the label or relative line number to go to if *testv* is not initialized.

setv the variable to set.

testv the variable or reserved word to test.

Examples

This example sets V1 to the code defining the kinds of characters in V2:

```
@DEF V1 V2 .
```

This statement sets <CODE> to the code defining the kinds of characters in <CHARS>:

```
@DEF <CODE>I1 <CHARS> .
```

This statement sets V5 to equal the alphabetic form type of the valid numeric form type in V6:

```
@DEF,A V5A1,V6 .
```

This statement sets <SIZE> to equal the number of characters in <NAME>:

```
@DEF,S <SIZE>I2 <NAME> .
```

If V1 was previously assigned to the variable name CAT, the following statement loads V199 with CAT:

```
@DEF,V V199H12 V1 .
```

The DEL statement deletes the lines appearing in the result of an update function from the report.

The following statements produce update results:

CAU
LCH (with the OU option)
LOC (with the OU option)
MAU
SRU

Format

@DEL .

Examples

In this example, the SRU statement searches tab lines for IP in column 2 for two characters. The DEL statement deletes all lines found in the search update from the report:

```
@SRU,0,B,2 D 2-2 □,IP .
@DEL .
```

To save a logic line, put both statements on one line, as in this example:

```
@SRU,0,B,2 D 2-2 □,IP DEL .
```

DEV (Device)

The DEV statement lists auxiliary devices and their status for a specified station. It creates a type A result if the device type you specify exists at the station number specified, and you do not specify a unit name.

Format

@DEV,*sn*[,*dev*,*unit*,*lab*] .

In field:	Enter:
sn	the station number.
dev	the device type: C printers D 5 1/4" diskette T tape cassettes/diskettes If you use the *dev* field, you must also use the *unit* field.
unit	the name of the unit in the terminal configuration report (such as COP, CQP, DS1, etc.). If you use the *unit* field, you must also use the *dev* field.
lab	the label or relative line number to go to if no device or unit exists at the station number specified.

Reserved Words

Reserved words: STAT1$, STAT2$, and STAT3$	
Word	Content
STAT1$	Number of devices specified in dev or unit subfield
STAT2$	Number of devices connected to the station
STAT3$	0 if requested station does not exist 1 if requested station exists

Examples

```
@DEV,123 .  LIST DEVICES AT STATION 123
```

The following statement lists type C devices, registered as COP, at station 123; the run goes to label 99 if no device (as specified) is found:

```
@DEV,123,C,COP,99 .
```

DFU (Defer Updates)

Use a DFU statement in conjunction with CMU and DCU to control multiple report updating. A DFU statement defers all updates to reports (that is, the updates are not made) until a CMU statement is executed.

If the run fails or aborts, the system restores all updates to the reports to their state before the DFU statement.

If the system fails during a run before the CMU statement executes, associated report updates are also decommitted.

You can control up to five reports with a DFU statement. Follow each DFU statement with a CMU or DCU statement before specifying another DFU statement.

If your run terminates for any reason, the system executes a DCU statement automatically and terminates the run with an error. Once a DFU statement executes, you must decommit or commit updates before you end a run normally.

You cannot specify just the mode and type in an attempt to lock an entire type.

These run statements cannot logically follow a DFU statement until a CMU or DCU statement executes:

DFU	set deferred update on a report
DSP	display a report
LOK	prevent other users from updating
OUM	mask output to screen
OUT	place lines on terminal screen
REL	stop run by releasing
RTN	return to a remote MAPPER run
RUN	start another MAPPER run
SC	create input screens or edit displayed data
WAT	stall run for a timed wait
XIT	sign user off from display terminal

Format

@DFU[,*lab*] *m,t,r*[,*m,t,r*,...*m,t,r*] .

In field:	Enter:
lab	the label or relative line number to go to if an update lock cannot be granted for any of the reports.
m,t,r	the modes, form types, and RID numbers of the reports to lock (up to five reports).

Example

This statement defers updating of reports 2B and 1C in mode 0; the run goes to label 99 if either of these reports is locked:

```
@DFU,99 0,B,2,0,C,1 .
```

DIR (Directory Information)

Use the DIR statement to load variables with information about a data name from the System Directory.

With the DIR statement, you can determine whether a data name is valid and the kind of data the name represents (mode, form type, or report).

Format

@DIR[,lab] name [vmode,vtype,vrid,vhiridr] .

In field:	Enter:
lab	the label or relative line number to go to if the name is invalid.
name	the data name to define. The data name can include nonsignificant characters, such as spaces or punctuation. The name need not be enclosed within apostrophes unless it contains embedded spaces.
vmode *	the variable to capture the mode number.
vtype *	the variable to capture the alphabetic form type.
vrid *	the variable to capture the RID number.
vhiridr *	the variable to capture the higher RID number if the data name defines a range of reports.

* A data name can represent a mode, a form type, a report, or a range of reports. Variables that do not apply for a particular data name are loaded with spaces.

Reserved Word

If the data name supplied is invalid, the run continues at the label. Examine STAT1$ for error codes. If there is no label, the run errs.

Reserved word: STAT1$	
Code	Error
1	Data name was not found in the System Directory.
2	Data name does not begin with an alphabetic character (A to Z).
3	Data name contains no alphanumeric characters (A to Z and 0 to 9).
4	Data name contains more than 16 alphanumeric characters.

Example

```
@DIR,99 ORDER-STATUS <MODE>I4,<TYPE>H1,<RID>I4,\
<HIRID>I4 .
```

where:

99	Go to label 99 if ORDER-STATUS does not exist.
ORDER-STATUS	Data name to obtain information about.
<MODE>I4	Load <MODE> with the mode number of ORDER-STATUS.
<TYPE>H1	Load <TYPE> with its form type.

<RID>I4	Load <RID> with its report number.
<HIRID>I4	Load <HIRID> with its higher range report number.

Use a DIS statement to write data to and read data from a 5-1/4-inch diskette drive connected to a Unisys UTS 30 display terminal.

The DIS statement is not supported on a UTS 60 display terminal. In addition, both the DIS statement and the corresponding manual function require level 3R4 microcode on UTS 30 display terminals.

NOTE: The DIS statement doesn't work on an IBM 3270 terminal.

Format

@DIS,m,t[,r,f,lab] code,dev,fn[,ext,tabs?,hdrs?,transp?] .

In field:	Enter:
m,t,r	the mode, type, and report to write to or read from.
f	the report format to read or write.
lab	the label or relative line number to go to if a diskette error occurs.
code	a **W** (for write), an **R** (for read), or a **D** (for delete).
dev	a three-character device name.
fn	a one- to eight-character file name.
ext	a one- to three-character file name extension.
tabs?	for writes: a **Y** to include tab characters. for reads: a **Y** to replace spaces with tabs from RID 0, or use predefined line type 0 to 9.

(continued)

(continued)

In field:	Enter:
hdrs?	for writes: a Y to include headers from the report. for reads: a Y to include headers from RID 0.
transp?	for writes: a Y to preserve lines longer than 80 characters. Enter Y for 132-character reports.

Reserved Word

Reserved word: STAT1$ (error codes)	
Code	Error
1	READY status was received when the diskette should be busy.
2	Diskette not ready or end of disk, directory, or file.
3	Diskette data error.
4	Diskette unable to proceed or no response.
5	Diskette device DLE 8.
6	Diskette timeout; THRU status wasn't received after BUSY.
7	Diskette file is empty or does not exist.
If you don't use a label, the run terminates and you get an error message (unless you have an RER statement in the run to register an error routine). If you use a label, STAT1$ indicates the type of error.	

Example

This statement writes RID 2B in mode 0 to a 5-1/4-inch diskette DS1 and gives it the file name FILE1:

```
@DIS,0,B,2 W,DS1,FILE1 .
```

DLL (Downline Load)

The DLL statement loads a precompiled program stored in a MAPPER report into a Unisys UTS 400 master or UTS 40 display terminal.

NOTE: The DLL statement doesn't work on an IBM 3270 terminal.

Format

@DLL,m,t,r,trfadr,dev[*,proglab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to downline load.
trfadr	the address to transfer control to after loading.
dev	the auxiliary device to load.
proglab	the program label reference to use when loading to cassette/diskette.

Example

This statement downline loads the program in RID 10A into the terminal's memory and transfers control to address A018 after loading:

```
@DLL,0,A,10,A018,MEM .
```

DLR (Delete Report)

The DLR statement deletes a MAPPER report, result, or renamed report. Normally, only the last person who updated a report can delete that report, but the DLR statement ignores this restriction, as well as read or write password restrictions.

Format

@DLR,*m,t,r*[,*lab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report or result to delete.
lab	the label or relative line number to go to if the report or result does not exist.

Examples

```
@DLR,0,B,3 . DELETE REPORT 3B IN MODE 0
```

The following statement deletes the current result; the run goes to label 99 if the result does not exist:

```
@DLR,-0,99 .
```

DPUR (DDP Purge)

Use a DPUR statement to delete a file on a DDP 1100 host.
Here are some guidelines for using the DPUR statement:

- ☐ You can remove all evidence of a file's existence with the DPUR statement.
- ☐ DDP provides protection to ensure that unauthorized deletion of files does not occur. Therefore, the name you specify must include any read/write keys.
- ☐ You can delete only one file with each DPUR statement.
- ☐ You can use the DPUR statement to delete only entire program or data files; you cannot delete elements of a file.

Format

@DPUR[,*lab*] *host,fn* .

In field:	Enter:
lab	the label or relative line number to go to if an error status is returned by DDP 1100.
host	the name of the host configured in DDP 1100 where the file is to be deleted or purged. Specify the host here as you specify it in the IPF environment.
fn	the name of the file to be deleted or purged. If the file name contains special characters, enclose the field in apostrophes.

Reserved Words

The reserved words STAT1$, STAT2$, and STAT3$ contain error codes. These error codes are listed in the DDP-PPC/DDP-FJT Messages Reference Manual.

Reserved words: STAT1$, STAT2$, and STAT3$ (error codes)	
Word	Content
STAT1$	Interface error
STAT2$	CLASS-CODE
STAT3$	DETAIL-STATUS code

Example

This statement deletes the file A*B. on host RSHS:

```
@DPUR,1 RSHS,'A*B' .
```

DSG (Display Graphics)

The DSG statement translates the primitive graphics code in a report and displays it on your terminal or another terminal. If you specify interim display, the run continues automatically; if you don't, you must press **F1** or enter **rsm** to continue the run.

NOTE: The DSG statement clears the output area after executing.

Format

@DSG,*m,t,r*[,*display,interim?,,sn,lab*] .

In field:	Enter:
m,t,r	the mode, form type, and RID number of the report containing the primitive graphics code.
display	an **A** to display alphanumeric text, a **G** to display graphics (default), or an **M** to display both text and graphics.
interim?	a **Y** for interim display (the run continues without the user resuming). (Default = N.)
sn	the station number where the DSG is to be displayed. If you leave this field blank or specify 0, the DSG is displayed at the station that is executing the run.
lab	the label or relative line number to go to if a DSG to another station cannot be successfully completed. If you leave this field blank, and the DSG statement cannot be successfully completed, the run is terminated with an error. See "DSG TO ANOTHER STATION" in this section for STAT1$ error codes.

DSG TO ANOTHER STATION

You use the *sn* and *lab* subfields to display graphics on terminals other than the one executing the run.

If you do not specify *lab* and the DSG statement cannot be successfully completed, the run is terminated with an error. Examine STAT1$ for the status code.

Reserved Word

Reserved word: STAT1$	
Code	Error
1	Station does not exist or it is a batch port, remote run, MTQ, or background station.
2	Station is not available because it is not currently connected to the MAPPER system.
3	Either no one is signed on at the specified station and interim was not specified, or the terminal is not a graphics terminal.
4	There is no answer. The user at the specified station did not respond to the message wait signal within one minute.

When you send a DSG to another terminal where a user is signed on, that user's message wait signal is activated and your run stalls. If the user responds to the message wait signal, the DSG information is displayed on the user's screen. If you specified N in the *interim?* subfield, your run stalls until the user at the receiving terminal presses any key; if you specified Y in the *interim?* subfield, your run continues automatically.

If the user at the other terminal does not respond within one minute, your run either continues at the specified label or terminates with an error (STAT1$=4). If subsequent DSGs are sent to that station, the user must respond to each message wait signal.

If you know that no user is signed on at the specified station, specify a Y in the *interim?* subfield. In this case, the message wait signal is not activated, the DSG information is displayed on the screen, and your run continues. If you do not specify an interim DSG, your run continues at the specified label or terminates with an error (STAT1$=3).

You cannot obtain exclusive use of any station. When more than one run is sending a DSG to the same station, the outputs may be intermixed and not displayed in the same order as they were sent. In addition, the one minute time limit may elapse for the second DSG before the first DSG is completed.

Examples

This statement displays text from the primitive graphics code in RID 1A, mode 0. The user must press F1 or enter rsm to continue:

```
@DSG,0,A,1 .
```

The following statement displays text and graphics from the primitive graphics code in RID 2B, mode 0. The run continues automatically:

```
@DSG,0,B,2,M,Y .
```

The following statement displays graphics from the primitive graphics code in RID 3C, mode 0, on station 123. The run continues automatically:

```
@DSG,0,C,3,,Y,,123 .
```

DSM (Display Message)

Use a DSM statement to display your own one-line message at the top of the screen in place of the control line. You can use DSM in one of two ways:

- Display a message at the top of an existing screen.
- Display a report or result, placing a message at the top of the screen. You can display the report or result beginning at line one or at a specified line number.

The message to display resides in a report or result. You display it by specifying its line number.

The DSM statement stalls the run until the user transmits or resumes, unless you specify the interim display, in which case the run continues automatically. Note that the DSM statement clears the output area.

Here are some characteristics of the DSM messages:

- The text appears as is does for MAPPER system error messages: reverse video for most monochrome terminals and white characters on a red background for color terminals.
- Less than (<) and greater than (>) signs are translated into blink characters.
- If you have a 132-character terminal, the messages are automatically centered.

Format

@DSM,m,t,r,lmsg[,tabp,erase?,interim?,pdq,dm,dt,dr,dspl,dspf] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report or result containing the message.
lmsg	the line number of the message in the report.
tabp	the tab position (maximum of 63) at which to place the cursor after the message is displayed. If you use a positive number (1-63), the cursor tabs forward from the HOME position; if you use a negative number, it tabs backwards from the HOME position.
erase?	a Y to erase the rest of the screen.
interim?	a Y to continue the run without the user transmitting or resuming.
pdq	the number of lines to push down on the screen.
dm	the mode number of the report to display.
dt	the form type of the report to display.
dr	the RID number of the report to display.
dspl	the line number of the specified report at which to start the display.
dspf	the format (1-6) of the report to display.

Examples

This statement displays a message at the top of the current screen. The message is located on line 4 of the current result.

```
@DSM,-0,4 .
```

The following statement displays RID 2D of mode 0, and places a message (located on line 4 of RID 12A in mode 0) on the top of the screen. The cursor is positioned at the 10th tab position.

```
@DSM,0,A,12,4,10,,,,0,D,2 .
```

DSP (Display Report)

The DSP statement displays a report or result on your display terminal and clears the output area. To continue the run after the display, press F1 or enter rsm.

You can use the DSP statement to display and manually update reports during run execution or to print the reports that are on display.

NOTE: Don't put other run statements on the same line after a DSP statement. The logic scan of the line terminates after executing the DSP statement.

Format

@DSP,m,t,r[,l,tabp,f,interim?,hold,msg80] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to display.
l	the line number in the report at which to start the display.
tabp	the tab character after which to position the cursor in the report. Keep in mind that if you are displaying a report with a control line, the default cursor position is in the RL field.
f	the report format.
interim?	a Y for interim display (the run continues without the user resuming). (Default = N.)

(continued)

(continued)

In field:	Enter:
hold	the number of lines already on the display screen to hold. The new report is displayed beginning at the first non-held screen line.
msg80	a message of up to 80 characters to display in the control line. Remember to place significant spaces within apostrophes.

Examples

```
@DSP,-0 . DISPLAY CURRENT RESULT

@DSP,-3 . DISPLAY RENAMED RESULT -3

@DSP,0,B,2 . DISPLAY REPORT 2B IN MODE 0
```

This statement displays format 5 of RID 1D, mode 0, starting at line 3:

```
@DSP,0,D,1,3,,5,,,' This line displayed on control \
line. '
```

This example displays a simulated control line with a message. It allows the user to select a line number or format or to roll the display without first requesting line control:

```
@DSP,-0,1,3,,,,'line'SOE$'□1      fmt'SOE$'□  rl'SOE$\
'□                <Press F1 or enter RSM to return>' .
```

DUP (Duplicate Report)

Use the DUP statement to create a new report by duplicating an existing report or result. The run loads the reserved word RID$ with the number of the new report.

Format

@DUP,*m,t,r*[,*rm,rt*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to duplicate.
rm,rt	the receiving mode and type to duplicate the report into.

Reserved Word

Reserved word: RID$	
Word	Content
RID$	RID number of the new report

Example

This example duplicates RID 1B in mode 20 into mode 0, type B. Variable <RID> captures the RID number of the new report.

```
@DUP,20,B,1,0,B .
@CHG <RID>|3 RID$ .
```

DVS (Define Variable Size)

Use the DVS statement to create variables equal to the size of report fields. For example, use it to capture input parameters to be processed against report fields, or to build screens whose input fields must be the same size as report fields.

Generally, you'll use a DVS statement with named fields, because a name does not directly specify the field size. The run defines the size of the variable when it executes. Any input parameter or screen using the affected variable dynamically adjusts to changes in field sizes.

In a DVS statement, you specify the variable number and its type; you don't specify the variable's size. The run assigns a size to the variable equal to its corresponding report field, then fills it with spaces.

Format

@DVS[,*m,t,r*] *field*[,....,*field*] *v*[,*v*,....,*v*] .

In field:	Enter:
m,t,r	the mode, form type, and RID number of the report or result where the fields reside. (Default = -0.)
field	the field whose size you want to define (can be field names or column-character positions).
v	the variables to define. Specify the variable number and type (for example V1H, V2S, and so forth).

Examples

This example initializes a variable to the size of the CUST CODE field in RID 2B, mode 0, for use as an input parameter:

```
@DVS,0,B,2 'CUST-CODE' V1H.
@CHG INPUT$ V1 .
```

The following example initializes variables to the size of the ORDER NUMBER and ORD QTY fields from the current -0, and creates a screen using their sizes:

```
@DVS 'ORDER-NUMBER','ORD-QTY' <NUM>H,<QTY>I .
@BRK .
      Enter Order Number: □<NUM>,
       Enter Quantity: □<QTY>,

      Transmit from here: □ ,
@BRK OUT,-0,2,23,1,1,Y,,,P .
@CHG INPUT$ <NUM>,<QTY> .
```

ECR (Encode Report)

Use an ECR statement to transform a report into code, making it unreadable unless the correct key is specified. The encoded report becomes the current (-0) result.

The ECR statement is particularly useful for highly sensitive reports and messages. If used appropriately, it is more secure than read/write passwords because no one, including the coordinator, can decode the report without knowing the key.

You decode an encoded report with the DECODE function or the DCR run statement (see DCR).

Use caution when encoding reports. Here are some important things to remember:

- Don't forget your key. A report cannot be decoded if you forget the key because no record of it is kept by the MAPPER system. Your coordinator cannot tell you what it is.

- Don't update an encoded report. Any change to the encoded report will corrupt it and you will not be able to decode it. Protect your report from updates by using an update password (see the PSW function in the Manual Functions Reference).

- You cannot move encoded data between form types with different report widths because you will not be able to decode it. If a form type width is changed, all encoded reports in that type will be corrupted and they will not be able to be decoded.

- If you are using a normal ASCII terminal, you may not be able to decode a report that was encoded from a National Character Set (NCS) terminal. In addition, you may not be able to decode a report that contains special NCS characters from a normal ASCII terminal.

- Because of high processing overhead, you may impact system performance if you use the ECR statement excessively.
- Encoding a report approximately doubles the size of that report.

Format

@ECR,m,t,r,key[,hdrs?] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to encode.
key	a one- to eight-character key used to encode the report. Valid characters are A-Z and 0-9. The more characters (up to eight) you use, the more difficult it is for an unauthorized user to guess the key.
hdrs?	a Y if you want report headers to also be encoded. (Default = N.)

Example

This statement encodes RID 10B of mode 0, including headers, using JMT6077 as the key:

```
@ECR,0,B,10,JMT6077,Y .
```

ELT (Element)

The ELT statement copies a MAPPER report or result to a standard OS 1100 program file or symbolic element, or to a data file. The file must be a sector-formatted file with no read or write keys.

If the file you're copying is not a currently assigned file, the ELT statement assigns it with a maximum granularity of 6400 tracks.

Format

@ELT,m,t,r[,lab] qual,fn[,cyc,elt,ver,mapperf?,hdrs?,cs,newcyc?] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report to copy.
lab	the label or relative line number to go to if the run encounters an error (see the STAT1$ and STAT2$ codes following this table).
qual	the qualifier.
fn	the file name to transfer the report to.
cyc	the file cycle.
elt	the element name.
ver	the version.
mapperf?	a Y to copy the report in MAPPER format.

(continued)

(continued)

In field:	Enter:
hdrs?	a Y to include the form type headers. Note that if you enter N in this subfield while processing a result, only the date line is omitted.
cs	the character set of the new report: L Fieldata F ASCII (default) U ASCII, uppercase
newcyc?	a Y to create a new cycle (+1) for the file, and ignore the entry in the *cyc* subfield. Note that the run errs if you try to create a new cycle of a nonexistent file.

Reserved Words

Reserved words: STAT1$ (error codes) and STAT2$	
Code	Error
1	File (relative cycle requested) does not exist.
2	File already assigned to MAPPER system.
3	File already assigned exclusively to MAPPER system.
4	File already assigned to another user.
5	File already assigned exclusively to another user.

(continued)

(continued)

Code	Error
6	File rolled out.
7	Facilities currently unavailable.
8	Private file, under different project-id.
9	Read or write restrictions on file.
10	File not sector-formatted mass storage file.
11	File not program file (if element specified).
12	File is a MAPPER file.
13	System I/O error.
14	Facility warning or reject.
15	Insufficient or improperly formatted statement.
16	File not data file (if element not specified).
17	Cycle attempted on nonexistent file.
18	Attempt to write past end of file.

STAT2$ has a line number identifying the error message.

Use an LSM statement to read the message. Place the number in STAT2$ in the msgno subfield in an LSM statement (see LSM).

Example

This statement creates qualifier MYQUAL file MYFILE and copies the data in RID 2B, mode 0, to it; or the run goes to label 99 if there is an error and the system cannot copy the data:

```
@ELT,0,B,2,99 MYQUAL,MYFILE .
```

DATA CONTROL COMMANDS

Use the same data control commands you use with an STR statement to control data in the report you're copying (see Table 7-14 under STR).

Enter these commands on any line in the report; they take effect from that point on in the data.

EL- (Element Delete)

The EL- statement deletes a standard OS 1100 program file or symbolic element, or a data file. The file must be a sector-formatted file with no read or write keys.

Format

@EL-[,lab] qual,fn[,cyc,elt,ver] .

In field:	Enter:
lab	the label or relative line number to go to if the run encounters an error (see the the STAT1$ and STAT2$ codes following this table).
qual	the qualifier.
fn	the name of the file to delete, or the file name containing the element to delete.
cyc	the file cycle.
elt	the element name.
ver	the version.

Reserved Words

Reserved words: STAT1$ (error codes) and STAT2$	
Code	Error
0	Requested element not found in specified file.
1	File does not exist.
2	File already assigned to MAPPER system.
3	File already assigned exclusively to MAPPER system.
4	File already assigned to another user.
5	File already assigned exclusively to another user.
6	File rolled out.
7	Facilities currently unavailable.
8	Private file, under different project-id.
9	Read or write restrictions on file.
10	File not sector-formatted mass storage file.
11	File not program file (if element specified).
12	File is a MAPPER file.
13	System I/O error.

(continued)

(continued)

Code	Error
14	Facility warning or reject.
15	Insufficient or improperly formatted statement.
STAT2$ has a line number identifying the error message. Use an LSM statement to read the message. Place the number in STAT2$ in the msgno subfield in an LSM statement (see LSM).	

Example

This statement deletes file MYQUAL*MYFILE:

```
@EL- MYQUAL,MYFILE .
```

Use an ESR statement to exit a subroutine and return to a specified line in the run control report that contains the calling RSR statement (see RSR).

NOTE: Don't put other run statements on the same line after an ESR statement. The logic scan of the line terminates after executing the ESR statement.

Format

@ESR[{,*q*|,-*q*}] .

In field:	Enter:
q	the location in the run control report to return to, where *q* is the relative number of lines after or before (-) the line following the RSR statement that called the subroutine. (Blank = 0.)

Example

This example shows a series of statements that could be used to run and execute a subroutine:

```
@SRH .    RETURN HERE IF q =-5
@RNM .    RETURN HERE IF q =-4
@SRH .    RETURN HERE IF q =-3
@MCH .    RETURN HERE IF q =-2
@RSR 1 .  GO TO INTERNAL SUBROUTINE AT LABEL 1; RETURN
          HERE IF q=-1
@GTO .    RETURN HERE IF q =0 OR IS NOT SPECIFIED
@DSP .    RETURN HERE IF q =1
@1:SRH .  RETURN HERE IF q =2
@TOT .    RETURN HERE IF q =3
@WRL .    RETURN HERE IF q =4
@RNM .    RETURN HERE IF q =5
@ESR,q .  EXIT SUBROUTINE AND GO TO q WHERE q IS A NUMBER
          FROM -5 to 5
```

See RSR for other examples using an ESR statement.

Use the EXT statement with updatable results to extract processed lines from the report. The system deletes these lines from the report; these lines become the current result (-0).

The following statements produce updatable results:

CAU
LCH (with the OU option)
LOC (with the OU option)
MAU
SRU

Format

```
@EXT .
```

Example

In this example, all lines found in the Search Update result are deleted from the report; these lines become the current result (-0):

```
@SRU,0,B,2 DH 2-2 □,IP EXT .
```

FDR (Find and Read)

Use an FDR statement to find a line. Follow it with one or more RLN statements (or an RDL statement) to read the line or lines.

Of the following two combinations, the first is more efficient because it uses fewer I/Os.

@FDR . . . RLN . . . (or RDL)

@FND . . . RDL . . .

See also RDC, RDL, and RLN.

NOTE: If a find is made, the report in which the find is made becomes the current -0.

Format

@FDR,m,t[,r,l,q,lab] o cc ltyp,p vrid,vlno .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to find (and later read) a line.
l	the line number at which to start the scan.
q	how many lines (quantity) to scan.
lab	the label or relative line number to go to if no find is made.

(continued)

(continued)

In field:	Enter:
o	options:
	A — Process all line types.
	C(*x*) — Alter normal character set processing:
	C(F) full character set C(L) limited character set C(S) strict character set of report
	R*x*{-*y*|,*y*} — Scan a range of reports from report *x* through report *y*; or reports *x*,*y*,...,*y*
	Examples:
	R2,5 Scan reports 2 and 5 R2-10 Scan reports 2 through 10 R2-10,14 Scan reports 2 through 10 and 14
	@ Find spaces
	/ Find slashes
cc	the column-character positions or names of the fields to scan.
ltyp	the line type to scan. (If you specify the A option, leave this subfield blank.)
p	the find parameters.
vrid	the RID number where the find was made.
vlno	the line number where the find was made.

Example 1: Finding a Particular Line in a Type

```
@FDR,0,B '' 'ST-CD' □,IP <RID>I6,<LINE>I6 .
@RLN,<LINE>,99 'ORDER' <ORD>I .
```

where:

0,B	Find a line in form type B, mode 0.
"	Use no options.
ST-CD	Look in the ST CD field (column 2 for two characters).
□	Process tab lines.
IP	Look for the characters IP.
<RID>I6	Capture the report number where the find was made in <RID>.
<LINE>I6	Capture the line number where the find was made in <LINE>.
RLN,<LINE>,99	Read the line (captured in <LINE>I6) and go to label 99 if no line number exists.
'ORDER'	Read the data in the ORDER field.
<ORD>I	Capture the order number from the ORDER field of the line read.

Example 2: Finding a Line in a Report

```
@FDR,0,B,2,6,100,99 '' 15-5 □,GREEN ,V116 .
@RLN,V1,99 39-5 V215 .
```

where:

0,B,2	Find a line in RID 2B in mode 0.
6	Start looking at line 6.
100	Scan 100 lines.
99	Go to label 99 if no finds are made.
''	Use no options.
15-5	Scan column 15 for five characters.
□	Process tab lines.
GREEN	Look for the characters GREEN.
V116	Capture in V1 the line number where the find was made.
RLN,V1,99 39-5 V215	Capture in V2 the order number on the found line in V1.

FMT (Format)

Use the FMT statement to create a display format by selecting which fields of a report or result to display on a following DSP, OUM, or OUT statement.

You can use directory names or standard column-character syntax to define report fields:

- If you name the fields, the display includes the columns of each field as well as the character immediately following the field.
- If you use the column-character positions, the run displays only the columns specified.

The system always includes column 1, which contains the line type designator, in the format.

You can list fields in any order; however, fields are always displayed in the same order they appear in the report.

Format

@FMT[,*m,t,r*] *field*[,...,*field*] .

In field:	Enter:
m,t,r	the mode, form type, and RID number of the report or result from which to display fields. (Default = -0.)
field	the fields to display (can be field names or column-character positions).

Examples

This statement displays the ST CD, SHIP DATE, and CUST CODE fields of the current -0:

```
@FMT 'ST-CD','SHIP-DATE','CUST-CODE' .
@DSP,-0 .
```

This example displays column 2 for 3 characters, column 45 for 5 characters and column 64 for 7 characters from report 2B in mode 0:

```
@FMT,0,B,2 2-3,45-5,64-7 .
@DSP,0,B,2 .
```

FND (Find)

The FND statement scans vertically through a report for an item, multiple items, or a range of items.

A find is different from a search. With a FND statement, you load a variable with the line number where the find is made; a search creates a result.

To update a single line, use a FND statement followed by a WRL statement (see WRL).

NOTE: If a find is made, the report in which the find is made becomes the current -0.

Format

@FND,m,t[,r,l,lab] o cc ltyp,p vrid,vlno .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to find data.
l	the line number at which to start the find.
lab	the label or relative line number to go to if no lines or data are found.
o	options:
	A Process all line types.

(continued)

(continued)

In field:	Enter:
	C(x) Alter normal character set processing: C(F) Full character set C(L) Limited character set C(S) Strict character set of report See also Appendix E.
	Rx{-y|,y} Scan a range of reports from report x through report y; scan reports x,y,...,y.
	Examples: R2,5 Find in reports 2 and 5. R2-10 Find in reports 2 through 10. R2-10,14 Find in reports 2 through 10 and 14.
	@ Find spaces.
	/ Find slant as data.
cc	the column-character positions or names of the fields in which to find.
ltyp	the line type to scan. (If you specify the A option, leave this subfield blank.)
p	the find parameters.
vrid	the RID number where the find was made.
vlno	the line number where the find was made.

Example 1: Searching a Type for an Item

This statement searches for IP in the ST CD field and captures the report and line numbers where the find was made:

```
@FND,0,B '' 'ST-CD' □,IP V1I6,V2I6 .
```

where:

0,B	Find data in form type B, mode 0.
"	Use no options.
ST-CD	Look in the ST CD field (column 2 for two characters).
□	Process tab lines.
IP	Look for the characters IP.
V1I6	Capture in V1 the report number where the find was made.
V2I6	Capture in V2 the line number where the find was made.

Example 2: Searching a Report for an Item

This statement uses traditional column-character syntax, searching for GREEN in column 15 for five characters, and captures the line number where the find was made:

```
@FND,0,B,2,6,99 '' 15-5 □,GREEN ,<LINE>I6 .
```

where:

0,B,2	Find data in RID 2B in mode 0.
6	Start looking at line 6.

99	Go to label 99 if no finds are made.
"	Use no options.
15-5	Scan column 15 for five characters.
□	Process tab lines.
GREEN	Look for the characters GREEN.
<LINE>I6	Capture the line number where the find was made in <LINE>.

Example 3: Searching for Spaces

This statement searches a field for spaces and captures the line number where the find was made:

```
@FND,0,B,2,,99 @ 25-6 □,@@@@@@ ,V1I6 .
```

where:

0,B,2	Find data in RID 2B in mode 0.
99	Go to label 99 if no finds are made.
@	Use the @ option to find spaces.
25-6	Find spaces in the SERIAL NUMBER field (column 25 for six characters).
□	Process tab lines.
@@@@@@	Look for spaces.
,V1I6	Capture in V1 the line number where the find was made.

GOC (Generate Organization Chart)

Use a GOC statement to generate organization charts using the statement's own command language.

Format

@GOC,m,t,r[,lab] ulvl?[,notxt?,ige?,igcz?,fxcz?] optmz?[,outrslt?,unp? vlinesi,vlineso,vco,vbuffz] .

In field:	Enter:
m,t,r	the mode, form type, and RID number.
lab	the label or relative line number to go to in case of error.
ulvl?	a Y if you want an upper level chart. If you type a Y, the statement produces a special chart that contains only one box at all levels except the bottom level.
notxt?	a Y to eliminate all text. This produces a chart containing boxes with no text (useful for determining the size of the boxes).
ige?	a Y to ignore all error conditions.
igcz?	a Y to ignore the character size. If you type a Y, the statement does not check whether the resulting character size is within the correct range for the selected display device.
fxcz?	a Y to select a fixed character size to use; this character size is dependent upon the selected display device.

(continued)

(continued)

In field:	Enter:
optmz?	a Y to optimize the result. A Y in this field reduces the number of characters necessary to produce the chart. (If you select the LOGO or FONT parameter, *optmz?* must be Y.)
outrslt?	a Y to display the graphics result on the screen.
unp?	a Y to unpack the result. If you type a Y, the result is displayed in unpacked format, where the resulting primitive commands are not split across successive lines. (Ignored if *optmz?*=Y.)
vlinesi	the variable to capture the number of input lines scanned following the header-divider line (*=).
vlineso	the variable to capture the number of output lines in the result following the header-divider line (*=).
vco	the variable to capture the number of characters to output.
vbuffz	the variable to capture the size of the buffer used to hold chart information.

Reserved Words

If the run encounters an error in the data RID, it goes to the label in the *lab* subfield and loads STAT1$ and STAT2$ with information about the error.

Reserved words: STAT1$ and STAT2$	
Word	Content
STAT1$	Message number. (Use an LSM statement to obtain the message.)
STAT2$	The line number where the error occurred.

Example

The Color Graphics Guide contains an example of an organization chart and the code used to produce it. With that code in RID 81H, mode 0, the following statement generates the organization chart:

```
@GOC,0,H,81 N N .
```

See the Color Graphics Guide for more details about generating organization charts.

Use the GS statement to increase or decrease the size of your charts and to create custom graphics using the expanded syntax.

See the Color Graphics Guide for more details about the GS statement as well as information about using primitive graphics code.

Format

@GS,m,t,r[,lab] maxy[,o,ige?,unp?,aga?,expand?,ighitxt?,outrslt?] sf[,offx,offy] angle[,absx,absy vci,vco,vminx,vmidx,vmaxx, vminy,vmidy,vmaxy] .

In field:	Enter:
m,t,r	the mode, form type, and RID number of the report containing the primitive graphics code.
lab	the label or relative line number to go to in case of error.
maxy	the maximum Y value to check the limits on (ranging from 0 to 32767). Type 0 if you want 32767.
o	options: A Put all absolute commands in the result (whenever possible). C Optimize and combine successive commands into a single command separated by commas. O Optimize resulting code.

(continued)

(continued)

In field:	Enter:
	S Strict interpretation of primitive graphics code. (Without the S option, all lowercase characters not in a text string are interpreted as uppercase characters.) If you use the S option, the *ige?*, *unp?*, *aga?*, and *expand?* subfields must all be N. (Type N or leave them blank.) Leave the options field blank or type N for normal mode (where leading spaces, for example, are ignored).
ige?	a Y to ignore errors. If you type a Y, values that are out of range are truncated, and invalid commands and sequences are ignored. (If you use the S option, *ige?* must be N.)
unp?	a Y to unpack the result. If you type a Y, the result is displayed in unpacked format with one command per line. (If you use the S option, or if *outrslt?* = Y, *unp?* must be N.)
aga?	a Y to assume graphics active. If you type a Y, it is assumed that ZI preceded the data, and no closing ZT is required to end the data. (If you use the S option, or if *outrslt?* = Y, *aga?* must be N.)
expand?	a Y to handle expanded syntax. (If you use the S option, *expand?* must be N.)
ighitxt?	a Y to ignore high text. If you type a Y, it is never converted to high-quality text, even if it encounters a CT2 or rotated text.
outrslt?	a Y to display the graphics result on the screen. If you type a Y, *unp?* and *aga?* must be N.

(continued)

(continued)

In field:	Enter:
sf	the scaling factor (floating point 0.0 to 32767.0). 0 = default value 1.0; 2.0 produces a result twice as large as input; .5 a result half as large.
offx	the offset value (-9999999999 to 9999999999) added to the absolute X components.
offy	the offset value (-9999999999 to 9999999999) added to the absolute Y components.
angle	the rotation angle (-360.0 to 360.0) used to rotate all X and Y values and angles. Positive angles produce counterclockwise rotation; negative angles produce clockwise rotation.
absx	the absolute X rotation value (-9999999999 to 9999999999).
absy	the absolute Y rotation value (-9999999999 to 9999999999).
vci	the variable to capture the number of characters scanned.
vco	the variable to capture the number of characters to output.
vminx	the variable to capture the minimum X value.
vmidx	the variable to capture the midpoint X value.
vmaxx	the variable to capture the maximum X value.
vminy	the variable to capture the minimum Y value.
vmidy	the variable to capture the midpoint Y value.
vmaxy	the variable to capture the maximum Y value.

Reserved Words

If the run encounters an error in the data RID, it goes to the label in the *lab* subfield and loads STAT1$ and STAT2$ with information about the error.

Reserved words: STAT1$ and STAT2$	
Word	Content
STAT1$	Message number. (Use an LSM statement to obtain the message.)
STAT2$	One less than the line number in report being processed where error occurred.

Example 1

```
@GS,0,A,1 20000 1 0 .
```

where:

0,A,1	Scale RID 1A, mode 0.
20000	Create maximum Y values to 20,000.
1	Set scale factor to 1.
0	Rotate angle to 0.

Example 2

```
@GS,0,B,2,10 23999,O,N,N,N,N,Y 0.75,2000,1000 \
45,16383,11399 V1I5,V2I5,V3I5,V4I5,V5I5,V6I5,\
V7I5,V8I5 .
```

where:

0,B,2	Scale RID 2B, mode 0.
10	Go to label 10 in case of error.
23999	Maximum Y value.
O	Optimize the results.
N,N,N,N,N	Do not ignore errors, unpack result, assume graphics code, handle expanded syntax, or ignore high text.
Y	Display the result on the screen.
0.75	Reduce to 3/4 size.
2000	Move to the right 2000.
1000	Move up 1000.
45	Rotate result 45 degrees counterclockwise.
16383	Rotate around the X value 16383.
11399	Rotate around the Y value 11399.
V1I5	Capture in V1 the number of characters scanned.
V2I5	Capture in V2 the number of characters in the result.
V3I5	Capture in V3 the minimum X value.

V4I5	Capture in V4 the midpoint X value.
V5I5	Capture in V5 the maximum X value.
V6I5	Capture in V6 the minimum Y value.
V7I5	Capture in V7 the midpoint Y value.
V8I5	Capture in V8 the maximum Y value.

Use a GTO statement to branch within a run.

Format

@GTO {*lab* | END[,*n*,*n*,*n*] | LIN [+]*n* | LIN -*n* | RPX *r*} .

In field:	Enter:
lab	the line label number (may be a variable).
END[,*n*,*n*,*n*]	**END** to indicate the end of run and display the output area. For runs started from a LNK statement, *n*,*n*,*n* are up to three status codes that the linked run can return to the original run. Codes can be integers (or variables with integer values) in the range -34359738367 to +34359738367. The codes are ignored if the run was not started via LNK (see LNK).
LIN	**LIN** to indicate the present line position.
+*n**	the number of lines following the present line (*n* must be an integer greater than 0).
-*n**	the number of lines preceding the present line (*n* must be an integer greater than 0).

* +n, -n, and r may be variables. Do not, however, type a minus sign in front of the variable. The plus sign is optional with the line number, so if V1 contained 5, you could use @GTO LIN V1. Here's another example: @GTO LIN V2, where V2 contains the value -5, is the same as saying @GTO LIN -5.

(continued)

(continued)

In field:	Enter:
RPX	**RPX**, which is the call to go to another run control report in the same mode and type.
*r**	the RID number of a run control report in the same mode and type (cannot be a result).

* +n, -n, and r may be variables. Do not, however, type a minus sign in front of the variable. The plus sign is optional with the line number, so if V1 contained 5, you could use @GTO LIN V1. Here's another example: @GTO LIN V2, where V2 contains the value -5, is the same as saying @GTO LIN -5.

Examples

This statement continues the run at label 7:

```
@GTO 7 .
```

The following statement continues the run at the line with the label equaling the contents of V6:

```
@GTO V6 .
```

This statement stops the run and displays the contents of the output area:

```
@GTO END .
```

This statement continues the run two lines beyond the current statement line:

```
@GTO LIN +2 .
```

This statement causes the run to go to the MAPPER run in report 2 of the same mode and type:

```
@GTO RPX 2
```

The GTO RPX statement executes the run statements in the specified run control report, with these considerations:

- The run control report being entered must be in the same mode and form type as the run that has the GTO RPX statement.
- All security checks for the first run must be met by the run control report being entered.
- All variables established in the run having the GTO RPX statement are valid in the run being entered. Labels are not valid.
- The run being entered by a GTO RPX statement need not be registered. However, since the RPX run control report resides in a form type especially for MAPPER runs, inform your coordinator that you intend to use RPX in the form type as part of the plan.

For examples of computed IF/GTO statements, see IF.

IDU (Index User)

An IDU statement produces an index of users' reports by mode, form type, date, and range, and creates a result. An IDU statement lets you find reports created (or last updated) by a specific user.

NOTE: You cannot use the IDU statement to index a mode and type in which your run control report is located.

Format

@IDU,m,t|,q,user,stdate,endate,strid,endrid vrids,vlines,vridst,vhiridt| .

In field:	Enter:
m,t	the mode and type to index.
q	how many lines (quantity) to display. (Default = header lines.)
user	the user-id to index or **ALL** to index all users).
stdate	the starting date for a date range (in format *ddmmmyy*). This is the date of the most recent update to the report, or the creation date if there are no updates.
endate	the ending date for a date range (in format *ddmmmyy*). This is the date of the most recent update to the report, or the creation date if there are no updates.
strid	the starting RID (for a range of reports).
endrid	the ending RID (for a range of reports).

(continued)

(continued)

In field:	Enter:
vrids	the variable to capture the number of RIDs found.
vlines	the variable to capture the number of lines found.
vridst	the variable to capture the number of RIDs in the form type.
vhiridt	the variable to capture the highest RID number in the form type.

Examples

This statement indexes all of this user's form type A reports, including header lines from each report in the result:

```
@IDU,0,A .
```

The following statement indexes a range of reports under the user-id of JDOE:

```
@IDU,0,B,7,JDOE,01JAN87,31DEC87,5,10 V113,V219,\
V314,V414 .
```

where:

0,B	Index mode 0, type B reports.
7	Display seven lines of data from each report indexed.
JDOE	Index user-id JDOE.
01JAN87	Index reports starting on January 1, 1987.

31DEC87	Index reports ending on December 31, 1987.
5	Start indexing at report 5.
10	Stop indexing at report 10.
V1I3	Capture in V1 the number of reports found.
V2I9	Capture in V2 the number of lines found.
V3I4	Capture in V3 the number of reports in the type indexed.
V4I4	Capture in V4 the highest report number created.

Use an IF statement to test and compare values and make logical decisions.

Format 1 — Common

@IF[,C] *val1 op val2 stmt ; .*

Format 2 — Logical OR

@IF[,C] *val1 op val2,val3*[,*valn,...,valn*] *stmt ; .*

Format 3 — Logical AND

@IF[,C] *val1 op val2 & op val3* [*& op valn ... & op valn*] *stmt ; .*

Format 4 — Computational IF/GTO

@IF[,C] *val1 op val2,(lab)*[,*valn,(lab),...,valn,(lab)*] *; .*

In field:	Enter:
C	a C to distinguish uppercase from lowercase characters.
val1	a value, usually in a variable or a reserved word (reserved words must end with a dollar sign).

(continued)

(continued)

In field:	Enter:	
op	one of these relational operators:	
	= or EQ	Equal
	GE	Greater than or equal to
	> or GT	Greater than
	LE	Less than or equal to
	< or LT	Less than
	NE	Not equal
	NOT = or NOT EQ	Not equal
	NOT < or NOT LT	Not less than
	NOT > or NOT GT	Not greater than
val2	a value to compare to *val1*.	
stmt	a statement indicating the action to take if the IF statement condition is met (TRUE).	
△;	space-semicolon to terminate the IF statement. Always enter a space before the semicolon.	
,val3,valn	a comma (logical OR) and a third value, fourth value, and so forth.	

(continued)

(continued)

In field:	Enter:
& op val3 & op valn	an ampersand (logical AND), another relational operator, and a third value, fourth value, and so forth.
,(lab)	a comma and label number or location in the run to go to if the IF statement condition is met (TRUE). Here are valid GTO locations: END — End of run LIN +*n* — Number of lines following present line, where *n* is an integer LIN -*n* — Number of lines preceding present line, where *n* is an integer RPX *r* — RID *r* (another run control report), where *r* is a RID number in the same mode and form type

If the condition or conditions of the IF statement are met (TRUE), the run executes the statement specified.

If the conditions of the IF statement are not met (FALSE), the run continues executing at the next logical line. The next logical line begins after the run encounters either a semicolon (;) or the end of the actual line. Note that all examples contain a semicolon.

Examples

If USER$ equals JDOE, go to label 2; or else continue:

```
@IF USER$ = JDOE GTO 2 ; .
```

If V1 equals V2, load V1 with 1 and continue; or else continue:

```
@IF V1 = V2 LDV V1=1 ; .
```

If <VAL1> and <VAL2> are not equal, go to label 3; or else continue:

```
@IF <VAL1> NE <VAL2> GTO 3 ; .
```

If V21 is greater than V20, go to label 3; or else continue:

```
@IF V21 > V20 GTO 3 ; .
```

If <NUM> equals 2 OR 4, go to label 1; or else continue:

```
@IF <NUM> EQ 2,4 GTO 1 ; .
```

If V1 is greater than 0 AND less than 100, go to label 3; or else continue:

```
@IF V1 > 0 & < 100 GTO 3 ; .
```

If V22 is greater than 10 AND less than 50, go to label in V99; or else continue:

```
@IF V22 > 10 & < 50 GTO V99 ; .
```

This example uses two IF statements. If V1 equals A and if V2 is less than B, go to label 1; or else continue:

```
@IF V1 = A IF V2 LT B GTO 1 ; .
```

The following examples use a computed IF/GTO sequence.

If <TOTAL> equals 30, go to label 1; if <TOTAL> equals 40, go to label 2; if <TOTAL> equals 50 OR 60, go to label 3; or else continue:

```
@IF <TOTAL> = 30,(1),40,(2),50,60,(3) ; .
```

If V2 equals 4, go to the next line, OR if V2 equals 5, go to the end of the run; or else continue:

```
@IF V2 = 4,(LIN +1),5,(END) ; .
```

If V1 equals 2, execute the LDV statement (load V3 with the value 4), go to label 1 and continue; if V1 does not equal 2, go to label 2:

```
@IF V1 = 2 LDV V3=4 GTO 1 ; GTO 2 .
```

Execute the LDV statement (load V3 with the value 4) only if V1 equals 2; in either case (TRUE or FALSE), go to label 2 and continue:

```
@IF V1 = 2 LDV V3=4 ; GTO 2 .
```

Note that in the previous example, the run goes to label 2 even if the condition of the IF statement is not met, because the GTO statement is on the next logical line.

This example uses an unknown trailing substring. (The 0-3 specifies the last three characters; the starting column position is unknown.) If the last three characters of V1 contain MON, go to label 25:

```
@IF V1(0-3) = MON,(25) ; .
```

This example uses a known trailing substring. (The 3-0 specifies the known starting position of 3 for the remainder of the field.) If the characters beginning with character 3 through the end of the field contain FRI, go to label 26:

```
@IF V1(3-0) = FRI,(26) ; .
```

INC (Increment Variables)

Use an INC statement to increase the numeric value of variables. You cannot use an INC statement with string (type S) variables. Also, if the variable you're trying to change contains alphabetic or special characters, the variable remains unchanged.

An INC statement is more efficient than a CHG statement, and it requires fewer characters.

Format

@INC[,*n*] *v*[,*v*,...*v*] .

In field:	Enter:
n	the floating-point or integer number by which to increase a value. (Default = 1.)
v	the variables you want to increase by *n*.

Examples

To add 1 to the numeric value of V2, use:

```
@INC V2 .
```

instead of:

```
@CHG V2 V2 + 1 .
```

To add 3 to the numeric values in V1 and V3, use:

```
@INC,3 V1,V3 .
```

The IND statement indexes a specific form type in a specified mode, and creates a result. The result contains a date line and lists the number of reports in the form type, the total number of lines in the form type, the number of lines contained in each report, and a specified number of lines from each report.

NOTE: If you are indexing reports that are fewer than 80 characters wide, the information containing the number of lines from each report is added as trailer lines.

Format

@IND,*m,t,q*[,*lab*] .

In field:	Enter:
m,t	the mode and type to index.
q	how many lines (quantity) to display from each report.
lab	the label or relative line number to go to if no reports exist in the specified form type.

Reserved Words

Reserved words: STAT1$ and STAT2$	
Word	Content
STAT1$	Number of reports in form type
STAT2$	Total number of lines in form type

Example

This example indexes mode 0, type A, and creates a result that has the first four lines from each report in the form type. If no reports exist, the run goes to label 99.

```
@IND,0,A,4,99 .
```

Use an INS statement to insert data from one variable into another.

Format

@INS *vld substrv* .

In field:	Enter:
vld	variables, literal data, reserved words, or any combination of these, to insert.
substrv	a substring of a variable not exceeding 18 characters to receive data.

Examples

This statement inserts ABCD in <STRING>, starting at column <COL> for four characters:

```
@INS ABCD <STRING>(<COL>-4) .
```

This statement inserts a portion of V2 from column V1 for V3 characters into V4 at column 6 for V3 characters:

```
@INS V2(V1-V3) V4(6-V3) .
```

This example uses a CHG statement to initialize V1 to QQ and V2 to AAA, and an INS statement to insert V1 in V2. V2 then equals AQQ:

```
@CHG V1H2 QQ CHG V2H3 AAA .
@INS V1 V2(2-2) .
```

JUV (Justify Variables)

Use a JUV statement to numerically justify the contents of variables. You can justify any numeric variable up to 18 characters. Once justified, a variable retains that justification until its content is altered. If the JUV statement finds the content of a requested variable to be nonnumeric or finds that the variable's size is greater than 18 characters, it does not justify the variable.

Format

@JUV,*o v*[,*v*,...*v*] .

In field:	Enter:
o	an option (you must use one of these options):

C Insert commas in the integer portion of the variable every third digit, eliminate leading zeros and nonsignificant trailing zeros, and insert the resulting value in the rightmost portion of the variable.

NOTE: You must remove commas inserted by the C option before executing a CHG or IF statement against the variable. Use any other option to remove commas.

D Delete commas, eliminate any leading zeros, and insert the resulting value in the rightmost portion of the variable.

(continued)

(continued)

In field:		Enter:
	L	Left-justify the contents of the variable, eliminate leading zeros and nonsignificant trailing zeros, and insert the resulting value in the leftmost portion of the variable followed by spaces.
	R	Right-justify the contents of the variable, eliminate leading zeros and nonsignificant trailing zeros, and insert the resulting value in the rightmost portion of the variable preceded by spaces.
	X	Expand the contents of the variable; eliminate leading zeros; insert the resulting value in the leftmost portion of the variable, add a decimal point where required, and insert zeros in the rightmost portion of the variable.
	Z	Right-justify the contents of the variable and insert leading zeros in the leftmost portion; eliminate nonsignificant trailing zeros.
v	a variable or variables to justify.	

Examples

In the following examples, Δ stands for a blank character position.

Initialize <STRING> to a type A, 12-character variable with a value of 6543.210:

```
@CHG <STRING>A12 6543.210 . <STRING> = 6543.210ΔΔΔΔ
```

Insert commas in <STRING>:

```
@JUV,C <STRING> . <STRING> = ΔΔΔ6,543.210
```

Delete commas from <STRING>:

```
@JUV,D <STRING> . <STRING> = ΔΔΔΔ6543.210
```

Left-justify the contents of <STRING>:

```
@JUV,L <STRING> . <STRING> = 6543.21ΔΔΔΔΔ
```

Right-justify the contents of <STRING>:

```
@JUV,R <STRING> . <STRING> = ΔΔΔΔΔ6543.21
```

Expand the contents of <STRING>:

```
@JUV,X <STRING> . <STRING> = 6543.2100000
```

Right-justify the contents of <STRING> and add leading zeros:

```
@JUV,Z <STRING> . <STRING> = 000006543.21
```

Use the KEY statement before a DSP, OUT, or SC statement to determine which function key the user pressed after the noninterim display.

Format

@KEY .

Reserved Word

Reserved word: FKEY$
The run continues executing after the DSP, OUT, or SC statement, and FKEY$ contains:
0 XMIT (with cursor below the control line) 1 F1 or RSM 2 F2 or PNT 3-22 F3-F22
FKEY$ always contains 0 if a KEY statement has not been executed.

Example

In this example, the KEY statement requests function key input, the DSP statement displays RID 2B in mode 0, and the LDV statement loads V1 with the contents of FKEY$. V1 can then be tested for its contents, and the run can be processed accordingly.

```
@KEY .
@DSP,0,B,2 .
@LDV,PW V112=FKEY$ .
```

LCH (Locate and Change)

The LCH statement scans specific column-character positions of a report for a specified target string, changes it to a specified replacement string, and creates a result.

Format

@LCH,m,t,r[,l,lab] o cc tgtstr/replstr[,vlines,vrid] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to locate and change data.
l	the line number at which to start the scan.
lab	the label or relative line number to go to if no target string is located.
o	options (use the A, F, and M options to change strings regardless of line type): A — Process all line types. (If used alone, it ignores the first character of the target string and changes the line type of all lines having finds to the first character of the replacement string.) C — Distinguish between uppercase and lowercase characters. (Applies only to FCS reports.) See also Appendix E.

(continued)

(continued)

In field:	Enter:
F	Process all line types and locate and change the entire string. (Does not locate strings that start in column 1.)
M	Treat the first character of the target string as the line type designator. (Use the M option instead of a line-type subfield. Also, you must use the M option to locate a string beginning in column 1.)
O	Create a result containing the items found.
OU	Create a result; then do one of the following: enter del to delete the changed lines from the report (see DEL), enter ext to delete the changed lines from the report and redisplay the result (see EXT), or make changes in the result (if desired) and enter upd to blend the changed lines from the result back into the report (see UPD). You cannot execute LCH with the OU option against a result.
S*x*	Start the scan at line *x*, where *x* is a positive number.
S*x*-*y*	Start the scan at line *x* and stop at line *y*.

(continued)

(continued)

In field:	Enter:
	S*x*,*n* Start the scan at line *x* and scan *n* lines.
	T*x* Set *x* to a transparent character to match any character in that position. Don't use the transparent character in the first character position of the target string.
cc	the column-character positions or names of the fields in which to scan.
tgtstr	the string of characters to locate.
replstr	the string of characters with which to replace the target string.
,	This subfield is not used in LCH statements. (Type the comma if continuing the statement.)
vlines	the number of lines located that contain the target string.
vrid	the RID number where the target string was located.

Example

```
@LCH,0,A,1,40,99 AFM 2-79 EXECUTION/PROCESSING ,\
<LINES>I4 .
```

where:

0,A,1	Process mode 0, type A, RID 1.
40,99	Start scan at line 40 and go to label 99 if no target string is located.
AFM	Use A, F, and M options.
2-79	Start scan in column 2 for 79 characters.
EXECUTION/ PROCESSING	Target and replacement strings: each time EXECUTION is encountered, replace it with PROCESSING.
<LINES>I4	Capture number of lines located that contain the target string in <LINES>i4.

LCV (Locate/Change Variable)

Use an LCV statement to locate (and optionally change occurrences of) a string within a variable.

You can use an LCV statement to literally compare the contents of one variable to another. Using an IF statement and a GTO statement to compare the contents of variables limits you to 18 characters in the variables. Using an LCV statement with a label lets you compare up to 132 characters, and it is more efficient than using IF and GTO statements.

An LCV statement is to variables what a LOC or an LCH statement is to data in reports.

You can use an LCV statement on all variable types, and you can use substrings.

Format

@LCV[,lab] o v tgtstr[/replstr vcol,voccs] .

In field:	Enter:
lab	the label or relative line number to go to if no finds are made.
o	options:
	B{*n*|*n*-*x*} "Bail out" option
	B*n*: Locate the *n*th occurrence of the target string or change *n* occurrences.

(continued)

(continued)

In field:	Enter:
	If you're locating but not changing, B1 is assumed (thus, you don't need to specify the B option). If you're locating and changing and you know there's only one occurrence, use B1; if there are two occurrences, use B2; etc. B*n*-*x*: Starting at the *n*th occurrence, change *x* occurrences of the target string.
C	Distinguish uppercase from lowercase characters.
L*x*	If the first character in the variable doesn't match *x*, don't consider it a find. The L option is especially useful for locating in lines that were read from a report.
M	When changing, do not insert transparent characters (see T*x* option) from the replacement string in the variable, but leave these character positions the same. (Valid only if the *replstr* field is used.)
N	Do not go to the label if no finds are made (or if the bail-out quantity specified in the B option is not found). Instead, go to the label if a find is made.
T*x*	Set *x* to a transparent character to automatically match any character in that position. Since the transparent character's default value is a space, you must use the T option to locate spaces.

(continued)

(continued)

In field:	Enter:
v	the variable to locate (and optionally change).
tgtstr	the string of characters in a variable to locate (1 to 132 characters).
replstr	a string of 0 to 132 characters to replace the target string in the variable with. If you don't use the B option, it changes all occurrences.
vcol	the column number in the variable where the first occurrence of the target string begins. With the B option, it is the column of *n*th occurrence of the target string. It contains 0 if there are fewer than *n* occurrences in the variable.
voccs	the number of occurrences of the target string found. This is useful when doing a change or locate with the B option.

Example 1: Locating Second Occurrence of an Item

Here are the contents of V1 for this example:

```
V1=DOGHORSECOWPIGBIRDCATCATCATCATMOUSE
                  |                  |
              column 15          column 34
                 | locate columns   |
```

This statement locates the second occurrence of CAT in V1 starting at column 15 for 20 characters:

```
@LCV,1 B2 V1(15-20) CAT V2I6 .
```

where:

1	Go to label 1 if fewer than two occurrences of CAT are found.
B2	Bail out on the second occurrence of CAT.
V1(15-20)	Scan V1 starting in column 15 for 20 characters.
CAT	Locate target string CAT.
V2I6	Capture the column number where the second occurrence of CAT begins in V2.

After the statement executes, V2 contains 22 (the column in V1 where the second occurrence of CAT was found).

Example 2: Counting Occurrences of an Item

In this example, V1 contains:

```
CATdogCATDOGCATDogCATDOGCATdog
```

This statement counts the number of times DOG occurs in V1:

```
@LCV B99 V1 DOG ,V3I6 .
```

where:

B99	Bail out on the 99th occurrence of DOG.
V1	Scan V1 (the entire variable).

DOG	Locate target string DOG.
V3I6	Capture the number of occurrences of DOG in V3.

In this example, the bail-out count was set to a value higher than the number of occurrences of DOG because we didn't want to bail out. V3 continues to count each occurrence of DOG. After the statement executes, V3 contains 5.

Example 3: Changing Character Strings

In this example, V1 contains:

```
*CATDOGCATDOGCATDOGCATDOGCATDOG
```

This statement changes the second, third, and fourth occurrences of DOG to CAT in V1:

```
@LCV L*B2-3 V1 DOG/cat V2I6 .
```

where:

L*	Make the change only if the first character of V1 is an asterisk (the line type indicator).
B2-3	Change target string starting at the second occurrence for three occurrences.
V1	Scan V1.
DOG	Locate target string DOG.
cat	Change target string DOG to cat.
V2I6	Capture column number of first occurrence of target string changed.

Since the first character of V1 matches the character specified in the L option, the change is made. V2 equals 11 and V1 contains:

```
*CATDOGCATcatCATcatCATcatCATDOG
    1       2       3       4       5
          v2=11
```

Example 4: Comparing Strings

Here are the contents of <STRING1> and <STRING2>:

```
<STRING1>=ABC123
<STRING2>=ABC+++
```

This statement compares <STRING1> to <STRING2>:

```
@LCV,1 '' <STRING1> <STRING2> .
```

where:

1	Go to label 1 if <STRING1> is not equal to <STRING2>.
''	Don't use any options.
V1 V2	Variables to compare.

Since <STRING1> is not equal to <STRING2>, the run goes to label 1.

Remember, if you use the N option, the run does NOT go to the label unless the two variables are equal.

Example 5: Masking Transparent Characters in the Replacement String

For this example, V1 contains:

```
BLACKBOX1*BLACKCAN1*BLACKBAG1*BLACKCUP1
```

This example uses the M option to locate each occurrence of BLACK followed by any three characters, followed by the number 1. Each time the locate string is found, the characters BLACK are changed to GREEN, the next three characters remain unchanged, and the last character, 1, is changed to 2.

```
@LCV M V1 'BLACK   1'/'GREEN   2' V2I6,V3I6 .
```

where:

M	Mask option — transparent characters in the replacement string are not inserted into the variable being changed.
V1	Variable in which to locate.
'BLACK 1'	Target string (characters to locate).
/	Change string delimiter.
'GREEN 2'	Replacement string (characters with which to replace each occurrence of the target string).
V2I6	Variable to capture the column of the first occurrence changed.
V3I6	Variable to capture the number of occurrences changed.

Variable V1 now contains:

GREENBOX2*GREENCAN2*GREENBAG2*GREENCUP2

Example 6: Using an Unknown Trailing Substring

For this example, V1 contains:

```
FEATURE010101
```

This statement uses an unknown trailing substring and bails out after the second find:

```
@LCV,1 B2 V1(0-6) 01 V2I3 .
```

where:

1	Go to label 1 if no finds are made.
B2	Bail out after the second occurrence.
V1(0-6)	Scan the last six characters of V1. (The 0-6 specifies the last six characters; the starting character position is unknown.)
01	Locate the target string 01.
V2I3	Capture the column number where the second occurrence of 01 begins in the specified substring. (In this example, V2 contains 10 because the second 01 begins in the tenth column of V1.)

Example 7: Using a Known Trailing Substring

In this example, <MONEY> contains:

```
$$$DOLLARS
```

This example uses a known trailing substring to scan column 4 through the end of the field and changes the word "DOLLARS" to the word "YEN" in all occurrences:

```
@LCV,1 '' <MONEY>(4-0) DOLLARS/YEN .
```

where:

1	Go to label 1 if no finds are made.
''	Use no options.
<MONEY>(4-0)	Scan <MONEY> beginning with column 4 through the remaining characters. (The 4-0 specifies the known starting position of 4 through the end of the field.)
DOLLARS/YEN	Change the target string DOLLARS to YEN.

LDV (Load Variables)

Use an LDV statement to perform these tasks:

- ☐ Initialize variables
- ☐ Load variables from other variables
- ☐ Load variables already initialized, including those with a substring of unlimited length
- ☐ Load variables with the contents of reserved words
- ☐ Load a variable with its own contents

When you load a variable with its own contents, you need only specify the receiving variable. This is particularly useful for packing variables. You can use, for example, @LDV,P V1,V2 rather than @LDV,P V1=V1,V2=V2. Loading variables in this manner is also useful when centering, left-justifying, right-justifying variables, and initializing space filled variables without setting the characters equal to spaces.

You cannot do arithmetic computations with an LDV statement as you can with a CHG statement (see CHG). However, an LDV statement is faster and more efficient than a CHG statement for loading literal data and reserved words.

Whenever an LDV statement encounters a space or comma, it stops loading the variable with data. To place a space or comma in a variable without placing either character in a variable beforehand, enclose spaces and commas in apostrophes.

Format

@LDV[,*o*] *v=vld*[,*v =vld*,...,*v=vld*] .

In field:	Enter:
o	options:
	C Center data within the variable.
	H Test the remote run link.*
	N Load the variable with a number based on the input. (For case sensitivity, use the S option also.)*
	L Left-justify the data being loaded.
	P Pack the variable down to significant characters only.
	R Right-justify the data being loaded.
	S Treat the input as case sensitive (valid only with the N option).*
	U Convert all alphabetic characters to uppercase.
	W Load the variable with the value of the reserved word. (The reserved word value will be left-justified.) Make sure the variable is large enough to hold the information in the reserved word. You can also use the P option to pack the variable. Use a DEF statement with the S option (DEF,S) to determine the size of the reserved word (see DEF).

* Don't use the v=vld subfield with these options. See further explanation in this subsection for the format.

(continued)

(continued)

In field:	Enter:
	Z Zero fill the data being loaded.
v	the variable being loaded.
vld	the information to load (variables, literal data, reserved words, or any combination of these).

LOADING MULTIPLE VARIABLES

In the following example, the first LDV statement loads V1 through V4; the second LDV statement then loads these four variables into V5:

```
@LDV V1A5=THIS,V2A3=IS,V3A3=AN,V4A8=EXAMPLE .
@LDV V5S40=V1V2V3V4 .
```

V5 now contains THIS IS AN EXAMPLE.

H OPTION — TESTING A REMOTE RUN LINK

The H option tests the remote run link between two sites to see whether the remote site is online or offline.

NOTE: With a non-GCS remote run link, the H option only indicates whether or not the remote site is configured.

Format

@LDV,H *v=rms* **.**

In field:	Enter:
v	the variable to capture test result: 0 site is offline. 1 site is online.
rms	the remote site number.

N OPTION — LOADING A VARIABLE WITH A NUMBER BASED ON INPUT

The N option produces a number within a specified range for a data item such as a name, address, or item description. As long as the LDV statement remains the same, the data item always receives the same number. The number is especially useful for indexing related reports to specific data.

Format

@LDV,N *v=vld,minmax* .

In field:	Enter:
v	the variable being loaded.
vld	the information to load (variables, literal data, reserved words, or any combination of these).
minmax	the range within which the number being loaded is to reside (for example, 3-12).

Example

Here are some comments about the following example:

- Part (a) displays the menu.
- Line (b) executes the LDV,N that produces a number between 1 and 20, which the run uses later on as the report number. This number is based on the data entered and remains the same as long as the LDV,N statement remains unchanged. Line (b) also checks to see whether the user is attempting to add or display an item.
- Part (c) displays an existing product whenever requested by the user. The number produced by LDV,N for the product name is always the same, and is used by the run as the report number in which to find the product.
- Part (d) finds a blank line to write and writes the line with the new product name, whenever requested by the user. The number produced by LDV,N for the product name is always the same, and is used as the report number to write the new product name into.

- Part (e) displays an error message when the user attempts to display a product that does not exist.

```
(a) @BRK LDV,W V10H1=SOE$ . MENU SECTION
(a) @1:CHG INVAR$ V4H9 .
(a)       ENTER
(a)  PRODUCT TO ADDV10□           , AND TRANSMIT□
(a)       OR
(a)  EXISTING PRODUCT
(a)       TO DISPLAYV10□          , AND TRANSMIT□
(a) @BRK OUT,-0,2,8,1,1,Y,,,P .
(b) @10:LDV,N V2012=V4,1-20 IF CURV$ = 2 GTO 2 .
(c) @FND,0,B,V20,,99 '' 'PRODUCT' □,V4 ,V515 .
(c) @DSP,0,B,V20,V5,,,,,\
(c)           ' PRESS F1 TO CONTINUE' .
(c) @GTO 1 .
(d) @2:FND,0,B,V20,,4 @ 'PRODUCT' □,@ ,V214 .
(d) @LOK,0,B,V20 .
(d) @3:WRL,0,B,V20,V2 'PRODUCT' □,V4 ULK GTO 5 .
(d) @4:LOK,0,B,V20 LN+,0,B,V20,5,15 LDV V214=6 GTO 3.
(d) @5:DSP,0,B,V20,V2,,,,,' \
(d)   PRODUCT ENTERED, PRESS F1 TO CONTINUE'
(d) @GTO 1 .
(e) @99:LDV,PU V4=V4 .     ERROR SECTION
(e)  PRODUCT V4 NOT FOUND, PLEASE TRY AGAIN.
(e) @CHG INVAR$ V4H9 BRK OUT,-0,2,1,,3,N .
(e) @GTO 10 .
```

P OPTION—PACKING A VARIABLE

With the P option, you can delete leading or trailing spaces from a variable. This is called *packing* a variable. Don't, however, pack a variable to contain no characters at all. If you do, other functions trying to use the variable will err.

Once you pack a variable to contain fewer than its original number of characters, you must reinitialize the variable to make it larger. If you try to place more than the original number of characters in a variable, you'll lose the extra characters.

LDV EXAMPLES WITH VARIOUS OPTIONS

The following statements are shown as if in sequential order in a run. The Δ stands for a blank character position.

This statement initializes V1 to 1:

```
@LDV V1I2=1 .
```

The following statement initializes V1 to A and V2 to 10:

```
@LDV V1A1=A,V2I2=10 .
```

This statement loads V1 with THIS IS DATAΔΔΔΔΔΔΔΔ:

```
@LDV V1S20='THIS IS DATA' .
```

This statement loads V1 with ΔΔΔΔΔΔΔΔΔ2ΔΔΔΔΔΔΔΔΔΔ:

```
@LDV,C V1=2 .
```

This statement loads V1 with 2:

```
@LDV,P V1=2 .
```

This statement loads V1 with ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ1:

```
@LDV,R V1S20=1 .
```

This statement loads V1 with 00000000000000000001:

```
@LDV,RZ V1=1 .
```

This statement loads V1 with ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ-1:

```
@LDV,R V1=-1 .
```

This statement loads V1 with -00000000000000000001:

```
@LDV,RZ V1=-1 .
```

This statement loads V1 with ΔΔΔ20000000000000001:

```
@LDV,R V1(1-4)=2 .
```

This statement loads V1 with 00020000000000000001:

```
@LDV,RZ V1(1-4)=2 .
```

This statement loads V1 with ABCΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ:

```
@LDV,U V1=abc .
```

In this statement, <MODE> contains the mode number, <TYPE> the numeric form type number, and <RID> the RID number of the last report or result processed or on display:

```
@LDV,W <MODE>I4=MODE$,<TYPE>I6=TYPE$,<RID>I4=RID$ .
```

In the following statement, if JDOE is on station 12, V4 contains ABCJDO12XYZ:

```
@LDV,PW V4S20='ABC'USER$(1-3)STNUM$XYZ .
```

The following statement uses an unknown trailing substring to load V2 with the minutes and seconds (MM:SS) substring from V1. The 0-5 specifies the last five characters, but the starting character position is unknown.

```
@LDV,W V1A8=TIME$ . V1 NOW CONTAINS HH:MM:SS
@LDV V2I5=V1(0-5) .  V2 NOW CONTAINS MM:SS
```

The following statement uses a known trailing substring to load V2 with the seconds (SS) substring from V1 (from the previous example). The 7-0 specifies the substring beginning with the seventh character through the end of V1.

```
@LDV V2I2=V1(7-0) . V2 NOW CONTAINS SS
```

LFC (Load Format Characters)

Use an LFC statement to capture the format of the report currently on display (the -0 RID).

An LFC statement works only in runs registered as format sensitive.

Format

@LFC *v* .

In field:	Enter:
v	the variable to capture the format of the report currently on display.

Example

In this statement, the variable receives a string of Xs and blank characters. The Xs stand for character positions displayed, similar to RID 0:

```
@LFC V1S80 .
```

You'll probably want to use an SFC statement after an LFC statement. See SFC for how to set format characters.

LFN (Load Field Name)

Use the LFN statement to load variables with the names of report fields that correspond to the column-character positions supplied. See Section 2 for a general description of named fields.

The LFN statement is especially useful for converting an existing run to one that uses named fields. It is also useful for translating column position data, such as that obtained from the OUM statement (see OUM), into field names.

Here are some things to remember:

- You can have field names enclosed in apostrophes; this makes it easier for you to create run statements.
- If a specified variable isn't large enough to contain an entire field name, the statement truncates any trailing characters in the name.
- If the specified columns don't represent an entire field, the statement loads the name followed by a partial field description.
- If the run cannot load a field name, it continues at the label.
- If the statement has no label and a field name cannot be loaded, the run continues at the next statement, loading the variable with column position data.

Format

@LFN[,m,t,r,tics?,lab] cc v[,v,...,v] .

In field:	Enter:
m,t,r	the mode, form type, and RID number of the report or result to load field names from. (Default = -0.) If you don't specify the *m,t,r* subfields, use only two commas to represent these subfields rather than three.
tics?	a Y to enclose the field name in apostrophes.
lab	the label or relative line number to go to if the field name cannot be loaded.
cc	the column-character positions of the fields.
v	the variables to load with field names.

Reserved Word

Reserved word: STAT1$ (error codes)	
Code	Error
1	Report header is improperly formatted for field names.
2	Columns supplied do not fall within field boundaries.
3	Field name in report header has no significant characters.
4	Field name is not unique in the report header.
5	Field name was truncated because of variable size, and name is not unique in report header.

Example

```
@LFN,0,B,2,Y 2-2,45-3 V1H18,V2H18 .
```

where:

0,B,2	Load field names from RID 2B in mode 0.
Y	Enclose field names in apostrophes.
2-2,45-3	Get names from column 2 for two characters and column 45 for three characters.
V1H18	Load V1 with the name from positions 2-2 (V1 = 'STCD').
V2H18	Load V2 with the name from positions 45-3 (V2 = 'CUSTCODE(1-3)').

LLN (Last Line Number)

Use an LLN statement to set a variable equal to the last line number of the specified report or result.

An LLN statement is especially useful for determining whether or not a report or form type exists.

Format

@LLN,m,t,r[,lab] vlines .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report or result in which to locate the last line.
lab	the label or relative line number to go to if no report (or result) or form type exists.
vlines	the variable to capture the number of lines in the report or result.

Reserved Words

Reserved words: STAT1$ and STAT2$	
Word	Content
If report exists. . .	
STAT1$	Date of last update in DATE1$ format (yymmdd)
STAT2$	Creation date of report in DATE1$ format (yymmdd)
If report has never been updated. . .	
STAT1$	0
If report does not exist. . .	
STAT1$	RID number of highest report
STAT2$	Disregard
If form type does not exist. . .	
STAT1$	0
STAT2$	Disregard
If neither the report nor form type exists, the LLN statement goes to the label in the lab subfield. If you don't specify a label, the run continues at the next statement.	

Example

This statement determines the number of lines in RID 1B, mode 0, and places that quantity in <LINES>. If the report or form type does not exist, the run goes to label 99:

```
@LLN,0,B,1,99 <LINES>I3 .
```

The LMG statement extracts lines or partial lines from a standard column-formed report (issuing report) and inserts them into the receiving report according to the following control character sequences within the receiving report. Note that each sequence begins with a tilde (~) character.

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

Format

@LMG,*im,it,ir,rm,rt,rr*[,*lab*] .

In field:	Enter:
im,it,ir	the mode, type, and RID number of the issuing report.
rm,rt,rr	the mode, type, and RID number of the receiving report.
lab	the label or relative line number to go to in case of an error.

Example

This statement merges lines and partial lines from RID 7H into RID 8H, both in mode 0:

```
@LMG,0,H,7,0,H,8 .
```

LNI (Line Insert)

The LNI statement inserts lines in a report or result. It duplicates the specified lines in the same report, but does not delete them from their original location. The report expands to make room for the new lines.

Unless you are processing a result, you must precede an LNI statement with a LOK statement. See LOK.

Format

@LNI,*m,t,r,lb4*[,*x*],*l*[,*q*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to insert lines.
lb4	the line number before the inserted lines. (The inserted lines follow this line.)
x	how many times to insert the lines. (Default = 1.)
l	the number of the first line to insert.
q	how many lines (quantity) to insert for each *x* starting at l. (Default = 1.)

Example

This statement inserts line 8 one time for two lines (lines 8 and 9) in mode 0, RID 3B, after line 6:

```
@LNI,0,B,3,6,1,8,2 .
```

LNK (Link to Another Run)

Use a LNK statement to execute another run in the MAPPER system. A LNK statement is like a RUN statement (see RUN), except that after the linked run executes a GTO END, the original run continues executing.

The system can transfer up to 40 input parameters, as well as the current result (-0), to the linked run. Pack or right-justify variables used to specify either the run name or other input parameters.

The linked run can return up to three status codes and the current result of the linked run to the original run via a GTO END statement (see GTO). Following the LNK statement in the original run, you can examine the reserved words STAT1$, STAT2$, and STAT3$ to obtain the status codes.

If the linked run terminates because it encounters a REL statement, because of an error, or because of manual intervention after a DSG, DSP, OUM, OUT, or SC statement, the original run does not resume executing.

You cannot nest LNK statements. Also, a run started by a LNK statement cannot itself contain another LNK statement unless it first clears the original link.

Use a CLK statement to terminate the link (see CLK).

Format

@LNK *run*[,*vld*] .

In field:	Enter:
run	the name of the other run to start.
vld	input parameters: variables, literal data, reserved words, or any combination of these, to send to the other run to be picked up via INPUT$. The maximum number of input parameters is 40; the maximum number of characters per parameter is 18. Pack or right-justify all variables.

Reserved Word

Reserved word: LINK$	
Word	Content
LINK$	0 if the run was not started by a LNK statement or non-zero if the run was started by a LNK statement.

Example

This example executes a run, TEST, with the contents of V1, V2, and SAM as input parameters to be picked up via INPUT$, and continues at the next statement when TEST executes a GTO END:

```
@LNK TEST,V1,V2,SAM .
```

This statement examines the status codes returned by the linked run:

```
@LDV,PW V116=STAT1$,V216=STAT2$,V316=STAT3$ .
```

LINKING TO THE SCHEDULE RUN

You can also use the LNK statement to call the SCHEDULE run from within a run. The SCHEDULE run schedules a registered background run for execution at a later time. See the Manual Functions Reference for more information on the SCHEDULE run.

Format

@LNK SCHEDULE,*run*,*extime*[,*exdate*,*sn*] .

In field:	Enter:
run	the name of the run to be scheduled. Note that this run must be registered for execution as a background run for the person executing your run.
extime	the time (in the format *hhmm*) the run is to be executed or NOW for the current time.
exdate	the date (in the format *yymmdd*) the run is to be executed. (Default = today's date.)
sn	the station number to be notified when execution of the SCHEDULE run is complete or N if no one is to be notified. (Default = user's station number.)

Reserved Word

Reserved word: STAT1$ (error codes)	
Code	Error
0	The background run has been scheduled.
1	The background run name specified is not a registered background run for the person executing your run.
2	The specified time is invalid.
3	The specified date is invalid.
4	The specified time and date have already passed.

Example

This example calls the SCHEDULE run to schedule the run called "myrun" at 1:30 P.M. on July 5, 1988. It notifies station 123 when the SCHEDULE run execution is complete.

```
@LNK SCHEDULE myrun,1330,880704,123 .
```

LINKING TO GRAPHICS RUNS

You can also use the LNK statement to call a chart run from within your run control report. You receive a result containing the primitive graphics code; the chart is not displayed. The primitive graphics code is then sent to the linking run via the -0 result. If you are linking to the MULTI run, see "Linking to the MULTI Run" in this subsection. See the Color Graphics Guide for more information on graphics runs.

Format

@LNK *chartrun,rt*[*,sn,psiz?,transpcy?*] .

In field:	Enter:
chartrun	the name of the chart run to link to (for example, PIEG or BARG).
rt	the RID number and form type of the chart input report (leave this subfield blank if you want the current -0 result).
sn	the station number of the plotter.
psiz?	a Y if you want to use 11 x 17 inch paper. (Default = N, 8 1/2 x 11 inches.)
transpcy?	a Y if you are making transparencies. (Default = N.)

Linking to the MULTI Run

If you are linking to the MULTI run, use this format:

@LNK MULTI,*q*[,*f*,*sn*,*psiz?*,*transpcy?*],*m*,*t*,*r*,*m*,*t*,*r*[,*m*,*t*,*r*,...] .

In field:	Enter:
q	the quantity (2-4) of the charts to place into one chart.
f	the format for the MULTI charts. Specify: formats 1, 2, or 3 for two charts, formats 1 or 2 for three charts, or no formats (leave blank) for four charts. Note that the *f* subfield is optional only if you specify four charts.
sn	the station number of the plotter.
psiz?	a Y if you want to use 11 x 17 inch paper. (Default = N, 8 1/2 x 11 inches.) Specify this subfield only if you specified a plotter station number in the *sn* subfield.
transpcy?	a Y if you are making transparencies. (Default = N.) Specify this subfield only if you specified a plotter station number in the *sn* subfield.
m,*t*,*r*	the mode, type, and RID number of the reports containing the primitive graphics code. You must specify at least two reports.

LNM (Line Move)

The LNM statement moves lines within a report or result. It deletes the lines from their original location and moves them to the new location.

Unless you are processing a result, you must precede an LNM statement with a LOK statement.

Format

@LNM,*m,t,r,lb4*[,*x*],*l*[,*q*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to move lines.
lb4	the line number before the moved line or lines. (The moved lines follow this line.)
x	how many times to move the lines. (Default = 1.)
l	the number of the first line to move.
q	how many lines (quantity) to move for each *x* starting at 1. (Default = 1.)

Example

This statement moves line 8 one time for two lines (lines 8 and 9) in mode 0, RID 3B, after line 6:

```
@LNM,0,B,3,6,1,8,2 .
```

LNX (Line Duplicate)

The LNX statement duplicates lines within a report or result. The report expands to make room for the new lines.

Unless you are processing a result, you must precede an LNX statement with a LOK statement.

Format

@LNX,*m,t,r,l,x*[,*q*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to duplicate lines.
l	the number of the first line to duplicate.
x	how many times to duplicate the lines. You must enter a number because *x* doesn't default to 1.
q	how many lines (quantity) to duplicate. (Default = 1.)

Examples

This statement duplicates line 12 five times in mode 0, RID 3B:

```
@LNX,0,B,3,12,5 .
```

The following example duplicates a three-line paragraph, starting at line 6 of RID 3B in mode 0, five times:

```
@LNX,0,B,3,6,5,3 .
```

LN+ (Line Add)

The LN+ statement adds lines to a report or result. The report expands to make room for the new lines. The system automatically inserts tab characters in the locations defined for the form type.

Unless you are processing a result, you must precede an LN+ statement with a LOK statement.

Format

@LN+,m,t,r,lb4,q[,predfl] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to add lines.
lb4	the line number before the added lines. (The added lines follow this line.)
q	how many lines (quantity) to add. You must enter a number because *q* doesn't default to 1. (Maximum of 999 lines.)
predfl	the reference number of the predefined line to add from RID 0 of the form type.

Example

This statement adds one type 2 predefined line to mode 0, RID 3B, after line 5:

```
@LN+,0,B,3,5,1,2 .
```

The LN- statement deletes lines from a report or result.

Unless you are processing a result, you must precede an LN- statement with a LOK statement.

Format

@LN-,*m,t,r,l,q* .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report from which to delete lines.
l	the number of the first line to delete.
q	how many lines (quantity) to delete. You must enter a number because *q* doesn't default to 1.

Example

This statement deletes one line from mode 0, RID 3B, starting at line 15:

```
@LN-,0,B,3,15,1 .
```

LOC (Locate)

The LOC statement locates a character string within a report or result, and with the O or OU option, creates a result.

Format

@LOC,m,t,r[,l,lab] o cc tgtstr vcol,vlno,vrid .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to locate a string of characters.
l	the line number at which to start the scan.
lab	the label or relative line number to go to if no target string is located.
o	options (use the A, F, and M options to locate strings regardless of the line type): A — Process all line types. (If used alone, it ignores the first character of the target string.) B*n* — Display the report *n* lines before the line where the find was made, where *n* is a one-digit number. Do not use the B option with the O option. C — Distinguish between uppercase and lowercase letters.

(continued)

(continued)

In field:	Enter:
F	Process all line types and locate the entire string. (Does not locate strings that start in column 1.)
M	Treat the first character of the target string as the line type designator. (Use the M option instead of a line-type subfield. Also, you must use the M option to locate a string beginning in column 1.)
O	Create a result that contains the items found. NOTE: With the O option, *vcol* contains 0, *vlno* contains the total number of lines that contain the target string (not the line number where the target string was located), and *vrid* contains the RID number where the target string was located.
OU	Create a result; then do one of the following: enter **del** to delete the lines in the result from the report (see DEL). enter **ext** to delete the lines in the result from the report and redisplay the result (see EXT). enter **upd** to make changes to the result and blend these lines into the report (see UPD). You cannot execute LOC with the OU option against a result.

(continued)

(continued)

In field:	Enter:
	S*x* Start scan at line *x*, where *x* is a positive number.
	S*x-y* Start scan at line *x* and stop at line *y*.
	S*x,n* Start scan at line *x* and scan *n* lines.
	T*x* Set *x* to a transparent character to match any character in that position. (Don't use the transparent character in the first character position of the target string. For example, the character string A$$D, where $ is the transparent character, locates all four-character strings, where A is the first character and D is the fourth character, such as ABCD, A 2D, and A%CD.)
cc	the column-character positions or names of the fields to scan.
tgtstr	the string of characters to locate.
vcol	the column number where the target string was located. This variable contains one less than the column number if you use the F option; add one to get the actual column number.
vlno	the line number where the target string was located.
vrid	the RID number where the target string was located.

Example

```
@LOC,0,B,2,,99 AFM 2-79 FED <COL>,<LINE> .
```

where:

0,B,2	Process RID 2B in mode 0.
99	Go to label 99 if no target string is located.
AFM	Use the A, F, and M options.
2-79	Start scan in column 2 for 79 characters.
FED	Locate the target string FED.
<COL>	Capture a number one less than the column number where the target string was located (leftmost column of report is column 0) in <COL>.
<LINE>	Capture the line number where the target string was located in <LINE>.

LOG (Accounting Log)

Use a LOG statement to log each function executed in a run.

Normally, whenever a run is executed, the system logs only one entry in the ***accounting log***, which is a summary of the data compiled while the run is executing.

A LOG statement, however, produces an entry in the accounting log for each function executed in the run.

Enter a LOG statement in the run control report ahead of all other statements (except :L statements, if your run has them). If the run contains a REL statement, replace it with a GTO END statement. This keeps the log list intact after the run completes.

When you're finished evaluating the log result, reinstate the REL statement and delete the GTO END statement (if you added one).

Format

@LOG .

HOW YOU CAN USE THE LOG LIST

You can evaluate the general quality of your run by studying the log result produced by the LOG statement. You can also submit your log result to the RUNA run (see Section 5), a run analyzer.

After executing your run, wait a couple of seconds, then resume to display the log list. If your run does not terminate normally because of a loop or if you have other problems and cannot get a log list, call your coordinator, who can obtain the log list for you.

Use a LOK statement to get update control of a report and to prevent other run users from updating your report until you release update control.

You don't need a LOK statement to update results.

Use a LOK statement before these statements:

LNI
LNM
LNX
LN+
LN-
WRL

These statements release update control:

ADR	GTO END
AUX	LOK
DEL	REP
DFU	SEN
DLR	SOR
DUP	ULK
EXT	UPD

NOTE: The MAPPER system releases update control whenever a run terminates—for any reason. The run does not terminate until the calling run and any runs called from within the run terminate.

If another user already has update control of the report and if the LOK statement in your run has no label, the run stalls until the other user releases update control; otherwise, the run continues at the label specified in the *lab* subfield.

LOK

Format

@LOK,*m,t,r*[,*lab*] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report in which to get an update lock.
lab	the label or relative line number to go to if another user has update control of the report.

Reserved Word

Reserved word: STAT1$	
Word	Content
STAT1$	Station number that has report locked

Example

This statement gets update control of mode 0, RID 6B:

```
@LOK,Ø,B,6 .
```

Use an LSM statement to load a variable with the contents of a MAPPER system message.

Messages can be up to 132 characters long, but usually take up one line of data. They are formatted so that columns 1 through 80 have the message and columns 82 through 88 have the message mnemonic.

Specify the number of characters of the message you want loaded in the variable size.

Format

@LSM,*msgno*[,*lab*] *vmsg* .

In field:	Enter:
msgno	the message number. If using an LSM statement in an error subroutine, capture this message number with the reserved word XERR$ (see RER).
lab	the label or relative line number to go to if the message number does not exist.
vmsg	the variable in which to place the message.

Example

This example loads V1 with the message number and V2 with the message:

```
@LDV,W V1I4=XERR$ LSM,V1 V2S80   .
```

LZR (Line Zero)

The LZR statement creates or loads variables with information from line 0 of the designated report or result.

The LZR statement does not create a result. Results (-0) existing before LZR executes continue to exist after LZR executes.

Format

@LZR,m,t,r[,lab vlines,vcpl,vhdrs,vcs,vupds,vdept,vuser,vrpw,vwpw] .

In field:	Enter:
m,t,r	the mode, type, and RID number of the report that contains the information.
lab	the label or relative line number to go to if no report (or result) or form type exists.

NOTES:

1. Use the *lab* subfield without any of these variables if you just want to find out whether the report or form type exists.
2. Use any of the following variables to capture the desired information.
3. If you specify a renamed result that is not previously renamed with an RNM statement, the run errs; it does not go to the specified label.

(continued)

(continued)

In field:	Enter:
vlines	the variable to capture the number of lines.
vcpl	the variable to capture the number of characters per line.
vhdrs	the variable to capture the number of protected header lines as defined in RID 0. The number of header lines always equals 1 for a result on display.
vcs	the variable to capture the number of the character set type: 0 = LCS, 1 = FCS, and 2 = FCSU.
vupds	the variable to capture the number of updates to the report since it was created.
	NOTE: The next four variables contain 0 if no password is set.
vdept	the variable to capture the department number if the report has a department-private read access lock.
vuser	the variable to capture the user-id if the report has a user-private read access lock.
vrpw	the variable to capture the read password, or the word LOCKED* if the report has a read password.
vwpw	the variable to capture the write password, or the word LOCKED* if the report has a write password.

* The word LOCKED is indicated unless access is by the key user sign-on accessible to coordinators.

Reserved Words

Reserved words: STAT1$, STAT2$, and STAT3$	
Word	Content
If report exists. . .	
STAT1$	Date of last update in DATE1$ format (yymmdd)
STAT2$	Creation date of report in DATE1$ format (yymmdd)
STAT3$	Save flag date, if one exists
If report has never been updated. . .	
STAT1$	0
If report does not exist. . .	
STAT1$	RID number of highest report
STAT2$	Disregard
If form type does not exist. . .	
STAT1$	0
STAT2$	Disregard
If neither the report nor form type exists, the LZR statement goes to the label in the lab subfield. If you don't specify a label, the run continues at the next statement.	

Example

In this statement, <LINES> captures the number of lines in RID 2B, mode 0, or the run goes to label 99 if no report or form type exists:

```
@LZR,0,B,2,99 <LINES>I5 .
```