WANG

Report Program Language (RPL) User Manual

Report Program Language (RPL) User Manual

Release 2, October 1977

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which this software package was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang Laboratories, Inc., or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the software package, the accompanying manual, or any related materials.

NOTICE:

All Wang Program Products are licensed to customers in accordance with the terms and conditions of the Wang Laboratories, Inc. Standard Program Products License; no ownership of Wang Software is transferred and any use beyond the terms of the aforesaid License, without the written authorization of Wang Laboratories, Inc., is prohibited.

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

HOW TO USE THIS MANUAL

This manual provides both an introduction to the Report Program Language (RPL) and detailed instructions for programming in RPL. Although this manual assumes a knowledge of Wang 2200 Systems and BASIC, and a passing acquaintance with Keyed File Access Method 3 (KFAM-3), it is written for the user who has relatively little programming experience. It is recommended that this entire manual be read through once before attempting to write or run an RPL program. After becoming familiar with the contents of the manual, the reader will find it useful as a reference manual when running and debugging RPL programs.

In this manual, the terms "RPL system" and "system" refer to the collection of programs supplied by Wang Laboratories on the RPL system disk, while the terms "RPL" and "RPL language" refer to the language itself. Chapter 1 elaborates on this distinction and lays the groundwork for the more system-specific information in the rest of the manual. Chapter 2 should be read by everyone, as it defines the syntax specification conventions which are used throughout the manual. Chapters 3, 4, and 6 contain basic RPL system information, while Chapter 5 describes the syntax and structure of the language itself.

> NOTE:
>
> The RPL system is supplied on one diskette (package number 195-0019-3 for the 2270 diskette or package number 195-0019-2 for the 2240 diskette). It must be run on a Wang 2200C-2 with Option 5 (Sort ROM), a 2200T (including the WCS/20, WCS/30, and PCS systems), or a 2200VP. The 2200 configuration must include at least 8K of main (user) memory, a dual disk or diskette unit, and a line printer. If the RPL system is to be run from a hard disk, it must first be transferred from the system diskette to the hard disk.
>
> While it is possible, for demonstration or testing purposes, to run the RPL system using only the one system diskette, for practical applications this is not feasible. For a more detailed discussion of this point, refer to Chapter 3.

TABLE OF CONTENTS

PART 1: THE RPL LANGUAGE AND SYSTEM

CHAPTER 1: OVERVIEW

1.1 WHY RPL?

The invention of computers has simplified life by automating many kinds of routine office work. As technology develops and computers become more sophisticated, an increasingly wider variety of computers becomes available for an increasingly wider variety of applications. When selecting a computer for a particular application, the individual is always faced with the critical question: Which computer is best suited for my application? Related considerations include how easily and effectively the work can be automated: how long it will take to automate it, how much of it can be automated, and how much time and money will have to be invested to maintain the hardware and software once the transition phase is complete.

Although computer hardware can be tailored specifically for a given application, usually it is better to build general-purpose hardware and add on software as the need arises. This is not only more practical in terms of manufacturing costs, but also enables consumers to use the same computer for more than one application. The Wang System 2200 is among these so-called "general-purpose" computers. As such, it can be adapted to a wide variety of applications by utilizing different software packages.

The System 2200 only accepts and executes programs coded in the 2200 version of BASIC. Although BASIC is a general-purpose language, useful for a variety of data processing tasks, coding sophisticated BASIC programs to perform these tasks requires a certain commitment to software development. One such common task is the generation of reports from data files. This is an especially time-consuming job if the reports to be generated are somewhat complex, or if a certain flexibility in the report format is desired.

The Report Program Language (RPL) has been developed to simplify report generation on the 2200. The RPL system consists of a number of program modules, all of which are coded in BASIC. When used as instructed in this manual, these modules accept code written in the Report Program Language and then transform that RPL code into BASIC code. This transformation is called "compiling". The BASIC code compiled by the RPL system can be run on the 2200 in conjunction with a specified data file or files to produce formatted reports.

In effect, then, the RPL software system allows the user to tailor-make reports to his/her needs easily and quickly. Since the user has to write RPL code (and not BASIC code), once the user has learned RPL, a wide variety of reports can be generated with relatively little expense of time and effort.

This manual provides both a description of the RPL language and operating instructions for setting up the RPL system. The entire manual should be read through once before attempting to write or run an RPL program. Section 1.2 contains basic information about the system: how it is supplied and on what 2200 hardware it can be run. Section 1.3 goes into an explanation of what some of the RPL program modules are and what they do. Chapter 2 defines the syntax specification conventions which are used throughout this manual. Chapter 3 provides instructions for the initial set-up of the system. Chapter 4 delineates precisely which kinds of data files can be used as input to an RPL program. Chapter 5 describes the syntax and structure of the language itself. Finally, Chapter 6 provides RPL system operating instructions: how to input the RPL program, compile it, and run it.

1.2 RPL SYSTEM SPECIFICATIONS

The RPL system is supplied on one diskette (package number 195-0019-3 for the 2270 diskette or package number 195-0019-2 for the 2240 diskette). The catalog index occupies the first 24 sectors (sectors 0 to 23), while the system occupies sectors 24 to 567, inclusive. As the system takes up roughly half of one diskette, it is not possible to run RPL on any sizable data base without using another diskette or a hard disk.

The RPL system must be run on a Wang 2200C-2 with Option 5 (Sort ROM), a 2200T (including the WCS/20 and the WCS/30), or a 2200VP. The 2200 configuration must include at least 8K of main (user) memory, a dual disk or diskette unit, and a line printer. If the RPL system is to be run from a hard disk, it must first be transferred from the system diskette to the hard disk.

1.3 THE FUNCTIONAL COMPONENTS OF RPL

As section 1.1 implies, there are three distinct phases in the creation of an RPL program: writing and keying in the program, compiling the program, and then running the compiled version on the 2200. Accordingly, the RPL system can be viewed as a collection of three types of program modules: those used to create and maintain RPL programs, those used to compile RPL programs, and those used when running the compiled versions. It is not important to know exactly which RPL program modules perform each of these functions, but it is essential to understand that these three functions must be performed serially for any given RPL program. For example, an RPL program can never be compiled until it is written and keyed in, nor can it be run until after it is compiled.

There must, then, be at least two versions of an RPL program: the user-written RPL program (referred to as the "source program" or "source code") and the compiled BASIC version of the program (referred to as the "object program" or "object code"). In fact, there are 4 or 5 separate disk files generated by the RPL system which, taken together, comprise one whole RPL program. These are:

1. <name> F1: The text of the RPL source program. It is stored on disk as a KFAM-3 user file.

2. <name> K1: The KFAM-3 key file associated with #1 (above). RPL statements are keyed by line number.

3. <name> : The start-up module of the RPL object program.

4. <name> R: The object module which actually prints the report specified by the source program.

5. <name> C: A special input procedure for SORT-3. This object module is generated only if COND and SORT statements are both included in the source program.

Example:

If a sample RPL program named TEST were created, and if it contained both a COND and a SORT statement, these files would be listed in the disk catalog after compiling:

```
TESTF1
TESTK1
TEST
TESTR
TESTC
```

Although at first blush this plethora of files appears formidable, it is not as complex as it may seem. #1 and #2 together comprise the RPL source program; #3, #4, and #5 (if included) comprise the BASIC object program which is generated. The object program may utilize some specialized RPL program modules (referred to as "system subroutines") which reside on the system disk. #3 is referred to as the "start-up" module because it does not contain all the object code itself, but rather utilizes or "calls" these system subroutines as the need arises. It also calls #4 (and #5) as appropriate.

Since Chapter 6 provides operating instructions for keying in RPL programs, compiling them, and running the object code, a thorough understanding of these five RPL program files is not mandatory. The purpose of this section is simply to delineate the roles of these three phases (keying in, compiling, and running) and to introduce the five files associated with each RPL program.

CHAPTER 2: SYNTAX SPECIFICATION CONVENTIONS

Below is a table of the syntax specification symbols used throughout this manual. These symbols are used most frequently in Chapter 5. If there is any confusion about the statement syntax presentations in Chapter 5, reading the examples underneath the individual syntax presentations should clarify the usage of these symbols.

Symbol	Name of Symbol	Meaning
<>	angle brackets or metalinguistic brackets	The enclosed is <u>not</u> to be taken literally; rather, an appropriate term is to be substituted.
{ }	curly brackets	Two or more choices are presented vertically. Choose only one.
[ ]	square brackets	The enclosed is optional, but may depend upon the circumstances. In certain cases, the enclosed may be mandatory. If there are such cases, they are discussed following the individual statement syntax presentations.
...	ellipsis	The immediately preceding term or clause may optionally be repeated until the program line is full. If repeated, the entire term or clause must be repeated.

Symbol	Name of Symbol	Meaning
∞	infinity	Mathematical infinity.
$-\infty$	negative infinity	Mathematical negative infinity.

CHAPTER 3: INITIAL INSTALLATION

3.1 STORAGE SPACE ALLOCATION

As mentioned in Chapter 1, there are three distinct phases involved in establishing an RPL program: creating (and updating) the program, compiling it, and running the corresponding object code. For any given RPL program, these three phases must be performed serially. An RPL program may be updated at any time after it is created; however, after updating it, it must be recompiled and then rerun to produce the desired report.

In accordance with these three phases, the RPL system can be viewed as a collection of three types of program modules: those used to create and update RPL source programs, those used to compile RPL programs, and those used to run the compiled versions. Further, the file requirements for each phase are slightly different. Although all the RPL system files usually reside on a mounted disk or diskette during all three phases, the input data file need only be mounted during the run phase. Similarly, while both the RPL source and object code files must be available during creation and compiling, the source code need not be available during the run phase. To summarize:

creating, updating, compiling	running
RPL system	RPL system (if SORT or KFAM-3 data file)
RPL source program	RPL object program
RPL object program	data file

The RPL system itself occupies somewhat over half of one diskette; therefore, to run RPL programs on any sizable data base, at least one more diskette is necessary. Furthermore, due to the difficulties involved in transferring RPL program files (particularly from a diskette to a hard disk or vice versa), it is advisable to devote some thought to storage mediums and space allocation before creating RPL programs. The only I/O peripherals the system supports are the CRT, line printer, minidiskette(s), diskette(s), and hard disk(s).

The first decision to be made regarding secondary memory space allocation is where to put the RPL system. Immediately after receiving the RPL system diskette, the user should write protect it and copy it to either another diskette or a hard disk. Note that the system (even without its work file) cannot be put on minidiskette since it would not fit. The system copy, which is the version actually used, must not be write protected. The system must always reside in its entirety on one disk or diskette; the program modules of the RPL system should not be split up and put on two or more disks with different device addresses. If the system is copied to another diskette, the MOVE instruction may be used. However, if the system is to be transferred to a hard disk, each program must be individually loaded into main memory and then stored on the hard disk. For a complete list of exactly which files are included on the system diskette, refer to Appendix D. The only system data file is WORKFILE; as it contains no permanent data, an empty file of at least 120 sectors with the name WORKFILE should be created. The work file need not reside on the same disk as the rest of the RPL system.

> NOTE:
>
> A clean copy of the RPL system diskette should be kept at all times.

The next question which arises with regard to space allocation is where to store the RPL program(s). Just before keying in the text of an RPL program, the user is prompted to enter an estimated number of source program statements. It is wise to make the estimate a generous one to allow for later insertions and modifications to the program. The current version of RPL does not provide any file manipulation utilities, so to expand the size of a program after it is created is an onerous task. In deciding where to store RPL programs, bear in mind that the source and object programs need not be stored together on the same disk (or diskette).

As the RPL system does not support tape cassette, the main requirement for the data file is that it reside on disk, diskette, or minidiskette. For a list of allowable device addresses, refer to Appendix C; for a discussion of allowable data file formats, refer to Chapter 4.

NOTE:

When deciding where to store the RPL system, an RPL program, and its corresponding input data file, bear in mind:

1. The source and object programs must be mounted simultaneously with the system during creation, updating, and compiling.

2. The object program and data file must be mounted simultaneously during the run phase. The system must also be mounted if the source program requested a SORT or if the input data file is a KFAM-3 data file.

3.2 SORT-3 PROGRAM

After allocating secondary memory space and transferring the RPL system from the supplied diskette to a viable location, it is usually necessary to modify the SORT3 module contained in the RPL system (unless RPL is accessed through the Integrated Support System (ISS)). If the main (user) memory on a particular system exceeds 8K, the SORT3 module should be modified to reflect this so that the sorting procedure will utilize all the available memory space and will thereby run faster. Also, it is not possible to sort a KFAM-3 data file unless there is at least 12K of main memory available. Any attempt to sort a KFAM-3 data file without modifying the SORT3 module will result in the error message NO ROOM TO SORT. This message will appear on the CRT screen when the object program is run. If a KFAM-3 data file (Format 2) is to be sorted in only 8K of main memory, it must be defined in the RPL program as a Format 0 data file (refer to Chapter 4).

To set the main memory size in the SORT3 module, change statement 3150 of the SORT3 program using the BASIC editing facilities. The variable "M" should equal the size of the main memory available; e.g., for a 16K system, M=16. After editing this one statement, scratch the SORT3 module on disk and save the newly modified version as SORT3.

3.3 RPL SYSTEM ADAPTATIONS

Section 3.1 discussed what considerations to take into account when allocating secondary memory space. Once disk space allocation is decided, this information must be made available to the RPL system. The chart on the next page indicates which lines of module RPL to change to reflect the particular software configuration.

SOFTWARE CONFIGURATION
(module RPL)

Device Addresses

Line #	Statement (as supplied on system diskette)	What resides at the device address	When the device address is used	Comments
100	SELECT DISK 310	RPL system	compile time	may not be overridden
110	SELECT #2 310	RPL system work file	compile time	may not be overridden
115	SELECT #4 310	RPL source program (KFAM-3 user file)	compile time	default device address - may be overridden prior to entering the RPL editor; must be the same device address as specified for source program key file
120	SELECT #5 310	RPL source program (KFAM-3 key file)	compile time	default device address - may be overridden prior to entering the RPL editor; must be the same device address as specified for source program user file
125	SELECT #6 310	RPL object program (all modules)	compile time	default device address - may be overridden prior to entering the RPL editor
135	F0$ = "310"	SORT-3 work file	run time	default device address - may be overridden by a device address in the RPL SORT statement
140	G1$ = "310"	RPL system	run time	default device address - may be overridden by the RPL SYSTEM statement
145	G0$ = "310"	RPL object program (all modules)	run time	default device address - may be overridden by the RPL RUN statement

File Names

Line #	Statement (as supplied on system diskette)	What is named	When it is used	Comments
105	D7$ = "WORKFILE"	RPL system work file	compile time	may not be overridden
130	F$ = "WORKFILE"	SORT-3 work file	run time	default file name - may be overridden by a file name in the RPL SORT statement

Note that there is no way to override the system variables used during compilation (system and work file device addresses and work file name). The exact reverse is true of all other variables: Their values may all be overridden. If the default values of all these variables contained in module RPL are used, this is termed "Standard Disk Usage" (SDU). Each individual RPL program is flagged as either using SDU or not using SDU. Before creating a new RPL program or editing an established one, the user is asked whether or not SDU is to be applied. If an RPL program does not use SDU, all necessary compile time variable values must be entered on the spot, and all necessary run time variable values must be supplied by appropriate statements in the RPL source program. So if SDU is not used, both a SYSTEM and a RUN statement must be included in the RPL program, along with any necessary parameters for sorting.

NOTE:

The three system variables used during compiling must be modified in module RPL to reflect the particular software configuration. All other variables in module RPL listed on the preceding chart need not be set to the correct values as long as Standard Disk Usage does not apply to the RPL program being run.

CHAPTER 4: DATA FILE FORMATS

There are four basic types of data files which can be used as input to an RPL program. These are delineated in the table which begins on the next page. The data file format must be defined in the RPL source program by specifying the appropriate format number in the FILE statement.

It is essential to know the exact layout of the input data file, as this layout must be defined in the RPL source program (refer to subsection 5.4.2). Each record must have the same format, with the exception of Format 1 and Format 3 header and trailer records. It is necessary to know:

1. The exact order of all the fields in the (each) record.

2. The type of variable in each field (character or numeric) and the length of each character field in bytes.

3. Whether the input data file is blocked or unblocked. If blocked, the records must be blocked as arrays (refer to subsection 5.4.2, BLOCK statement).

It should also be pointed out that if the records in the data file are blocked, each record may contain up to 38 fields; if the records are unblocked, the maximum number of fields allowed is 55.

Allowable Data File Formats

Format # (as specified in the FILE statement)	Description of Format
0	No header, end-of-file (trailer) record (the last record in the file must have been written with a DATASAVE END statement). The entire file must have been written in DC or DA mode, as BA mode is not supported.
1	Basic Accounting System 1 (BAS-1) format with standard BAS-1 header and trailer records. Multivolume files are supported if no SORT is specified in the RPL program; otherwise, only one volume is processed at a time (end-of-volume is treated as end-of-file).
2	Keyed File Access Method 3 (KFAM-3) file with a record type of A, N, or M. KFAM-3 record type C files cannot be used with RPL.
3	Integrated Support System (ISS) format: a) The file must be a cataloged file. b) The file name in the disk catalog must be the same as the file name in the header record. c) The first data sector of the file must be a software header record. The complete format of this header record is defined by the ISS OPEN routine. For RPL, the first field of the record must be a character string containing the value "HDR". The second field of the record must be a character string containing the file name. The remainder of the header record is ignored by RPL. d) Data records themselves may be either unblocked or blocked. They must all have identical formats. If they are blocked, they must be written in array form. BA mode is not supported.

Format # (as specified in the FILE statement)	Description of Format
3 (continued)	e) The end of data is recognized by the following conventions: If the first field in each data record is a character field, a value of HEX(FF) in the first byte of the field marks the end of data. If the first field in each data record is numeric, a value of 9E99 (exactly) marks the end of data.
	f) The sector after the record or block of records containing the end of data marker must contain a software trailer record. The format of this record is defined in the ISS CLOSE routine. RPL reads only the first field of this record, which must be a character field. A value of "EOF" signals an end-of-file condition. Any other value signals an end-of-volume situation, and a prompt to mount the next volume appears.
	g) Multivolume files can only be sorted one at a time.

CHAPTER 5: PROGRAMMING IN RPL

5.1 INTRODUCTION

To take full advantage of the many convenient and versatile features of RPL, it is essential to have a thorough grasp of both the semantics and the syntax of the language. Put more simply, to profit from RPL, you need to know both what RPL can do and how to go about doing it. To a certain extent these two topics are intertwined; a discussion of RPL syntax rules will inevitably involve some functional considerations. Nevertheless, in this chapter the emphasis will be on syntax: both the syntax rules for composing individual statements and the rules for combining groups of statements into cohesive programs.

Although the material covered in this chapter is fundamental to writing RPL programs, a word of caution is in order: writing a program and running the object code are two related but separate operations. To achieve the desired effect with an RPL program, the programmer must understand not only how to write the program, but also exactly what will happen when the corresponding object code is run. A discussion of keying in RPL programs, compiling them, and running them is left to Chapter 6. For a comprehensive view of an entire RPL program and the report it generated, refer to Appendix B.

5.2 STATEMENT TYPES AND PROGRAM SYNTAX

5.2.1 Statement Types

There are 16 fundamental RPL statements: SYSTEM, RUN, FILE, INDEX, RECORD, BLOCK, WORK, COND, SORT, KEYS, LINES, START, HEADER, DETAIL, TOTAL, and Comment. For ease of reference, these have been grouped into 4 basic statement types according to the function(s) they perform in the context of an RPL program. They are:

Statement Types

System Definition	Data Definition	Comment	Report
SYSTEM	FILE	* (Comment)	LINES
RUN	INDEX		START
	RECORD		HEADER
	BLOCK		DETAIL
	WORK		TOTAL
	COND		
	SORT		
	KEYS		

The Data Definition statements are further subdivided:

Data Definition Statements

Data File Definition	Data Selection and Sort
FILE	COND
INDEX	SORT
RECORD	KEYS
BLOCK	
WORK	

The Report statements are further subdivided:

Report Statements

Lines	Print
LINES	START
	HEADER
	DETAIL
	TOTAL

In addition, there is a group of substatements which are used in conjunction with the Report statements:

Substatements

Calculation	Program Flow
= (Equation)	GOTO
CONVERT	GOSUB
UNPACK	RETURN
ROUND	EXIT

All of the statement and substatement names taken together (SYSTEM, RUN, FILE,...EXIT) are referred to as the "command keywords". Each of the four basic statement types is dealt with individually in sections 5.3 to 5.6; section 5.7 contains a discussion of the substatements.

5.2.2 Program Syntax

Each program line consists of two elements: a line number followed by a statement. Line numbers specify the order in which the statements are to be executed; they must be integers and may range from 1 to 2000, inclusive. As the RPL system does not provide a RENUMBER command, the line numbers assigned when the program is initially written should be incremented by tens to allow for possible insertions later. Each program line may be up to 62 keystrokes in length. (As RPL was not designed for an 80 character CRT screen, the editing features are slightly more inconvenient on this wider screen. When using this screen, it is generally good to keep program lines to 61 keystrokes or less.) The coding is free form; blanks may be inserted anywhere between operands in a program line for readability, and will be ignored by the system unless they are enclosed in quotation marks.

An RPL program must contain a minimum of 3 statements: one FILE statement, at least one RECORD statement, and at least one Print statement. Other statements, such as the System Definition statements, INDEX statement, and BLOCK statement may also be necessary. The primary rule for combining statements into programs is that all fundamental statements be grouped together when the program is displayed in line number order. All RECORD statements must appear together, all COND statements together, etc. The only exception to this rule is the Comment statement, which is used only for internal documentation of the program. Since Comment statements are ignored by the RPL compiler, they may appear anywhere in the source code.

For the sake of program readability, it is recommended that variables not be referenced before they are defined. For example, do not SORT a file before the file is defined by the appropriate Data Definition statements. In order to facilitate comprehensibleness and so insure flexibility of RPL programs, it is advised that RPL programmers adopt the convention of ordering statement groups in the sequence they appear in this chapter, from SYSTEM, RUN, FILE, INDEX, etc. up to TOTAL. Appendix A contains a summary of this recommended order, which is illustrated in the sample program in Appendix B.

Lastly, it should be noted that a specific ordering scheme is sometimes mandatory within one group. For example, RECORD statements must define fields in the exact order that they appear on the data file records. These statement-specific rules are discussed following the individual statement syntax presentations in the upcoming sections.

Example:

Grouping fundamental statements:

RIGHT

```
10  FILE 0
20  RECORD AGE = #
30  HEADER 1: DATE
40  HEADER 1: PAGE
      .
      .
      .
      .
      .
```

WRONG

```
10  FILE 0
20  HEADER 1: DATE
30  RECORD AGE = #
40  HEADER 1: PAGE
      .
      .
      .
      .
      .
```

NOTE:

In the examples presented in the rest of this chapter, line numbers are omitted wherever possible for the sake of clarity. It is assumed that if the example statements were incorporated into actual programs, appropriate line numbers would be assigned to them. The example statements do not represent parts of any one complete program.

5.3 SYSTEM DEFINITION STATEMENTS

```
SYSTEM <device address>
```

This statement defines the device address of the RPL system disk (refer to Appendix C for a list of allowable device addresses).

The SYSTEM statement is not used if Standard Disk Usage was specified the last time the source program was edited (refer to Chapter 3 for an explanation of Standard Disk Usage). If SDU was not specified, a SYSTEM statement must be included, but the device address specified will be ignored unless the RPL program contains a SORT statement or the input data file is a KFAM-3 file. If SDU was specified and a SYSTEM statement is included, an error message will be printed when the program is compiled.

Example:

```
SYSTEM 310
```

```
RUN <device address>
```

This statement defines the device address of the disk which will contain the object program (refer to Appendix C for a list of allowable device addresses).

The RUN statement is not used if Standard Disk Usage was specified when the source program was last edited. If SDU was specified and a RUN statement is included, an error message will be printed when the program is compiled.

Example:

```
RUN B10
```

5.4 DATA DEFINITION STATEMENTS

5.4.1 Naming Conventions

When assigning field names with the RECORD and WORK statements, certain naming conventions must be observed. If an invalid name is assigned, one of two outcomes will result: either the compiler will generate an error message when it compiles the program, or it will compile the object program incorrectly. Since no error message will be generated in this latter case, when the object program does not run as expected, the source of the trouble may be difficult to locate. Therefore, it is recommended that care be taken to adhere strictly to the naming conventions.

The most important rule to follow when selecting field names concerns reserved names, also known as "reserved keywords". The RPL system has "reserved" certain words for its own use. Since the system cannot differentiate between two identical names, the RPL programmer cannot use a reserved keyword as a field name. These reserved keywords fall into two categories: command keywords and "special" keywords. The command keywords are the words which define the RPL statements and substatements, such as SYSTEM, FILE, START, CONVERT, and GOTO. The special keywords are DATE, PAGE, LINE, EXPAND, and KEY. These keywords are used in conjunction with the Print statements and are discussed in section 5.6. A complete list of reserved keywords can be found in Appendix C.

In addition to this restriction, field names assigned by the programmer must be from 1 to 6 characters in length and must consist only of upper case letters (A to Z). Within one RPL program, each field name must be a unique combination of letters.

5.4.2 Data File Definition Statements

```
FILE <format> [: ["] <file name> ["]: <device address>]
```

This statement defines the format of the file from which data is to be extracted (refer to Chapter 4 for a discussion of allowable data file formats). One and only one FILE statement must be included in every RPL program.

The file name and device address may be omitted, in which case the system will prompt for this information before running the object program. If either the file name or the device address is omitted, both must be omitted. The file name, if included, must be the same as the data file name listed in the disk catalog (unless the file is a Basic Accounting System 1 (BAS-1) file, in which case the file name specified must be the file name contained in the header record). The file name must be enclosed in quotation marks if it contains embedded special characters (e.g., blanks); otherwise, the quotation marks are optional. For a list of allowable device addresses, refer to Appendix C.

Examples:

```
FILE 0

FILE 0: ATEST: B10

FILE 0: "ATEST": B10

FILE 0: "A TEST": B10
```

```
INDEX <KFAM-3 key file number> [: <device address>]
```

If the data file is defined in the FILE statement as a KFAM-3 user file, an INDEX statement must be included to indicate which KFAM-3 key file to use when extracting data. Records will be read in ascending sequence using this key file (FINDFIRST/FINDNEXT sequence). Unless a SORT statement is used, records will also be printed in this order.

The device address may be omitted, in which case the system will prompt for it before running the object program. For a list of allowable device addresses, refer to Appendix C.

Examples:

```
INDEX 1

INDEX 1: B10
```

```
RECORD <field name 1> = { #<field length>
                        { <field name 2>, <start>, <bytes> }
    [ : <field name 3> = { #<field length>
                        { <field name 4>, <start>, <bytes> } ] ...
```

This statement defines the record layout of the file from which data is to be extracted. At least one RECORD statement must be included in every RPL program.

Every record of the data file must have the same format, with the exception of Format 1 and Format 3 header and trailer records (refer to Chapter 4 for a discussion of allowable data file formats). Every field in the data file must be defined; the fields must be defined in the exact order that they appear in the records. If the records are blocked into 2 or more records per sector, the maximum number of fields that can be defined with the RECORD statement is 38. If the records are not blocked (or the blocking factor is 1) the maximum number is 55. The total number of field names defined in all RECORD and WORK statements cannot exceed 100.

More than one field can be defined in one RECORD statement by separating the definitions with colons. A single definition may not be split between consecutive RECORD statements. It is recommended that the field definition precede all references to it in any other RPL statements (e.g., DETAIL, COND, or KEYS).

All field names assigned in the RECORD statement (field name 1, field name 3, etc.) must conform to the naming conventions summarized in Appendix C. If the field is a numeric field, it should be defined with a number sign (#); if it is a character field, its length in bytes should be specified (one character occupies one byte). The length in bytes must be a positive integer less than 65. (If a character field length is not explicitly defined in BASIC with a DIM statement, its length defaults to 16.) Partial character fields may be defined by entering a character field name previously defined by a RECORD statement (field name 2, field name 4, etc.), the numerical position of the first character of the partial field, and its length in bytes. This is analogous to the STR function in BASIC. The numerical position of the first character combined with the length must define a valid proper or improper subset of the original field. Partial fields may not be defined on other partial fields.

Examples:

```
RECORD SEX = 1

RECORD AGE = #

RECORD SEX = 1: AGE = #: ACCT = #: LETTER = 1
```

Defining a partial field:

```
RECORD NAME = 30
RECORD INIT = NAME, 1, 1

RECORD NAME = 30: INIT = NAME, 1, 1

RECORD NAME = 30
RECORD INIT = NAME, 1, 1: AGE = #: SEX = 1

RECORD NAME = 30
RECORD AGE = #
RECORD SEX = 1
RECORD INIT = NAME, 1, 1
```

```
RECORD NAME = 30: AGE = #
RECORD INIT = NAME, 1, 1: SEX = 1
```

In each of the five examples above, the partial field INIT would contain the first letter of NAME.

BLOCK <factor>

This statement is used only if the input data file contains more than one record per disk sector. If a BLOCK statement is not included, no blocking (record type N or M in KFAM-3) is assumed; i.e., it is assumed that either the data file contains exactly one record per disk sector or that it contains records which each occupy some fixed number of sectors.

The BLOCK statement specifies the number of records per sector. This number, or factor, must be an integer from 1 to 255, inclusive. Numbers from 127 to 255 will not generate RPL compiler error messages; however, these numbers may not be used as blocking factors with KFAM-3 data files, as KFAM-3 does not allow multiple-sector blocked records. A blocking factor of 1 indicates that there is exactly one record per disk sector (record type N in KFAM-3); it is never necessary to include a BLOCK statement if the blocking factor is 1.

The records must be blocked as arrays (record type A in KFAM-3). The first variable in each record would be contained in the first array, the second variable in each record would be contained in the second array, etc. Each block, or sector, must have the same format and must contain exactly the same number of records.

Example:

Suppose there were 12 records in the data file TEST, each containing a 20 character name, a numeric age, and an 8 character questionnaire response. The variables could be initialized with the BASIC statement:

```
DIM A$(12)20, B(12), C$(12)8
```

After entering the data into these arrays, they could be written out to an existing disk file like this:

```
DATA SAVE DC CLOSE ALL
DATA LOAD DC OPEN R "TEST"
FOR N = 1 TO 12
     DATA SAVE DC A$(N), B(N), C$(N)
NEXT N
DATA SAVE DC END
DATA SAVE DC CLOSE ALL
```

If the records were written with the above statements, they would be unblocked. The data records alone would occupy 12 sectors, and there would be one record per sector. They could be defined with these RPL statements:

```
FILE 0: TEST: B10
RECORD NAME = 20: AGE = #: RESP = 8
```

Alternatively, the records could have been blocked. Suppose the records had been blocked into 6 records per sector. Then the BASIC code to initialize the variables might be:

```
DIM A1$(6)20, A2$(6)20, B1(6), B2(6), C1$(6), C2$(6)
```

After entering the data into these arrays, they would be written out to disk:

```
DATA SAVE DC CLOSE ALL
DATA LOAD DC OPEN R "TEST"
DATA SAVE DC A1$(), B1(), C1$()
DATA SAVE DC A2$(), B2(), C2$()
DATA SAVE DC END
DATA SAVE DC CLOSE ALL
```

In this blocked form, the data records alone would occupy only 2 sectors, and there would be 6 records per sector. They could be defined with these RPL statements:

```
FILE 0: TEST: B10
RECORD NAME = 20: AGE = #: RESP = 8
BLOCK 6
```

```
WORK <field name 1> [(<elements>)] = { #<field length>
                                     { <field name 2>, <start>, <bytes> }

[ : <field name 3> [(<elements>)] = { #<field length>
                                    { <field name 4>, <start>, <bytes> } ] ...
```

This statement defines the work fields, if any, to be used by the RPL program. Work fields are reserved variables to be used later on for computational purposes. Work fields are not fields in the input data file; fields in the data file are defined by the RECORD statement.

The only other difference between the WORK statement and the RECORD statement is that work fields may be defined as arrays, while data file fields may not. To define an array work field, include the number of elements in parentheses after the field name. The number of elements may range from 1 to 255, inclusive. Arrays may not be defined as partial fields, nor may any array name be used in defining a partial field.

Any character field defined by a WORK statement is automatically initialized to blanks; any numeric field is automatically initialized to zero. Partial character fields defined by a WORK statement must refer only to previously defined WORK character fields (not RECORD character fields).

Examples:

```
WORK SEX = 1

WORK AGE = #

WORK SEX = 1: AGE = #: ACCT = #: LETTER = 1
```

Defining a partial field:

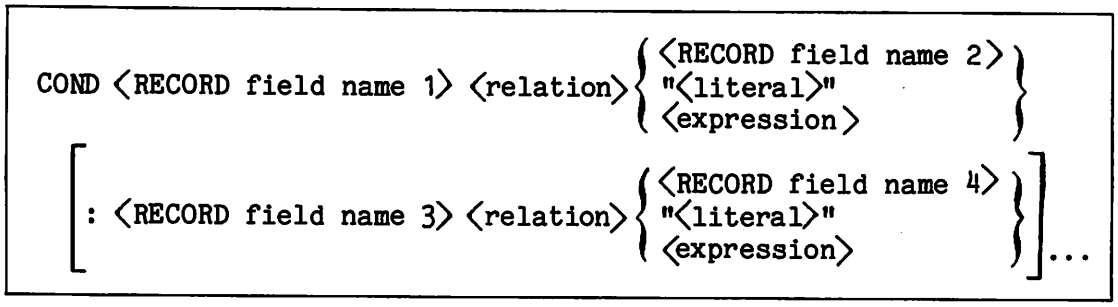
```
WORK NAME = 30
WORK INIT = NAME, 1, 1

WORK NAME = 30: INIT = NAME, 1, 1

WORK NAME = 30
WORK INIT = NAME, 1, 1: AGE = #: SEX = 1

WORK NAME = 30
WORK AGE = #
WORK SEX = 1
WORK INIT = NAME, 1, 1

WORK NAME = 30: AGE = #
WORK INIT = NAME, 1, 1: SEX = 1
```

In each of the five examples above, the partial field INIT would contain the first letter of NAME.

Defining an array of ten 30-character fields:

```
WORK ARRAY(10) = 30
```

5.4.3 Data Selection and Sort Statements



This optional statement selects only those records which meet certain conditions for inclusion in the report. If more than one condition is specified in one COND statement, records are selected only if all the specified conditions are true (logical AND). If more than one COND statement is included, records are selected if they meet the criteria of any one COND statement (logical OR). The maximum number of COND statements allowed in one RPL program is 39.

All of the field names specified as the first operand (field name 1, field name 3, etc.) must be valid field names defined in a RECORD statement. The relations that may be specified are:

SYMBOL	MEANING
=	first operand is identical (equal) to second operand
<>	first operand is not identical (equal) to second operand
<	first operand is less than second operand
<=	first operand is less than or equal to second operand
>	first operand is greater than second operand
>=	first operand is greater than or equal to second operand

The second operand may be a valid RECORD field name, a literal enclosed in quotation marks, or an expression. If the first operand is a character field name, the second operand must define a character field; likewise, if the first operand is a numeric field name, the second operand must define a numeric field. Consequently, literals are only allowed if the first operand is a character field name, while expressions are only allowed if the first operand is a numeric field name. Character fields are left justified for comparisons, while numeric fields are right justified.

Expressions may not contain parentheses; they may consist of any combination of numbers and(or) numeric field names separated by the operators +, -, *, and(or) /. Only the first number in the expression may be a negative number; all other numbers must be nonnegative. Operator precedence in expressions is the same as in BASIC. Expressions are evaluated from left to right in two passes. Multiplication (*) and division (/) are computed in the first pass, while addition (+) and subtraction (-) are computed in the second.

Examples:

Suppose these fields were defined:

```
RECORD NAME = 10: CITY = 16: STATE = 16
RECORD AGE = #: AUTOS = #: TWO = #
```

Suppose that the data file contains the names, cities, states, ages, and number of automobiles belonging to all members of an automobile club. In addition, suppose each record ends with a numeric field which always contains the value "2". This statement selects for printing only those members who live in Boston:

```
COND CITY = "Boston"
```

This statement selects all members who do not live in Massachusetts:

```
COND STATE <> "Massachusetts"
```

Each of these statements select all members who own two or more cars:

```
COND AUTOS >= 2

COND AUTOS >= 1+4/4*1

COND AUTOS >= TWO
```

This statement selects all members whose names begin with B to Z:

```
COND NAME>= "B"
```

These statements select all members who are under 25 OR own two or more cars:

```
COND AGE <25
COND AUTOS>= 2
```

While this statement selects all members who are under 25 AND own 2 or more cars:

```
COND AGE <25: AUTOS >= 2
```

To compound the COND statements further, we could ask for all members under 25 who own two or more cars OR all members who live in Boston:

```
COND AGE <25: AUTOS >= 2
COND CITY = "Boston"
```

```
SORT [<device address>: ["] <file name> ["]]
```

This optional statement causes the records of the data file to be sorted by the field(s) specified in the KEYS statement before printing the report. No more than one SORT statement may be included in an RPL program.

The sorting procedure must have a cataloged disk file to use as a scratch work file. The device address and file name are optional; if omitted, the system work file (120 sectors) will be used. If either the device address or the file name is omitted, both must be omitted.

As the RPL system will use either a program or a data file as a SORT work file, great care should be taken when specifying a work file to avoid overwriting a valuable file. The file name, if specified, must be the same as a cataloged disk file on a disk accessible to the system when the RPL program is run. Quotation marks are required only if there are embedded blanks in the file name; otherwise, the quotation marks are optional. Refer to Appendix C for a list of allowable device addresses.

In most cases, the required size of the SORT work file will not exceed the size of the input data file. The number of sectors required for the work file can be estimated as follows:

```
  K = total number of bytes in the SORT key field(s) (not including con-
      trol bytes)
  R = total number of records
  W = required work file space in sectors
INT = truncated to an integer value

    W = 20 + R/INT(250/(K+5))
```

This figure should be accurate to within 10% of the actual work file size required. If the allotted work file is inadequate, an error message will appear on the CRT screen when the object program is run. For a more exact calculation of the work file size, the SORT-3 program itself may be used. For details, consult the Integrated Support System User Manual, Release 2, Chapter 33.

If the input data file is a KFAM-3 user file, records will automatically be printed in ascending key sequence (using the KFAM-3 key file designated by the INDEX statement) unless a SORT is specified. Note also that if the input file is a KFAM-3 user file, the SORT3 module must be modified. For further details, refer to Chapter 3.

Examples:

```
SORT

SORT B10: SCRATCH

SORT B10: "SORT FIL"
```

```
KEYS <RECORD field name 1> [,D] [: <RECORD field name 2> [,D]] ...
```

This statement defines which fields are key fields. The key fields function in two ways: firstly, they define which fields are to be sorted if a SORT is included; secondly, they determine when and in what sequence the TOTAL statements are executed. When used in conjunction with a SORT statement, the KEYS statement defines the primary, secondary, tertiary, etc. SORT fields. Fields are sorted in ascending order (A to Z, $-\infty$ to ∞) unless "D" is specified, in which case the preceding key field is sorted in descending order (Z to A, ∞ to $-\infty$). When the KEYS statement is used in conjunction with a group of TOTAL statements, whenever the value of a key field changes the corresponding TOTAL statement is executed. The position of the operand in the KEYS statement defines the control level of the TOTAL statement. For further information, consult the TOTAL syntax presentation and discussion in section 5.6.

The KEYS statement is always used in conjunction with a SORT statement and/or a group of TOTAL statements. The KEYS statement is not required unless there is a SORT statement or a TOTAL statement of level 1 or greater. The KEYS statement may be continued by following it with another KEYS statement on the next line; however, the number of key fields cannot exceed 9, and the length of all the key fields together cannot exceed 64 bytes. The fields defined as key fields must be valid fields defined in a RECORD statement.

Example:

Suppose these fields were defined:

```
RECORD NAME = 10: CITY = 16: STATE = 16
RECORD AGE  = #: AUTOS = #
```

Suppose that the data file contains the names, cities, states, ages, and number of automobiles belonging to all members of an automobile club. Suppose further that it is necessary to print a report containing members' names and number of automobiles. The report should be sorted primarily by state and secondarily by name. The listing for each state is to begin on a separate page, and at the end of each state listing we would like to print the total number of automobiles owned by club members who are state residents. Finally, a grand total of all automobiles belonging to club members is to be printed at the end of the report. These RPL statements would accomplish this:

```
SORT
KEYS STATE: NAME
TOTAL 1: +1: 1: "TOTAL AUTOS FOR THIS STATE IS: ": 40: &AUTOS
TOTAL 1: +PAGE
TOTAL 0: +PAGE: 1: "GRAND TOTAL OF AUTOS IS: ": 40: &AUTOS
```

5.5 COMMENT STATEMENT

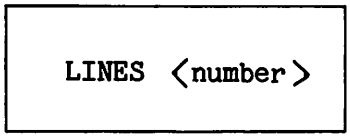
The Comment statement is used only for internal documentation of the program. Since no object code is compiled for Comment statements, they may appear anywhere in the source code; they need not all be grouped together.

Examples:

```
* This is a comment here.

***** So is this.

**********************************************************************
** No object code will be compiled for these statements.  **
**********************************************************************
```

5.6 REPORT STATEMENTS

5.6.1 LINES Statement



This optional statement specifies the maximum number of lines to be printed on a report page. Blank lines are included in this figure. If a LINES statement is not included, the maximum number of lines defaults to 56 (numbered 1 to 56).

The number specified in the LINES statement must be a positive integer. It must be greater than the number of lines required to print the page header plus one detail line, if we consider one detail line to be one execution of the DETAIL group. If the number of lines specified is less than this minimum number, the HEADER plus one DETAIL line will be printed per report page. A maximum number of 999 lines per page may be specified. Pagination never occurs while executing a print group; consequently, the LINES number specifies the last line on which a single execution of a print group may start. Unless otherwise specified in the TOTAL statement, TOTAL lines are printed on the same page as the preceding data (which they total), whether or not the maximum number of lines has been exceeded. The maximum number of lines should not exceed the page length encoded on the paper tape in the printer, or erratic paginations will result. The system automatically ejects a page after finishing the entire report, unless the last Print statement executed ends with a colon (:).

Examples:

If a minimum number of 8 lines per page was necessary (in order to print the heading plus one detail line), any one of these statements would be allowed:

```
LINES   8

LINES  25

LINES 999
```

5.6.2 Print Statements

The Print statements control all the calculating, formatting, and printing necessary to generate the report. At least one Print statement must be included in every RPL program; additional Print statements are optional. A group of one type of Print statement as it appears in an RPL program (START, HEADER, DETAIL, or TOTAL) is referred to as a "Print group".

Each of the Print statements has two syntactic formats: the print format and the substatement format. The two formats may not be mixed in any one statement. The print format syntax is presented in this section; a discussion of the substatement format is left to section 5.7.

Although the individual statement syntax presentations may seem complex, the basic function of the Print statements is simple: to do necessary calculations and then format and print the report. Since the calculations are all performed by using substatements, this section only concerns itself with how the Print statements format and print the report. To do this, each statement must contain three types of information: where to print, what to print, and how to print.

How to print is the first question to resolve. Report lines may be printed in either regular or expanded print, but the two cannot be mixed in one report line. If the option to EXPAND is not specified in a Print statement, the corresponding report line will be printed in regular sized print. There is a standard format for printing; however, some Print statements may specify the format in greater detail by including print images very similar to those constructed for the PRINTUSING statement in BASIC.

The next two questions are where and what to print. Before executing a Print statement, the RPL system normally executes a line feed and carriage return on the printer, so that it is ready to begin printing a new line of the report. Some Print statements may also optionally specify pagination or additional line feeds. Next, the Print statement contains pairs of column positions followed by elements to be printed. For each report line, the column positions should be specified in ascending order, and should be calculated to avoid overlap of print elements on the report.

Print elements may be valid RECORD or WORK field names or special keywords. There are restrictions in the usage of print elements; these restrictions have been incorporated into the individual statement syntax presentations. The special keywords that may be used in conjunction with the Print statements are:

DATE	Report date entered when the object code is run (mm/dd/yy).
PAGE	Current report page number. This variable is initially set to 1 on the first page printed with a header and automatically incremented by 1 each time system pagination occurs.
LINE	Current line number of the current page of the report. It is initially set to one at the start of each page and automatically incremented as report lines are printed. Skipped (blank) lines are included in this figure.

EXPAND	Specifies that the report line is to be printed in expanded print. EXPAND is ignored unless the report is generated on a matrix printer. If specified, EXPAND should be the first print element on the report line. Expanded print characters occupy twice the horizontal space of regular sized characters. The value of EXPAND is always HEX(0E).
KEY	May only be used in TOTAL statements. For further details, consult the TOTAL statement definition.

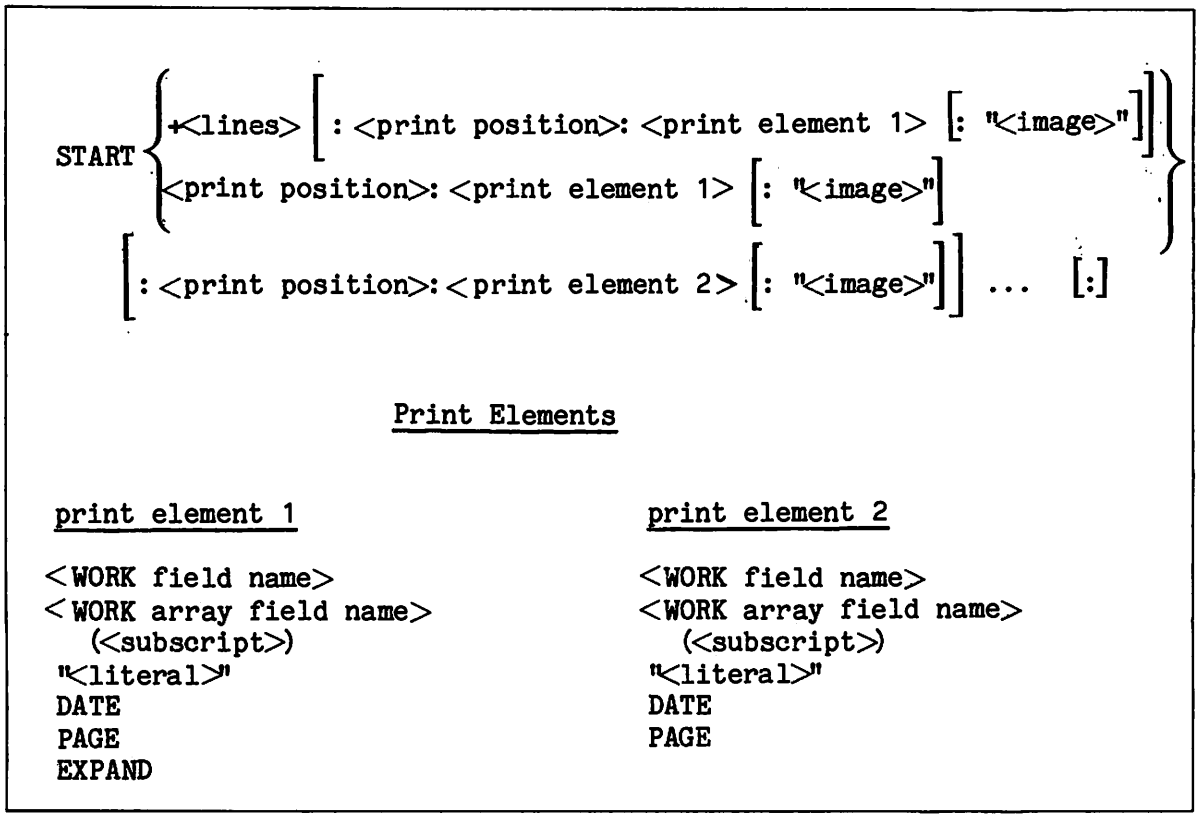
Unless otherwise noted in the individual statement syntax presentations, these variables should be restricted to these values:

RECORD field name	Valid field name previously defined in a RECORD statement.
WORK field name	Valid field name previously defined in a WORK statement.
WORK array field name (subscript)	The array field name should be a valid array field name previously defined in a WORK statement. The subscript should be a valid array subscript and can be specified either as a number or by using a valid RECORD or WORK numeric field name.
lines	Positive integer from 1 to 20, inclusive.
print position	Positive integer from 1 to 132, inclusive (regular print) or 1 to 66, inclusive (expanded print). Positions for one report line should be specified in ascending order.
literal	Non-null character string.
image	Valid PRINTUSING image without the percent sign (%).

It is a good idea to plan the report format carefully, adhering to the above restrictions. Failure to do so will have unpredictable results, ranging from compiler error messages to report format difficulties. In some cases, although no compiler error message will be generated, print elements will be dropped from lines and so not included when the report is printed. The report should be structured so that print elements do not overlap and do not run off the end of the printed line. If one whole print element does not fit into the allocated space at the end of the report line, it will usually be printed in its entirety on the next report line beginning in print position 1. Numeric fields are automatically padded on the left with one blank (two regular sized blanks if the line is expanded).

The RPL system automatically signals a carriage return/line feed at the end of each Print statement unless the statement ends with a colon (:). In this case, the system treats the next statement as a continuation of the previous statement, and continues to print on the same line on the printer. If nothing else is printed - that is, if the last Print statement executed ends with a colon - the print elements in the last Print statement will be put in the printer buffer but will not be printed by the RPL program. A print position and its subsequent print element must appear together in the same statement line; they may never be split between consecutive Print statements. Note also that if a Print statement ends with a colon, the following Print statement should not specify +PAGE, +<lines>, or EXPAND.

As the Print statements are the ones which actually structure and generate the report, understanding the syntax of these statements is not nearly as crucial as understanding their semantic content. When writing Print statements, the RPL programmer must constantly bear in mind the effects of running the corresponding object code. The primary thing to remember is that for each execution of the DETAIL group, the system will simply read a record from the input data file, format it, and print it on the report. For convenience, a summary of the order in which the Print groups are executed is presented in the flowchart in Appendix E. It may also be helpful to refer to the sample program and output found in Appendix B.



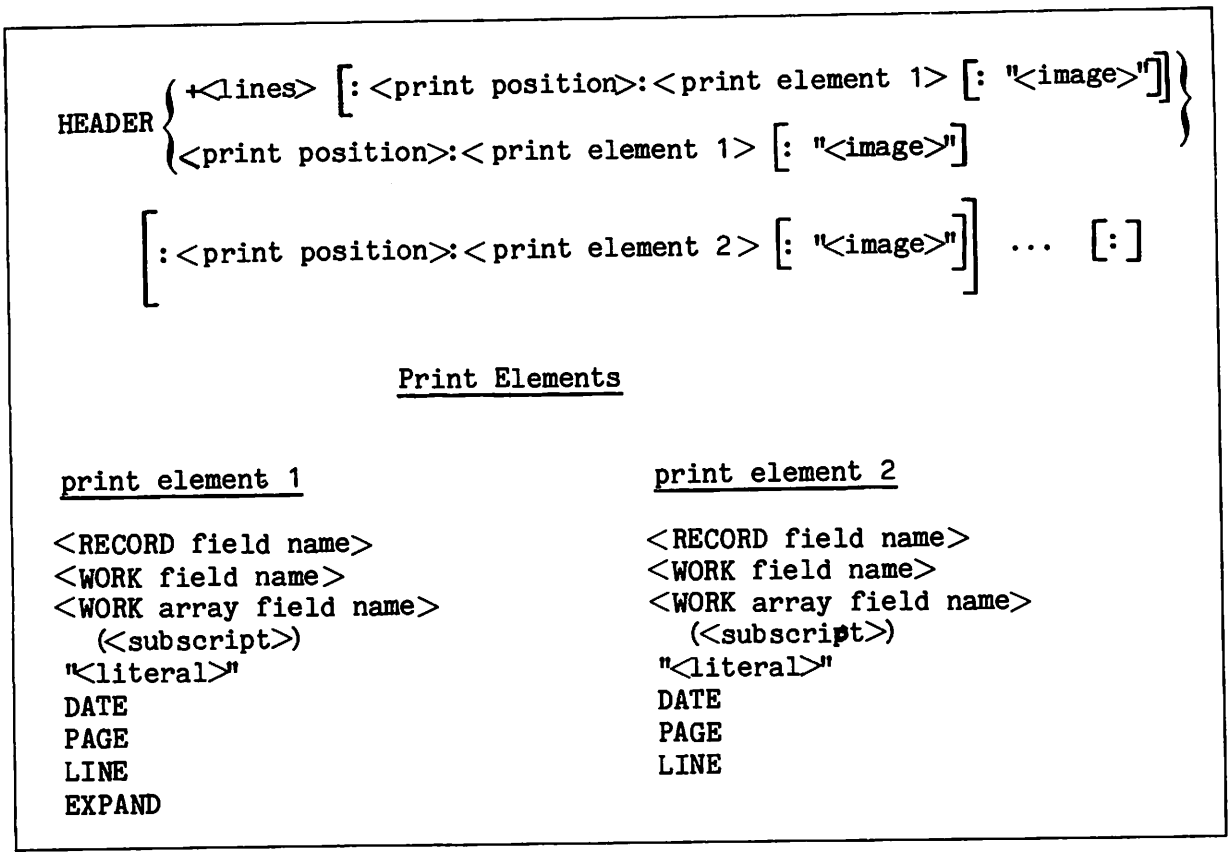
Print Elements

print element 1	print element 2
<WORK field name>	<WORK field name>
<WORK array field name> (<subscript>)	<WORK array field name> (<subscript>)
"<literal>"	"<literal>"
DATE	DATE
PAGE	PAGE
EXPAND	

The START group is executed only once, at the beginning of the report. The print format (above) is used for printing a title page. The title page is considered PAGE zero; it is the only report page printed without a HEADER. The system automatically ejects a page on the printer after executing the entire START group unless the last START statement executed ends with a colon (:). If there is no START group (or the START group does not print anything), the first page of the report is skipped. The next page begins with a HEADER and is considered PAGE one.

Note that the LINES statement does not apply to the title page; no pagination (other than automatic printer page ejects) takes place during the execution of the START group.

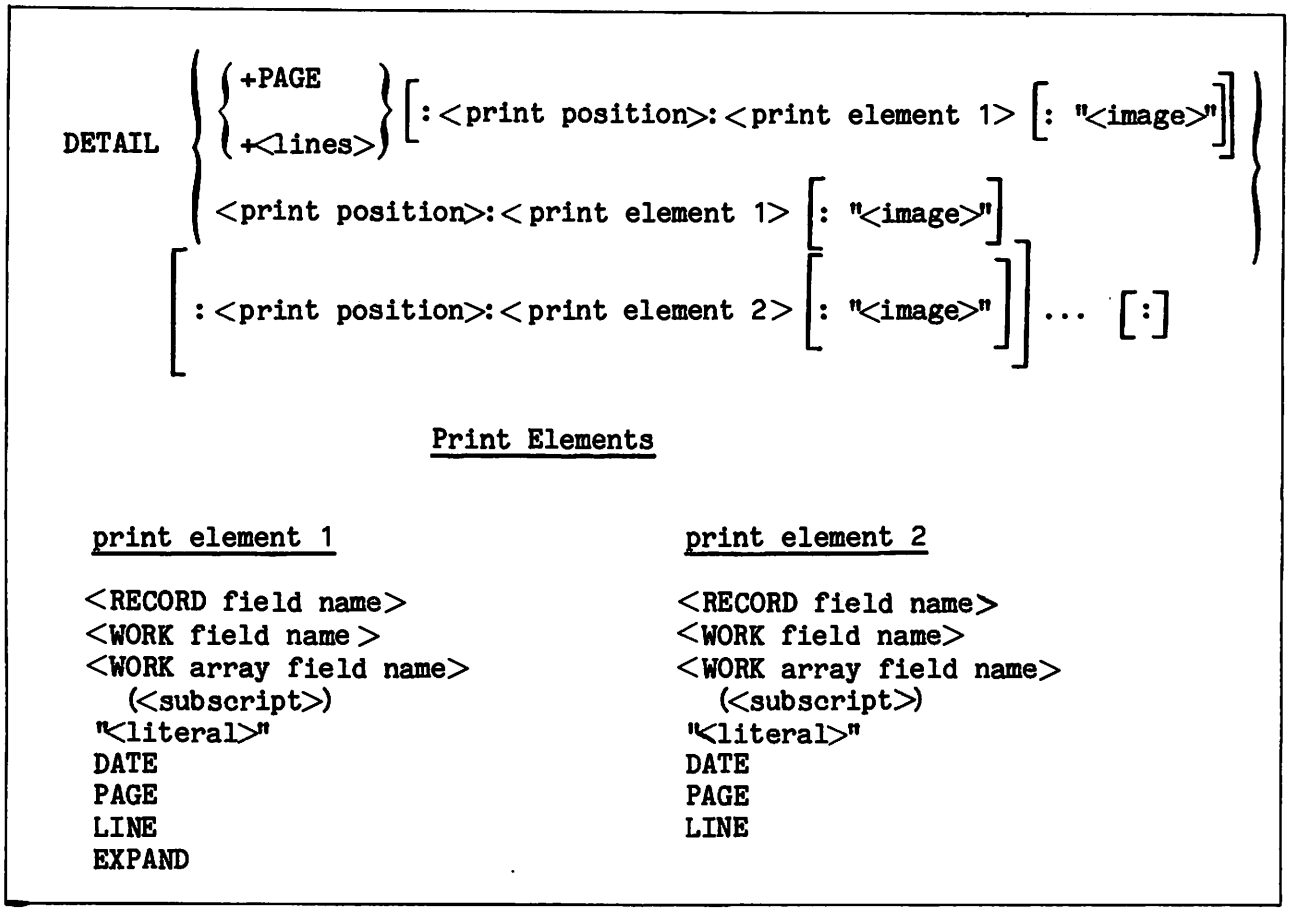
For examples, refer to Appendix B.



The HEADER group is executed immediately after each system page eject except the final page eject at the end of the report. It does not matter whether the page eject is caused by exceeding the maximum number of report lines per page (LINES or default) or whether it is triggered by a "+PAGE" in a Print statement. The HEADER group is always executed after the START group and before the first DETAIL statement is processed.

Note that if a RECORD field is printed by a HEADER statement, the first record of the report page will be used; i.e., the record printed by the next DETAIL statement.

For examples, refer to Appendix B.

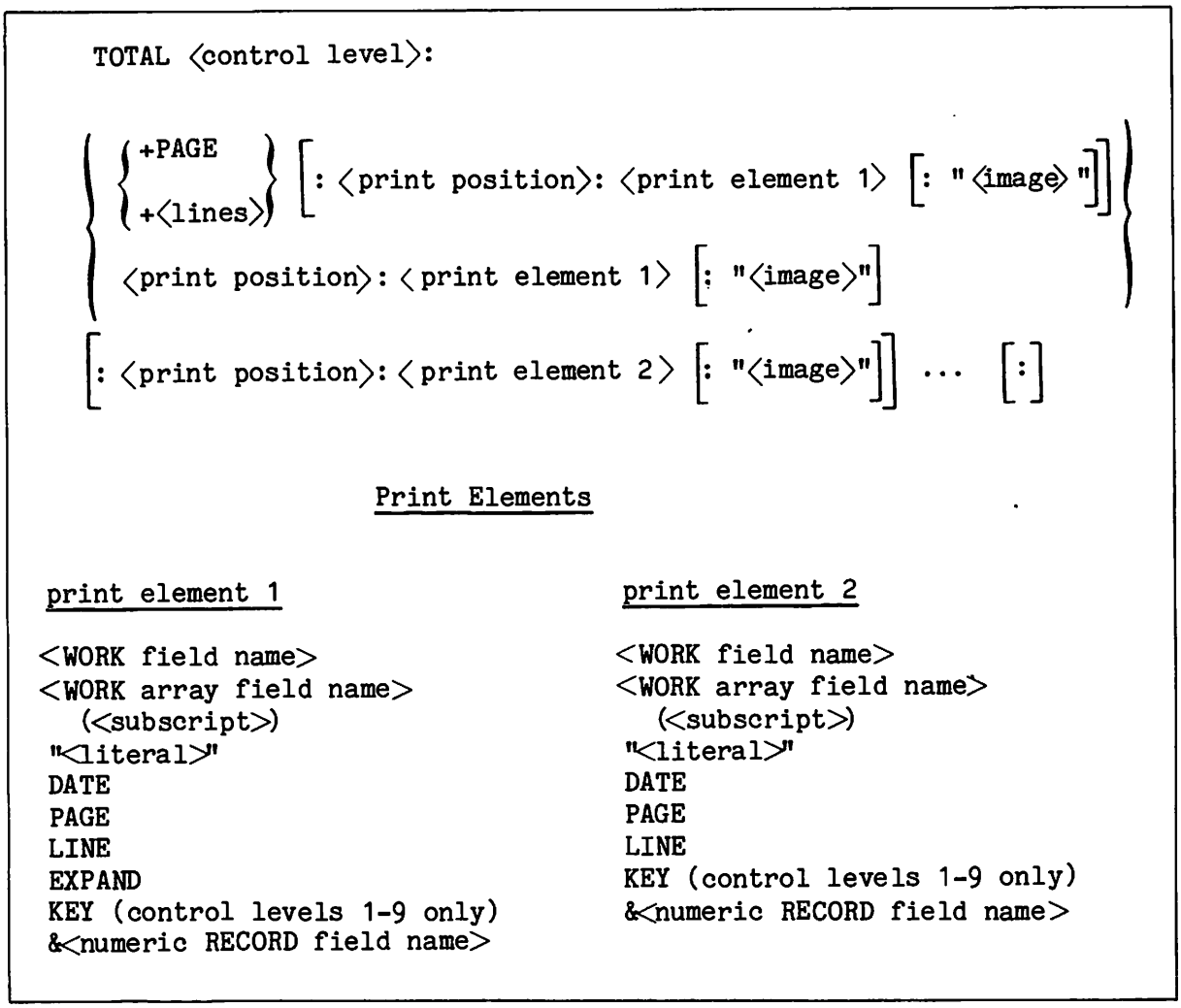


Print Elements

print element 1	print element 2
<RECORD field name>	<RECORD field name>
<WORK field name>	<WORK field name>
<WORK array field name> (<subscript>)	<WORK array field name> (<subscript>)
"<literal>"	"<literal>"
DATE	DATE
PAGE	PAGE
LINE	LINE
EXPAND	

The DETAIL group is executed once for each record in the data file. If a COND group is included, the DETAIL group is executed only for those records which satisfy the selection criteria.

For examples, refer to Appendix B.



The TOTAL <control level> group is executed once each time the value of the control field at the specified control level changes. The control level must be an integer from 0 to 9, inclusive. The control fields are the key fields defined by the KEYS statement. They may be either character or numeric fields. The first field in the KEYS statement corresponds to control level 1, the second field corresponds to control level 2, etc. up to control level 9. Control level 0 corresponds to the overall (grand) total; consequently, there is no field in the KEYS statement which corresponds to it. Within the TOTAL group, statements for each control level must be grouped together; it is recommended that the TOTAL <control level> groups be arranged in descending order, from control level 9 to control level 0. It is not necessary to have a TOTAL group for every (or any) key field.

Whenever a TOTAL <control level> group is executed, all TOTAL groups with numerically greater control levels are executed first. For example, if a level 4 TOTAL group must be executed, levels 9, 8, 7, 6, and 5 would be executed first, in that order. If two or more control fields change simultaneously, the corresponding TOTAL <control level> groups are also executed in descending order, from level 9 to level 0. When the data in the data file has been exhausted, all the specified levels are automatically executed.

The basic function of the TOTAL statement is to set control breaks and perform automatic totaling of numeric fields. A numeric RECORD field name preceded by an ampersand (&) is used to designate the total of the field at the specified level. Including this print element in a TOTAL statement serves to both totalize and print the total of the desired field. Numeric RECORD fields may be totaled at any or all of the specified levels; the number of automatic totals requested must not exceed 30, counting 1 for each total at each level. The variable KEY always contains the value of the control field at the same level as the TOTAL statement; since there is no control field for level 0, KEY may not be included in any TOTAL 0 statements. Although the TOTAL ⟨control level⟩ group is only executed when the value of a control field changes, the value of KEY at the time the TOTAL ⟨control level⟩ group is executed always equals the prior value of the control field, not the next value. The RECORD fields at the time the TOTAL ⟨control level⟩ group is executed contain the values in the record that caused the control break.

For examples, refer to Appendix B.

5.7 SUBSTATEMENTS

If we were to write an RPL program using only the information covered so far in this chapter, we could generate a minimally satisfactory report from a given data file. The file could be sorted, only those records which met certain conditions extracted, any or all fields of the records printed in any appropriate format, and numeric fields optionally totalized. However, we would be very limited in other respects. There would be no way at all of using WORK fields, no way of specifying arithmetic calculations, and no way of executing statements in a nonlinear fashion. Accordingly, substatements which closely resemble BASIC statements have been provided for use in conjunction with the Print statements. The syntax for the substatement format of the Print statements is given below.

```
START
HEADER          <substatement> [: <substatement>] ...
DETAIL
TOTAL <level> :
```

There are two main types of substatements: Calculation substatements and Program Flow substatements. These are discussed in subsections 5.7.1 and 5.7.2, respectively.

5.7.1 Calculation Substatements

```
<name 1> = { <name 2>
           { "<literal>"
           { <expression>
```

name 1 and name 2

START	HEADER or DETAIL	TOTAL <control level>:
<WORK field name>	<RECORD field name>	<WORK field name>
<WORK array field name> (<subscript>)	<WORK field name>	<WORK array field name> (<subscript>)
DATE	<WORK array field name> (<subscript>)	DATE
PAGE	DATE	PAGE
	PAGE	LINE
	LINE	KEY (control levels 1-9 only)
		&<numeric RECORD field name>

The equation substatement assigns the value of the second operand to the field specified as the first operand. This substatement is the RPL equivalent of the LET statement in BASIC.

If the first operand is a character field name, the second operand must define a character field; likewise, if the first operand is a numeric field name, the second operand must define a numeric field. Consequently, literals are only allowed if the first operand is a character field name, while expressions are only allowed if the first operand is a numeric field name. Expressions must conform to the same restrictions as COND statement expressions; they are evaluated in the same way (refer to subsection 5.4.3 for details).

Note that the names allowed in the equation substatement depend upon which Print statement is used, and also that some system variables (DATE, PAGE, LINE, and KEY) may be used as operands. DATE is an 8 character variable, while PAGE and LINE are numeric variables. If the value of LINE is changed by an equation substatement, subsequent paginations will be affected. The LINE variable is incremented after printing each line, so that if the LINE number is changed it will not be incremented until after the next report line is printed.

Examples:

In order to back date a report it might be desirable to set the DATE field (assuming that for some reason we decided against back dating the report at run time). Any one of these four statements would accomplish this:

```
START DATE = "1/1/77"

HEADER DATE = "1/1/77"

DETAIL DATE = "1/1/77"

TOTAL 0: DATE = "1/1/77"
```

If the DATE is printed before setting it with the equation substatement, the date entered when the object program is run will be printed.

CONVERT <character field name> / <numeric field name>

character field name	numeric field name
START statement	START statement
<character WORK field name> <character WORK array field name> (<subscript>)	<numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE
HEADER or DETAIL statement	HEADER or DETAIL statement
<character RECORD field name> <character WORK field name> <character WORK array field name> (<subscript>)	<numeric RECORD field name> <numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE LINE
TOTAL <control level>: statement	TOTAL <control level>: statement
<character WORK field name> <character WORK array field name> (<subscript>) KEY (must be a character key field - control levels 1-9 only)	<numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE LINE KEY (must be a numeric key field - control levels 1-9 only) &<numeric RECORD field name>

The CONVERT substatement converts the first operand from a character field to a numeric field and stores the result in the field designated by the second operand. This substatement is the RPL analog of the CONVERT statement in BASIC. Note that the slash (/) replaces the word "TO" (in BASIC), and also that expressions and images are not permitted in the RPL CONVERT.

The character field to be converted must contain a numeric value.

Example:

Suppose a transaction file contained, among other things, an amount (AMOUNT) and an 8 character transaction date (TRDATE) in each record. Suppose further that the file contained no transactions except those for the current month and the previous 11 months. To totalize the amounts for each month of the previous year, we could do it like this:

```
     .
     .
     .
RECORD AMOUNT = #:  TRDATE = 8:  MONTH = TRDATE, 1, 2
WORK MTOTAL (12) = #:  SUB = #
     .
     .
     .
DETAIL CONVERT MONTH/SUB
DETAIL MTOTAL(SUB) = MTOTAL(SUB) + AMOUNT
     .
     .
     .
```

Note that this would only calculate the total amount for each month in the WORK array MTOTAL; to print the results would require another Print statement. Another way of accomplishing the same thing would be to SORT the data file by MONTH and then to do automatic totaling of the AMOUNT field with the TOTAL statement. If this option was taken, the results of the totalizations would automatically be printed.

UNPACK (<image>) <character field name> / <numeric field name>

character field name	numeric field name
START statement	START statement
<character WORK field name> <character WORK array field name> (<subscript>)	<numeric WORK field name> <numeric WORK array field name> (<subscript>)
HEADER or DETAIL statement	HEADER or DETAIL statement
<character RECORD field name> <character WORK field name> <character WORK array field name> (<subscript>)	<numeric RECORD field name> <numeric WORK field name> <numeric WORK array field name> (<subscript>)
TOTAL <control level>: statement	TOTAL <control level>: statement
<character WORK field name> <character WORK array field name> (<subscript>) KEY (must be a character key field - control levels 1-9 only)	<numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE LINE KEY (must be a numeric key field - control levels 1-9 only) &<numeric RECORD field name>

The UNPACK statement unpacks the first operand and stores the result in the field designated by the second operand. This substatement is the RPL analog of the BASIC UNPACK. Note that the slash (/) replaces the word "TO" (in BASIC), and also that arrays are not permitted in the RPL UNPACK.

The character field to be unpacked must have been packed with a BASIC PACK statement. The image specified must be a valid image as used in the BASIC UNPACK statement. As with the BASIC PACK/UNPACK statements, all numbers are assumed to be positive unless the first character of the image is a plus (+) or a minus (-) sign, in which case all numbers retain their original sign.

Example:

Suppose that a packed field (AMOUNT) in the input data file was to be unpacked before doing calculations and printing. Suppose we also knew that AMOUNT was a monetary value which never exceeded a positive value of $9,999.99. It could be unpacked as follows:

```
WORK UNAMT = #
DETAIL UNPACK (####.##) AMOUNT/UNAMT
```

ROUND <name>, <number>

name

START	HEADER or DETAIL	TOTAL <control level>:
<numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE	<numeric RECORD field name> <numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE LINE	<numeric WORK field name> <numeric WORK array field name> (<subscript>) PAGE LINE KEY (must be a numeric key field - control levels 1-9 only) &<numeric RECORD field name>

The ROUND substatement rounds a numeric field to a specified number of decimal places and stores the result back in the same field. The number in the ROUND statement specifies the number of places to retain to the right of the decimal point: It must be an integer between 0 and 9, inclusive. There is no BASIC corollary of the ROUND substatement (although there is one in BASIC-2). A digit of 5 is always rounded up.

Example:

Suppose a numeric WORK field, INT, was used to compute the amount of interest to be charged to each client. After computing the interest in the DETAIL group, it could be rounded to the nearest cent:

```
DETAIL ROUND INT, 2
```

Thus, if the interest were equal to, say, 2.065, it would be rounded off to 2.07.

5.7.2 Program Flow Substatements

The Program Flow substatements allow the RPL programmer to execute Print statements in a nonlinear fashion. While the Program Flow substatements make RPL a more powerful and flexible language, they should naturally be used judiciously to avoid "spaghetti bowl" programming. The RPL rule of thumb is this: Any statement executed from a given Print group is treated as a part of that Print group, and must therefore adhere to its coding rules with regard to which variables may be referenced. Further details may be found in the discussions of the GOTO and GOSUB substatements. When writing Program Flow substatements it may be particularly useful to refer to the flowchart in Appendix E.

BASIC code with line numbers of 8000 to 9999 may be executed from RPL. If BASIC code is used as a subroutine, it should return with the BASIC RETURN statement. When branching back from a BASIC program segment executed via an RPL GOTO, 4000 must be added to the RPL line number to get the correct line number in the RPL object program. For details on executing BASIC subroutines from RPL, refer to Chapter 6.

The object code variable names assigned to fields in the RPL source program are in the range A0-A9, A0$-A9$, B0-B9, B0$-B9$, ..., L0-L9, L0$-L9$. After compiling an RPL program, the compiler will print a list of variables in this range which have been assigned (Object Program Variable Listing). These assigned variables may be referenced in BASIC code executed from RPL; in addition, unassigned variables in this range may be used. Variables outside this range should not be used.

GOTO <line number> [IF <name 1><relation> {<name 2> / "<literal>" / <expression>}]

name 1 and name 2

START	HEADER or DETAIL	TOTAL <control level>:
<WORK field name>	<RECORD field name>	<WORK field name>
<WORK array field name> (<subscript>)	<WORK field name>	<WORK array field name> (<subscript>)
DATE	<WORK array field name> (<subscript>)	DATE
PAGE	DATE	PAGE
	PAGE	LINE
	LINE	KEY (control levels 1-9 only)
		&<numeric RECORD field name>

The GOTO substatement is used to execute a branch or a conditional branch to another line number. This substatement is the RPL equivalent of the BASIC GOTO. The line number specified must be either another line number in the same Print group or a number from 8000 to 9999 (to access BASIC code). If a GOTO substatement specifies a line number in the same Print group and that Print group has more than 255 statement lines, the compiler error message "UNRESOLVED REFERENCE TO LINE <number>" may occur. If the line number is a valid one, this error message should be ignored; the GOTO will be compiled correctly.

If the first operand is a character field name, the second operand must define a character field; likewise, if the first operand is a numeric field name, the second operand must define a numeric field. Consequently, literals are only allowed if the first operand is a character field name, while expressions are only allowed if the first operand is a numeric field name. The relations and expressions specified must conform to the same restrictions as in the COND statement. Expressions are also evaluated in the same way as in the COND statement (refer to subsection 5.4.3 for details).

Example:

Suppose a field (MONEY) contained a monetary amount always less than or equal to $999,999.99. In addition, suppose we would like to print the amount if it is positive; otherwise, we want to print the absolute value of the amount followed by "CR". These statements would do the trick:

```
500 DETAIL GOTO 530 IF MONEY < 0
510 DETAIL 85: MONEY: "###,###.##"
520 DETAIL EXIT
530 DETAIL MONEY = 0-MONEY
540 DETAIL 85: MONEY: "###,###.##CR"
```

GOSUB <line number>

The GOSUB substatement is used to execute subroutines. It functions like the BASIC GOSUB. There are three types of subroutines which can be executed by using the GOSUB statement: RPL subroutines in the same Print group, RPL subroutines in another Print group, and BASIC subroutines (lines 8000-9999). All of these subroutines must end with a RETURN in order to branch back to the RPL statement following the GOSUB.

The only difference between subroutines executed within the same Print group and subroutines in other Print groups lies in which fields may be referenced in the subroutine itself. For example, if there were a subroutine in the HEADER group, it could be executed indiscriminately from anywhere else in the HEADER group by a HEADER GOSUB statement. If, however, the same subroutine were executed from the TOTAL 0 group, it would be legal only if the subroutine made no reference to RECORD fields. Although technically all variables may be referenced in all subroutines regardless of the calling Print group, variables not legal in the calling Print group will contain unpredictable values. The variables which may be referenced by each Print group are:

START	HEADER or DETAIL	TOTAL <control level>:
<WORK field name>	<RECORD field name>	<WORK field name>
<WORK array field name> (<subscript>)	<WORK field name>	<WORK array field name> (<subscript>)
DATE	<WORK array field name> (<subscript>)	DATE
PAGE	DATE	PAGE
	PAGE	LINE
	LINE	KEY (control levels 1-9 only)
		&<numeric RECORD field name>

These rules with regard to allowable field names in subroutines also apply to BASIC subroutines.

A GOSUB substatement which calls a subroutine in another Print group will generate a compiler error message, "UNRESOLVED REFERENCE TO LINE <number>". The same error message may occur within one Print group with more than 255 statement lines. In both cases, this error message should be ignored; the subroutine call will be compiled correctly. Lastly, it should be emphasized that all subroutines must be out of line of execution. A strategically placed EXIT substatement is often the most logical way to keep subroutines out of line of execution.

For examples, refer to Appendix B.

RETURN

The RETURN substatement is used to return from an RPL subroutine. It is analogous to the BASIC RETURN.

For examples, refer to Appendix B.

EXIT

The EXIT substatement is used to terminate execution of the Print group. Although execution is ended automatically at the last line of the Print group, in situations involving branching (GOTO or GOSUB) the EXIT substatement is useful to terminate execution of a procedure. The EXIT substatement may appear in any line of the Print group. It has no corollary in BASIC.

Examples:

Suppose a field (MONEY) contained a monetary amount always less than or equal to $999,999.99. In addition, suppose we want to print the field only if its value is nonzero. This could be accomplished in at least two ways:

```
500 DETAIL GOTO 520 IF MONEY = 0
510 DETAIL 85: MONEY: "###,###.##"
520 DETAIL EXIT
```

```
500 DETAIL GOTO 510 IF MONEY <>0: EXIT
510 DETAIL 85: MONEY: "###,###.##"
```

CHAPTER 6: RPL SYSTEM OPERATING INSTRUCTIONS

There are three main steps in establishing any commercial software program: writing the program, debugging it, and putting it into production. The initial writing of the program is the only step which does not involve any physical interaction with the computer. After the program is at least partially written, it must somehow be entered into the computer's memory, debugged, and used. During both debugging and production the program usually has to be run and modified several times: during debugging to correct bugs, and during production to both correct bugs and update the program's original function. Hence, although an RPL program is only created once, it is subsequently modified an arbitrary number of times. The method for creating a new RPL program is very similar to that employed for modifying an old one; in both cases, the RPL editor is used.

Generally speaking, each time the RPL editor is used to create or alter a particular RPL source program, the edited program is then recompiled and rerun to monitor the effects of the source code changes. So for any given RPL program, the programmer can expect to go through the three phases of editing, compiling, and running an arbitrary number of times.

6.1 THE RPL EDITOR

As mentioned in Chapter 1, there are 4 or 5 separate disk files which, taken together, comprise one whole RPL program. The names of these files are all derived from a 4 character name assigned by the RPL programmer when the program is first created. The source program is composed of the KFAM-3 files <name>F1 and <name>K1, while the object program contains the files <name>, <name>R, and sometimes <name>C. These files are all created and cataloged automatically by the RPL system.

Once a software configuration has been decided upon, the necessary system adaptations made (refer to Chapter 3), and an RPL program written, it is time to begin using the RPL editor. After mounting all the appropriate disks and clearing memory, load module RPL and run it. The first prompt to appear requests the date, which may contain up to 8 characters. This date will be printed on the program listing generated during compilation. After keying in the requested information, always key RETURN(EXEC) to proceed.

The system then asks whether or not Standard Disk Usage (SDU) is to be applied. The system flags each RPL program as using or not using SDU at the beginning of each editing session; therefore, it is a simple matter to modify SDU usage later on. The next prompt asks the user to enter the (RPL) program name. This name must be 4 characters long; if there are embedded special characters, the name must be enclosed in single or double quotation marks. If there are trailing blanks they are considered a valid part of the program name. For example, if an RPL program were named BIT, the disk files created would be BIT F1, BIT K1, BIT, BIT R, and (if applicable) BIT C.

Next, the system asks whether the RPL program already exists. Enter N (No) to create a new program, or Y (Yes) to update an extant one. If SDU was not specified, the source program's disk device address is requested (refer to Appendix C for a list of allowable device addresses). The system does not allow creation of two RPL programs with the same name on the same disk; similarly, if a program is to be updated, it must already reside on the specified disk. If a new program is being created, the system asks for an estimate of the maximum number of source program lines. Since RPL does not provide any file manipulation utilities, this estimate should be large enough to allow for possible future expansions of the original program. If a new program is being created and there is not enough room on the specified disk to contain the two source program files, an appropriate error message appears on the CRT screen; otherwise, the files are created and initialized. If SDU was not specified, the next prompt requests the object program's disk device address. The object program files are not created until the source program is compiled; hence, if there is not enough disk space for the object code, an appropriate error message will appear on the program listing generated by the compiler.

Lastly, the message ENTER PROGRAM STATEMENTS appears. This message signals that control has been passed over to the RPL editor. The RPL editor can be used to create and edit RPL source programs in much the same way as the BASIC editor is used to create and edit BASIC programs. The EDIT, RECALL, INSERT, DELETE, ERASE, and cursor movement Special Function Keys and also the LINE ERASE key may all be used in RPL in almost exactly the same way as they are used when editing BASIC programs. In RPL, the EDIT SFK cannot be used unless the cursor is positioned at the very beginning of the statement line; however, the cursor may be moved with the cursor movement keys even when the line is not being edited (e.g., it is being typed in for the first time). (As RPL was not designed for an 80 character CRT screen, when editing 62 keystroke lines on this wider screen depress EDIT, ←, DELETE, replace the deleted character, and then edit as usual.) The RPL program may be listed in its entirety by typing LIST (EXEC); LIST <number> (EXEC) lists a particular line, and LIST <number 1>, <number 2> (EXEC) lists all program lines from line number 1 to line number 2, inclusive. The main difference between listing a BASIC program and listing an RPL program is that in RPL, the command word LIST must be keyed in letter by letter. To stop listing the program, depress any key except the HALT/STEP key. The RPL command REORG (EXEC) functions like the KFAM REORG: It compresses the source program files on disk, freeing more space for additional source program text.

Keying END (EXEC) serves to inform the RPL editor that the editing session is over, and the editor then passes control to the compiler. A message to turn on the printer and key RETURN(EXEC) to resume appears; make sure the printer is selected before doing so. The program is then compiled and a compiler listing of the source program is generated. (This compiler program listing fits on narrow (8 1/2" x 11") paper.) If you wish to make further changes to the source code at a later time before compiling it, key HALT/STEP (EXEC) CLEAR (EXEC) instead of END (EXEC). By doing this, the changes made in the source program are recorded on disk, but as the program is not compiled the object program is not updated and no listing is produced.

If error messages appear on the source program listing, generally the source program should be corrected and recompiled before running the object program. If the errors encountered during compilation are severe, the object program will simply give rise to a System 2200 error message when run; usually even if the object code does run, the report generated will not be satisfactory. One exception to this rule is the compiler error message which always appears whenever a subroutine is called from another Print group. In this case, provided that the subroutine call is a legitimate one, the error message should be ignored.

6.2 RUN TIME

Once the RPL program has been created and successfully compiled, it is time to initiate the run phase of the program. Since running the program is entirely separate from compiling it, the run phase described in this section is also the phase which will be repeated at prescribed intervals to produce reports.

Mount all the necessary disks in the appropriate devices (refer to Chapter 3 for an explanation of the software configuration). If the RPL program accesses any BASIC code, key CLEAR (EXEC), load the BASIC code, load <name>R, scratch <name>R, and save the entire program back as <name>R. Then clear main memory, load <name> and run it. A prompt requests the date, which may contain up to 8 characters. The date entered will be used as the report date (referred to as DATE in the source program). After keying in the information requested, always key RETURN(EXEC) to proceed. The next prompt requests that the printer be turned on; be sure it is also selected and supplied with paper sufficiently wide for the report.

The system then prompts for the input data file name and its corresponding device address if this information was not included in the FILE statement. If the data file is KFAM-3 file and the correponding key file device address was not specified in the INDEX statement, the next prompt asks for the key file device address. If a multivolume data file is to be processed, prompts to mount the next volume appear at appropriate times.

There are a number of errors which can occur during the run phase, but most of these can be easily diagnosed as long as the RPL programmer has a clear idea of the sequence in which RPL statements are executed. Obviously, the first questions which must be resolved in some way when an RPL program is started up are: Where is the rest of the object code (<name>R and, if applicable, <name>C)? If SORT has been specified or the data file is a KFAM-3 file, where is the RPL system disk? Where is the data file? After correctly ascertaining this information, the RPL system processes all Data Selection and Sort statements. Lastly, the Report statements are processed, beginning with the START group (if there is one). During the actual report generation (execution of <name>R), the object code essentially reads records from the data file and prints them, one at a time. The complete semantics of the Report statements is summarized in the flowchart in Appendix E.

PART 2: APPENDICES

APPENDIX A: RPL PROGRAM SYNTAX SUMMARY

All fundamental RPL statements except the Comment statement must be grouped together when an RPL program is displayed in sequential line number order. It is recommended that variables not be referenced before they are defined. The recommended order for each statement group is given in the table below; for further details, refer to subsection 5.2.2.

lowest line # ↓

RPL statement	Mandatory, Conditional, or Optional	# of statements allowed per program	Comments
SYSTEM	conditional	zero or one	It must be included if not SDU; otherwise, it must not be included.
RUN	conditional	zero or one	It must be included if not SDU; otherwise, it must not be included.
FILE	mandatory	one	
INDEX	conditional	zero or one	It must be included if the data file is a KFAM-3 file. Otherwise, it must not be included.
RECORD	mandatory	at least one	Enough RECORD statements must be included to define the whole data record layout. If the data records are unblocked, there may be up to 55 fields per record; if they are blocked, up to 38. The total number of RECORD plus WORK fields may not exceed 100. Fields must be defined in the exact order that they appear in the records.
BLOCK	conditional	zero or one	It must be included if the data file is blocked; otherwise, it should not be included. Only array-type blocking is allowed.

RPL statement	Mandatory, Conditional, or Optional	# of statements allowed per program	Comments
WORK	optional	as necessary	It must be used if WORK fields are needed for computational purposes. The total number of RECORD plus WORK fields may not exceed 100.
COND	optional	up to 39	It may be used to select records for printing only if the specified conditions are met.
SORT	optional	zero or one	
KEYS	conditional	zero or one (plus optional continuation statements)	It must be included if a SORT or a TOTAL <levels 1 to 9> group is included. Otherwise, there is no need for it.
LINES	optional	zero or one	
START	conditional	as necessary	At least one Print statement must be included.
HEADER	conditional	as necessary	At least one Print statement must be included.
DETAIL	conditional	as necessary	At least one Print statement must be included.
TOTAL	conditional	as necessary	At least one Print statement must be included. TOTAL levels must each be grouped together, preferably in descending order from level 9 to level 0.
* (Comment)	optional	as necessary	Comment statements are not compiled; they need not be grouped together.

highest line #

```
IWC        DATE 9/22/77        PAGE  1

SOURCE PROGRAM

100  * THIS RPL PROGRAM GENERATES A REPORT ON THE ONE AND
110  * ONLY PRODUCT OF THE INTERNATIONAL WIDGET CORPORATION
120  * (IWC), THE "DELUXE AUTOMATIC WIDGETIZER".  A DATA BASE
130  * IS MAINTAINED ON THIS PRODUCT, GIVING A DESCRIPTION
140  * OF EACH OF ITS COMPONENT PARTS.  THIS PROGRAM WILL
150  * PRODUCE A PARTS BREAKDOWN LISTING FROM THE DATA
160  * BASE.
170  *
180  * SINCE STANDARD DISK USAGE IS SPECIFIED EVERY TIME THIS
190  * PROGRAM IS EDITED, NEITHER A SYSTEM NOR A RUN
192  * STATEMENT IS INCLUDED.
200  *
210  * THE FOLLOWING STATEMENTS DEFINE THE DATA BASE MASTER
220  * FILE.  IT IS AN UNBLOCKED KFAM-3 FILE WHICH WILL BE
221  * RESIDENT ON THE "R" FLOPPY DISK (DEVICE ADDRESS B10)
222  * AT RUN TIME.  WE WANT TO USE THE #1 KEY FILE TO ACCESS
224  * RECORDS, ALTHOUGH SO LONG AS A VALID KEY FILE IS
226  * SPECIFIED IT DOESN'T MATTER WHICH ONE IS USED, SINCE
228  * THE RECORDS WILL BE SORTED BEFORE PRINTING ANYWAY.
229  *
230  * EACH RECORD IS COMPOSED OF EXACTLY SIX FIELDS:
231  * (UNBLOCKED)
233  *      PART      COMPLETE PART NUMBER
234  *      DESCR     DESCRIPTION OF THE PART
235  *      QTY       QUANTITY OF THE PART PER SUBASSEMBLY
236  *      COST      IWC COST FOR THE PART
238  *      TYPE      ONE-LETTER CODE FOR THE PART TYPE
239  *      REVDT     LAST DATE PART SPECIFICATIONS WERE REVISED
240  *
250  FILE  2: "MASTFO": B10
260  INDEX 1: B10
270  RECORD PART  = 12
280  RECORD   ASM=PART,1,3: SUBASM=PART,5,2: COMP=PART,8,5
290  RECORD DESCR = 30
300  RECORD QTY   = #
310  RECORD COST  = #
320  RECORD TYPE  = 1
330  RECORD REVDT = 8
340  *
350  * WE NEED SOME WORK FIELDS TO GENERATE THIS REPORT.
360  * XCOST WILL BE USED TO COMPUTE THE EXTENDED COST
370  * OF EACH PART, AND THE ARRAY (XCOSTT) WILL BE USED TO
380  * TOTALIZE THIS VALUE ON THE 3 CONTROL LEVELS.  TYPEW
390  * WILL HOLD A WORD DESCRIBING A PART TYPE.  LREVDT WILL
400  * CONTAIN THE LASTEST REVISION DATE.
410  *
420  WORK XCOST  = #:  XCOSTT(3) = #
430  WORK TYPEW  = 12
440  WORK LREVDT = 8
450  *
460  * WE ARE GOING TO SORT ON THE THREE FIELDS ASSEMBLY,
```

SOURCE PROGRAM

```
470  * SUBASSEMBLY, AND COMPONENT, WITH CONTROL BREAKS ON THE
480  * FIRST TWO.  FURTHERMORE, WE ARE GOING TO ELIMINATE
490  * ANY PARTS WITH A ZERO QUANTITY, AS THEY ARE IN THE
500  * FILE ONLY FOR HISTORICAL PURPOSES AND ARE NOT CURRENTLY
505  * IN USE.
510  *
520  SORT
530  KEYS ASM: SUBASM: COMP
540  COND QTY <> 0
550  *
560  * THIS SECTION OF THE PROGRAM PRINTS OUT THE TITLE PAGE
570  * AT THE BEGINNING OF THE REPORT.  IT INCLUDES THE
580  * COMPANY NAME, PRODUCT NAME, REPORT TITLE, DATE, AND
585  * SLOGAN.  BEFORE PRINTING THE TITLE PAGE, WE SET THE
586  * MAXIMUM NUMBER OF LINES PER PAGE TO 52.
590  *
595  LINES 52
600  START +20: 1: EXPAND: 15: "INTERNATIONAL WIDGET ":
605  START                 36: "CORPORATION"
610  START +4: 47: "Deluxe Automatic Widgetizer"
620  START +1: 52: "PARTS BREAKDOWN"
630  START +4: 52: "Date:": 59: DATE
640  START +18: 1: "'We only make one product, so it's good!'"
650  *
660  * THIS SECTION OF CODE PRINTS THE HEADING ON EACH PAGE
670  * OF THE REPORT.  THE HEADING INCLUDES THE PAGE NUMBER
680  * AND THE CURRENT ASSEMBLY NUMBER.
690  *
700  HEADER  1: EXPAND: 1: "DELUXE AUTOMATIC WIDGETIZER":
710  HEADER             35: "PARTS BREAKDOWN":
720  HEADER             53: "Page": 58: PAGE:"##"
730  HEADER +1: 111: "ASSEMBLY"
740  HEADER  1: EXPAND: 57: ASM
750  HEADER +2: 1: "PART NUMBER ": 15: "DESCRIPTION          ":
760  HEADER    40: "PART TYPE   ": 60: "QUANTITY  X  COST":
770  HEADER    83: "=  EXT. COST": 110: "REV. DATE"
780  HEADER   1: "------------": 15: "--------------------":
790  HEADER  40: "------------": 60: "--------     --------":
800  HEADER  83: "   ---------": 110: "---------"
810  HEADER +1
820  *
830  * THIS SECTION PRINTS A DETAIL LINE.  THIS INVOLVES
840  * TRANSLATING THE 1-CHARACTER PART TYPE CODE INTO A
850  * COMPLETE WORD.  IF TYPEW IS EVER "UNKNOWN" ON THE
855  * REPORT, THERE IS SOME ERRONEOUS TYPE IN THE DATA BASE
856  * FOR THAT PART.  THIS SECTION OF CODE ALSO COMPUTES THE
860  * EXTENDED COST WORK FIELD.
870  *
880  DETAIL TYPEW = "FABRICATED":  GOTO 970 IF TYPE = "F"
900  DETAIL TYPEW = "MANUFACTURED":  GOTO 970 IF TYPE = "M"
920  DETAIL TYPEW = "OEM":  GOTO 970 IF TYPE = "O"
```

SOURCE PROGRAM

```
940  DETAIL TYPEW = "RAW MATERIAL":  GOTO 970 IF TYPE = "R"
960  DETAIL TYPEW = "UNKNOWN"
970  DETAIL XCOST = QTY*COST:  ROUND XCOST,2
980  *
984  * PRINT THE DETAIL LINE FOR THE CURRENT RECORD.
987  *
990  DETAIL    1: PART: 15: DESCR: 40: TYPEW: 61: QTY:"#,###":
1000 DETAIL   73: COST:"######.##": 86: XCOST:"#######.##":
1010 DETAIL  110: REVDT
1020 *
1030 * TOTALIZE THE EXTENDED COST ON EACH CONTROL LEVEL.
1031 * SINCE THE EXTENDED COST IS NOT A NUMERIC RECORD FIELD,
1032 * WE CANNOT TOTALIZE IT AUTOMATICALLY WITH A TOTAL
1034 * STATEMENT.  RPL ALWAYS INITIALIZES NUMERIC FIELDS TO
1036 * ZERO, SO ALL 3 ELEMENTS OF XCOSTT EQUAL ZERO WHEN WE
1038 * EXECUTE THIS CODE FOR THE FIRST TIME.
1040 *
1050 DETAIL XCOSTT(3) = XCOSTT(3)+XCOST
1060 DETAIL XCOSTT(2) = XCOSTT(2)+XCOST
1070 DETAIL XCOSTT(1) = XCOSTT(1)+XCOST
1080 *
1090 * REMEMBER THE LATEST REVISION DATE IN LREVDT.
1100 *
1110 DETAIL GOSUB 8000
1120 *
1130 * THIS IS THE CODE FOR CONTROL LEVEL 2.  HERE WE
1140 * TOTALIZE AND PRINT THE QUANTITY (&QTY) AND ALSO PRINT
1145 * THE EXTENDED COST (XCOSTT) FOR THIS SUBASSEMBLY.
1150 * REMEMBER TO ZERO THE EXTENDED COST TOTAL FOR
1160 * THIS LEVEL.
1170 *
1180 TOTAL 2:  61: "-----": 85: "----------"
1190 TOTAL 2:  61: &QTY:"#,###": 85: XCOSTT(3):"$#######.##"
1200 TOTAL 2: +2
1210 TOTAL 2: XCOSTT(3) = 0.00
1220 *
1230 * THIS IS THE CODE FOR CONTROL LEVEL 1.  HERE WE
1240 * TOTALIZE AND PRINT THE QUANTITY (&QTY) AND ALSO PRINT
1245 * THE EXTENDED COST (XCOSTT) FOR THIS ASSEMBLY.
1250 * NEXT, WE PRINT A SEPARATOR LINE OF PLUS SIGNS (+)
1260 * BEFORE THE NEXT ASSEMBLY.
1270 *
1280 TOTAL 1:  61: "=====": 85: "=========="
1290 TOTAL 1:  61: &QTY:"#,###": 85: XCOSTT(2):"$#######.##"
1300 TOTAL 1: +2: 50: "+ + + + + + + + + +"
1305 TOTAL 1: +3
1310 TOTAL 1: XCOSTT(2) = 0.00
1320 *
1330 * THIS IS THE CODE FOR CONTROL LEVEL 0, THE GRAND
1340 * TOTALS.  WE SUMMARIZE THE ENTIRE PRODUCT WITH ITS
1350 * TOTAL QUANTITY AND COST, PLUS THE DATE OF THE MOST
```

SOURCE PROGRAM

```
1360 * RECENTLY REVISED PART (SAVED IN LREVDT BY
1365 * THE BASIC SUBROUTINE CALLED IN LINE 1110).
1370 *
1380 TOTAL 0: +1:  60: "------": 84: "----------":
1390 TOTAL 0:     103: "LATEST REV. DATE:"
1400 TOTAL 0:      60: "------": 84: "----------"
1410 TOTAL 0:  45: "GRAND TOTALS:": 60: &QTY:"##,###":
1420 TOTAL 0:  84: XCOSTT(1):"$########.##": 110: LREVDT

161 SOURCE STATEMENTS
```

NOTE:

A listing of the BASIC subroutine may be found after this source program listing.

OBJECT PROGRAM VARIABLE LISTING

VARIABLES

FIELD NAME	VARIABLE NAME	TYPE
ASM	STR(A0$,01,03)	RECORD
COMP	STR(A0$,08,05)	RECORD
COST	A3	RECORD
DATE	Y4$	SPECIAL
DESCR	A1$	RECORD
LINE	Z8	SPECIAL
LREVDT	A9$	WORK
PAGE	Z9	SPECIAL
PART	A0$	RECORD
QTY	A2	RECORD
REVDT	A5$	RECORD
SUBASM	STR(A0$,05,02)	RECORD
TYPE	A4$	RECORD
TYPEW	A8$	WORK
XCOST	A6	WORK
XCOSTT	A7()	WORK

KEYS

LEVEL	VARIABLE NAME
1	X1$
2	X2$
3	X3$

TOTALS

TOTAL NAME	LEVEL	VARIABLE NAME
&QTY	0	Z(03)
&QTY	1	Z(02)
&QTY	2	Z(01)

NO ERRORS

```
8000  REM  THIS IS A BASIC SUBROUTINE CALLED FROM THE
8010  REM  RPL IWC PROGRAM.  THIS SUBROUTINE TAKES THE
8020  REM  CURRENT RECORD'S VALUE OF REVDT (IN A5$) AND
8030  REM  COMPARES IT TO THE MOST RECENT OVERALL REVISION
8040  REM  DATE, LREVDT (IN A9$).  THE DATE IS IN THE FORM
8050  REM  MM/DD/YY.  WHEN WE EXECUTE THIS CODE FOR THE FIRST
8060  REM  TIME, LREVDT WILL BE ALL BLANKS SINCE IT IS
8070  REM  AUTOMATICALLY INITIALIZED BY RPL.
8080  REM
8090  REM  FIRST, SEE IF THE CURRENT YEAR IS MORE RECENT.
8100  REM  IF SO, STORE REVDT IN LREVDT AND RETURN.
8110  REM  IF NOT, SIMPLY RETURN.
8120  REM
8130       IF STR(A5$, 7, 2) > STR(A9$, 7, 2) THEN 8320
8140       IF STR(A5$, 7, 2) < STR(A9$, 7, 2) THEN 8330
8150  REM
8160  REM  NEXT, SEE IF THE CURRENT MONTH IS MORE RECENT.
8170  REM
8180       IF STR(A5$, 1, 2) > STR(A9$, 1, 2) THEN 8320
8190       IF STR(A5$, 1, 2) < STR(A9$, 1, 2) THEN 8330
8200  REM
8210  REM  NEXT, TEST THE CURRENT DAY.  IF IT IS NOT MORE
8220  REM  RECENT, RETURN WITHOUT REPLACING THE OLD VALUE
8230  REM  OF LREVDT.
8240  REM
8250       IF STR(A5$, 4, 2) > STR(A9$, 4, 2) THEN 8320
8260       GOTO 8330
8270  REM
8280  REM  IF WE HAVE BRANCHED TO STATEMENT #8320, WE SHOULD
8290  REM  REPLACE THE VALUE OF LREVDT AND THEN RETURN;
8300  REM  OTHERWISE, SIMPLY RETURN.
8310  REM
8320       A9$ = A5$
8330       RETURN
```

NOTE:

This subroutine is supplied only as an example. It need not have been written in BASIC; it could easily have been done in RPL.

IWC DATA BASE AS OF 9/22/77

(SEQUENTIAL RECORD LISTING)

PART	DESCR	QTY	COST	TYPE	REVDT
124-01/A0001	MOTOR HOUSING	1	38.00000	F	01/07/76
124-01/A0002	FAN	1	7.00000	O	02/11/75
124-01/B1278	SCREW	6	0.02000	R	07/14/76
124-02/Q0001	BRUSH	2	6.50000	F	04/28/77
124-02/Q0002	BRUSH SPRING	2	0.75000	O	03/26/77
124-02/Q0003	BRUSH MOUNT	2	1.50000	M	12/11/77
124-03/L0201	COMMUTATOR	1	11.70000	M	06/19/77
124-03/M3685	BEARING	2	3.50000	O	11/14/76
372-01/B0300	WHEEL	6	11.00000	O	11/01/77
372-01/B0301	WHEEL BEARING	12	3.60000	O	02/13/77
372-01/B0302	WHEEL MOUNT	6	24.00000	M	03/18/76
372-02/Z2950	STEERING HOUSING	1	75.50000	F	05/14/77
372-02/Z2951	STEERING MONITOR	1	486.00000	O	07/22/75
372-03/W0001	GYRO	1	365.26000	O	08/26/77
372-03/W0002	GYRO GEARING	2	62.50000	M	06/09/77
372-05/A1234	STEERING FAN	1	11.20000	O	04/12/77
891-02/P0001	0108 PROCESSOR	1	###.#####	O	04/26/77
891-02/P0100	PROCESSOR ADHESIVE	8	26.00000	R	03/27/77
891-03/S0200	POWER SUPPLY	1	698.80000	O	04/11/76
891-03/S0201	SUPPLY MOUNT	4	6.32000	M	05/19/77
891-04/M0001	MEMORY	32	100.00000	O	06/06/77
891-04/M0002	MASTER BUS LINE	1	278.00000	O	07/04/77
891-04/M0003	MASTER BUS ENDUNIT	2	40.00000	M	08/01/75
891-04/R0100	REGULATOR	1	129.80000	M	09/13/77
124-03/S1368	SPACER	4	0.20000	F	07/12/76
372-03/Z2960	GYRO HOUSING	1	112.00000	F	07/26/75
891-02/M1391	PROCESSOR MOUNT	1	33.90000	M	02/12/76
891-03/B1111	CIRCUIT BREAKER	1	28.45000	M	05/20/77
891-04/S3617	CUT-OFF SWITCH	2	26.50000	O	04/01/77
891-11/A1653	CABINET	1	285.00000	M	12/03/76

INTERNATIONAL WIDGET CORPORATION

Deluxe Automatic Widgetizer

PARTS BREAKDOWN

Date: 09/22/77

'We only make one product, so it's good!'

ASSEMBLY
124

PART NUMBER	DESCRIPTION	PART TYPE	QUANTITY	X	COST	=	EXT. COST	REV. DATE
124-01/A0001	MOTOR HOUSING	FABRICATED	1		38.00		38.00	01/07/76
124-01/A0002	FAN	OEM	1		7.00		7.00	02/11/75
124-01/B1278	SCREW	RAW MATERIAL	6		0.02		0.12	07/14/76
			8				$45.12	
124-02/G0001	BRUSH	FABRICATED	2		6.50		13.00	04/28/77
124-02/G0002	BRUSH SPRING	OEM	2		0.75		1.50	03/26/77
124-02/G0003	BRUSH MOUNT	MANUFACTURED	2		1.50		3.00	12/11/77
			6				$17.50	
124-03/L0201	COMMUTATOR	MANUFACTURED	1		11.70		11.70	06/19/77
124-03/M3685	BEARING	OEM	2		3.50		7.00	11/14/76
124-03/S1368	SPACER	FABRICATED	4		0.20		0.80	07/12/76
			7				$19.50	
			21				$82.12	
372-01/B0300	WHEEL	OEM	6		11.00		66.00	11/01/77
372-01/B0301	WHEEL BEARING	OEM	12		3.60		43.20	02/13/77
372-01/B0302	WHEEL MOUNT	MANUFACTURED	6		24.00		144.00	03/18/76
			24				$253.20	
372-02/Z2950	STEERING HOUSING	FABRICATED	1		75.50		75.50	05/14/77
372-02/Z2951	STEERING MONITOR	OEM	1		486.00		486.00	07/22/75
			2				$561.50	
372-03/W0001	GYRO	OEM	1		365.26		365.26	08/26/77

ASSEMBLY
372

PART NUMBER	DESCRIPTION	PART TYPE	QUANTITY	X COST	= EXT. COST	REV. DATE
372-03/W0002	GYRO GEARING	MANUFACTURED	2	62.50	125.00	06/09/77
372-03/Z2960	GYRO HOUSING	FABRICATED	1	112.00	112.00	07/26/75
			4		$602.26	
372-05/A1234	STEERING FAN	OEM	1	11.20	11.20	04/12/77
			1		$11.20	
			31		$1428.16	
891-02/M1391	PROCESSOR MOUNT	MANUFACTURED	1	33.90	33.90	02/12/76
891-02/P0001	0108 PROCESSOR	OEM	1	1065.00	1065.00	04/26/77
891-02/P0100	PROCESSOR ADHESIVE	RAW MATERIAL	8	26.00	208.00	03/27/77
			10		$1306.90	
891-03/B1111	CIRCUIT BREAKER	MANUFACTURED	1	28.45	28.45	05/20/77
891-03/S0200	POWER SUPPLY	OEM	1	698.80	698.80	04/11/76
891-03/S0201	SUPPLY MOUNT	MANUFACTURED	4	6.32	25.28	05/19/77
			6		$752.53	
891-04/M0001	MEMORY	OEM	32	100.00	3200.00	06/06/77
891-04/M0002	MASTER BUS LINE	OEM	1	278.00	278.00	07/04/77
891-04/M0003	MASTER BUS ENDUNIT	MANUFACTURED	2	40.00	80.00	08/01/75
891-04/R0100	REGULATOR	MANUFACTURED	1	129.80	129.80	09/13/77
891-04/S3617	CUT-OFF SWITCH	OEM	2	26.50	53.00	04/01/77
			38		$3740.80	
891-11/A1653	CABINET	MANUFACTURED	1	285.00	285.00	12/03/76
			1		$285.00	
			55		$6085.23	

+ + + + + + + + + +

	------	----------	LATEST REV. DATE:
GRAND TOTALS:	107	$7595.51	12/11/77

NOTE:

The grand totals (above) appear on a separate page due to a printer hardware page eject.

APPENDIX C: VARIABLE NAMING AND DEVICE ADDRESS CONVENTIONS

Variable Naming Conventions

RPL Source Program Variables

Field names assigned with the RECORD and WORK statements must be from 1 to 6 characters in length and must be composed only of upper case letters (A to Z). Duplicate field names are not allowed within one RPL program. These reserved keywords may not be used as field names:

command keywords			special keywords
SYSTEM	LINES	CONVERT	DATE
RUN	START	ROUND	PAGE
FILE	HEADER	UNPACK	LINE
INDEX	DETAIL	GOTO	EXPAND
RECORD	TOTAL	GOSUB	KEY
BLOCK		RETURN	
WORK		EXIT	
COND			
SORT			
KEYS			

BASIC Subroutine Variables

The RPL object program contains variables in the range A0-A9, A0$-A9$, B0-B9, B0$-B9$, ..., L0-L9, L0$-L9$ which correspond to source program field names. A list of BASIC variables in this range which have been assigned by the compiler is printed on the source program listing. These assigned variables may be referenced in BASIC code executed from RPL; in addition, unassigned variables in this range may be used. Variables outside this range should not be used.

Device Address Conventions

The printer is always assumed to have a device address of 215. The RPL system, system work file, RPL source programs, RPL object programs, and associated data (and work) files must all be resident on some disk platter if they are to be used. The disk device addresses allowed are:

310	320	330
B10	B20	B30
350		

Note that the RPL system (even without its work file) may not reside on minidiskette since it is too large for one minidiskette.

APPENDIX D: CONTENTS OF THE RPL SYSTEM DISK

The RPL system disk contains the following files:

1. To create or update source programs and compile:

RPL	System start-up.
RLS301AA	Create or update source program.
RLS301BA	Reorganize source program file.
RLS302AA	Set up for first scan.
RLS302BA	First scan of source program; print source program listing.
RLS303AA	Process RECORD, WORK, and KEYS statements; create name table.
RLS304AA	Allocate space for object modules; create object program start-up module.
RLS305AA	Generate code for SORT (part 1).
RLS305BA	Generate code for SORT (part 2).
RLS306AA	Start to generate report module of object program.
RLS306BA	Continue to generate report module.
RLS307AA	Process START, HEADER, DETAIL, and TOTAL statements.
RLS307BA	Check line number references.
RLS308AA	Finish generating report module.
RLS309AA	Print source program error listing.
RLS310AA	Process COND statements.
RLS311AA	Print Object Program Variable Listing.

2. Object program skeleton modules:

RLS340AA	Skeleton start-up module.
RLS350AA	Start of report module.
RLS351AA	End of report module.
RLS360AA	Format 0 read subroutine.
RLS361AA	Format 1 and 3 read subroutine.
RLS362AA	Format 2 read subroutine.
RLS363AA	Read subroutine for sorted data.

*3. Work file:

WORKFILE	Work file for compiler; may be used for SORT-3 work file.

*4. SORT-3 modules:

SORT3	Check parameters and calculate sort dimensions.
SORT300B	Generate code for SORT301A.
SORT300C	Generate code for SORT303A.
SORT300D	Generate code for SORT302B.
SORT301A	Pass 1 - Internal Sort.
SORT302A	Pass 2 - Merge (key sort only).
SORT302B	Pass 2 - Merge (full-record sort only) (never used by RPL).
SORT303A	Pass 3 - Output (key sort only).

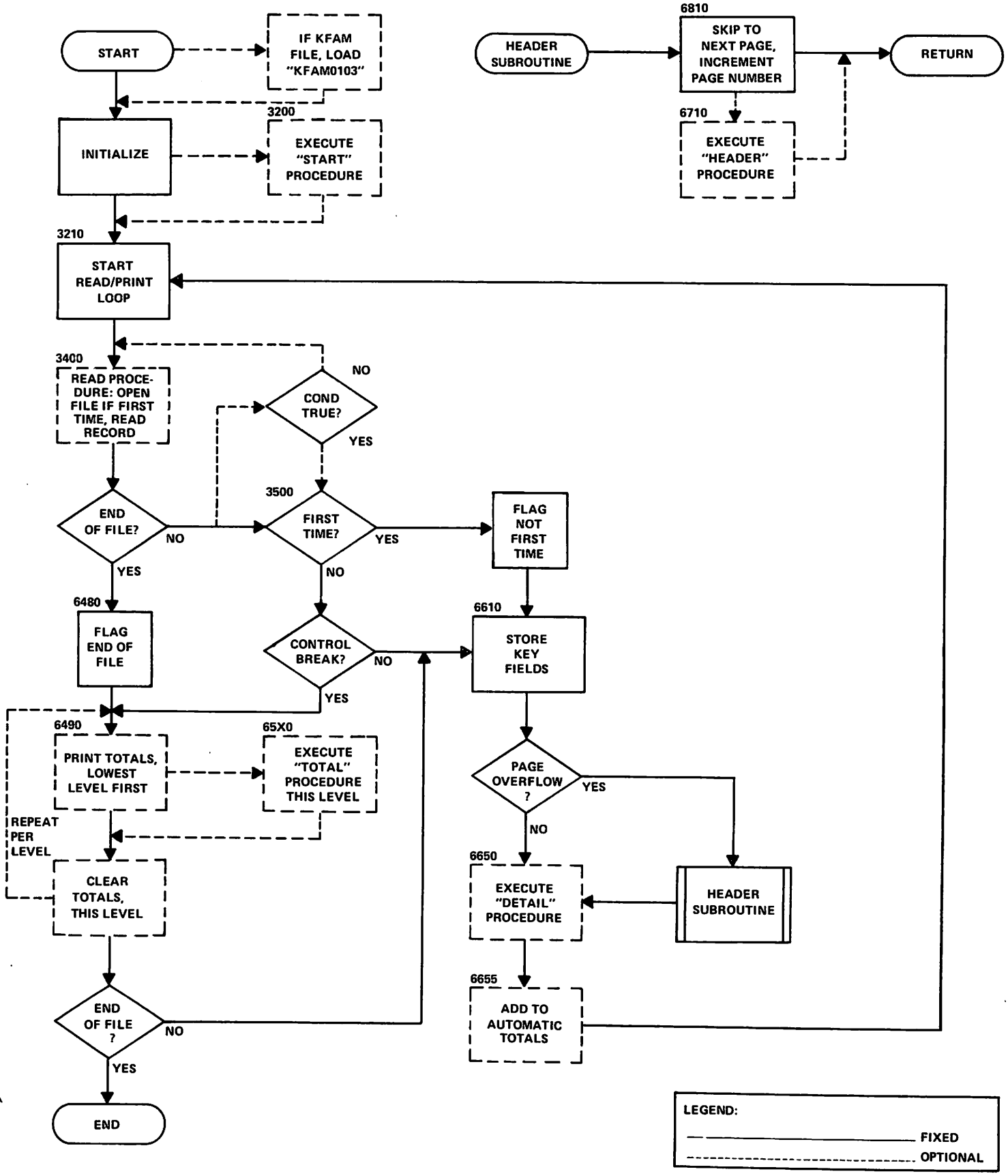
*5. KFAM-3 modules:

KFAM0103	Subroutines - inquiry-only version.

*Needed during the run phase if SORT is specified (with the default work file) or a KFAM-3 data file is used.

APPENDIX E: REPORT STATEMENTS FLOWCHART

(EXECUTION OF <NAME>R)



APPENDIX F: ERROR MESSAGES

Prior to Entering the RPL Editor

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
ERR 72	Disk read error.	Rerun the program. If the error persists, recreate the RPL system disk or the source program from backup.
ERR 85	Disk write error.	Rerun the program.
RE-ENTER	Too many characters were entered, or an invalid response was made to a "yes" (Y) or "no" (N) question.	Correct the response and reenter it.
INVALID DEVICE ADDRESS	The disk device address entered was not 310, B10, 350, 320, B20, 330, or B30.	Enter a valid disk device address.
SOURCE PROGRAM NOT FOUND	The named source program to be updated was not found on the specified disk device.	Recheck the software configuration and enter the correct information.
KEY FILE NOT FOUND (STOP)	The KFAM-3 user file of the source program (<name>F1) was found, but not its associated key file (<name>K1).	The key file must be on the same disk device as the user file of the source program. This error only arises if the key file has been tampered with or if SDU has been defined incorrectly. If SDU is being used, be sure lines 115 and 120 of module RPL contain the same disk device address. When the error has been corrected, rerun the editing session.
DUPLICATE NAME ON VOLUME	The new source program user file cannot be created because a file of the same name already exists on the specified disk device.	Assign the new program a unique 4 letter name. Especially if there are KFAM files on the disk, check the disk catalog index for the duplicate name (<name>F1).

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
DUPLICATE KEY FILE NAME ON VOLUME (STOP)	<name>F1 is not a duplicate, but its associated key file (<name>K1) which is to be created is a duplicate of an existing file name on the specified device.	Either specify a different program name or a different device address. Check the disk catalog index for the duplicate name.
ERR 29	The data entered was not numeric; numeric data was expected.	Enter a number.
NOT ENOUGH SPACE ON DISK (STOP)	There is not enough room on the disk platter to store the source program, considering the number of statement lines requested.	Mount a new disk with more available space or enter a smaller number for the maximum number of statement lines.

RPL Editor

ERROR ** <line>	An invalid line number or command was entered.	Depress the LINE ERASE key and then reenter the line.
ERROR ** LINE TOO LONG	An attempt was made to enter more than 62 characters in a statement line.	The line is set up for editing. On an 80 character CRT screen, move the cursor left one space, depress the DELETE SFK, replace the deleted character, and then edit as usual. Key RETURN (EXEC) when corrected.
ERROR ** NO SPACE - REORG	There is no space left in the source program file.	Enter REORG to reorganize the source program file. If this error occurs twice in a row, the source program contains its maximum number of statements. If this is the case and you wish to enter more statements, a new, larger source program must be created.
ERROR ** LINE # NOT FOUND	An attempt was made to edit a line that does not exist.	Depress the LINE ERASE key and then enter a valid command or statement line.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
ERROR ** LINE # INVALID	The line number entered was less than 1, greater than 2000, or not an integer.	Depress the LINE ERASE key and then repeat with a valid line number.
NULL FILE	There are no statements in the source program file.	None necessary.
STOP NULL FILE	END was entered, but there are no statements in the source program file. No attempt is made to compile the program.	Edit the source program before recompiling.
STOP PROGRAM ERROR	Hardware or software error.	Rerun the program. If the error persists, notify Wang Laboratories.
System hangs	An attempt was made to compile the program, but the printer was not turned on or not manually selected.	Turn the printer on and press the SELECT button.

RPL Compiler

These messages appear on the source program listing. If any other RPL system error messages occur consistently during compilation, notify Wang Laboratories. (Other System 2200 error messages may occur.) After correcting errors in the source program, it must be recompiled before running the object program.

NUMBER <number> INVALID	The number printed is not numeric, or is outside the range permitted.	Use a valid number within the range permitted.
INVALID CHARACTER IN <field name>	A source program field name contains a character greater than "Z". The character itself may be unprintable.	Retype the name, adhering to the naming conventions.
SOURCE STATEMENT MISSING	The program line contains a line number, but no source statement.	Delete the line or insert an appropriate instruction.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
SOURCE STATEMENT <statement> INVALID	The source statement is invalid.	Correct the error in the statement.
NO OPERANDS IN SOURCE STATEMENT	The source statement does not contain required operands.	Insert an appropriate operand or operands, or correct the syntax of the statement.
QUOTES NOT CLOSED	Quotation marks are not in pairs.	Modify the statement to contain an even number of quotation marks.
LEVEL NUMBER <number> INVALID	The TOTAL statement must contain a one-digit level number.	Insert a valid level number (0-9).
NO COLON FOLLOWING LEVEL NUMBER	The level number in the TOTAL statement must be followed by a colon.	Place a colon (:) after the level number or correct the syntax of the statement.
(<command keyword>) NO SPACE LEFT IN WORK FILE	The work file space (normally 120 sectors) has been exhausted. To compile exceptionally large RPL programs, the work file space may have to be increased. The command keyword at which compilation was terminated appears in parentheses.	Recalculate the work file size (1 sector per 4 source statement lines). Either shorten the RPL program or enlarge the system work file (WORKFILE).
DUPLICATE <command keyword> STATEMENT	Only one of each of these statements is allowed: SYSTEM, RUN, FILE, INDEX, BLOCK, SORT, or LINES.	Delete the extra statement.
<file name> NOT KFAM FILE NAME	Format 2 is specified in the FILE statement, but the input file does not appear to be a KFAM-3 file. The first 6 characters of the input data file name are printed. For KFAM files, position 5 must be "F" and position 6 must be a digit (0-9).	Correct the KFAM-3 file name or use the correct format in the FILE statement.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
NO PRINT GROUPS	At least one START, HEADER, DETAIL, or TOTAL statement must be included in the source program.	Include at least one START, HEADER, DETAIL or TOTAL statement.
<format number> INVALID FILE FORMAT	The file format number in the FILE statement must be 0, 1, 2, or 3.	Correct the FILE statement, using a valid file format number.
INDEX VALID ONLY WITH KFAM FILE	The INDEX statement should not be included unless Format 2 (KFAM-3) is specified.	Delete the INDEX statement or reexamine the format.
<number> INVALID RECORDS PER BLOCK	In the BLOCK statement, the records per block specified must be an integer from 1 to 255.	Correct the BLOCK statement.
<number> INVALID NUMBER OF LINES PER PAGE	In the LINES statement, the lines per page specified must be an integer from 1 to 999.	Correct the LINES statement.
<number> INVALID KEY FILE NUMBER	In the INDEX statement, the key file number specified must be an integer from 1 to 9.	Correct the INDEX statement.
INVALID FILE NAME	Quotation marks are not in pairs, a delimiter is missing, or the file name contains no characters. The file name must be enclosed in quotation marks whenever there are embedded special characters.	Correct the FILE statement syntax or file name.
INVALID DEVICE ADDRESS	A device address is not 310, B10, 350, 320, B20, 330, or B30.	Use a valid device address.
<command keyword> STATEMENTS MUST BE CONTIGUOUS	More than one group of a particular statement type was encountered.	Reorder the program into proper statement groups.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
LIMIT 200 ERROR MESSAGES	Compilation is terminated if there are 200 (or more) error messages.	Debug the source program.
<command keyword> STATEMENT MISSING	The indicated statement is required, but was not found.	Insert the missing statement or reexamine the program syntax.
NO RECORD DEFINITION	There are no RECORD statements in the source program.	Insert at least one RECORD statement.
MISSING FIELD NAME	A field name was expected; none was found.	Insert the field name necessary or correct the statement syntax.
NAME <field name> TOO LONG	The field name exceeds 6 characters in length. Only the first 6 characters are printed.	Change the field name so it conforms to the naming conventions.
INVALID DELIMITER = <delimiter>	The delimiter printed is not the delimiter expected.	Correct the statement syntax.
INVALID FIELD DEFINITION	A field definition is invalid in a RECORD or WORK statement.	Correct the indicated source program line.
DEFINING NAME <field name> INVALID	A partial field has been defined incorrectly.	Correct the definition or reference. Refer to subsection 5.4.2.
TOO MANY KEYS	The KEYS statement may not contain a total of more than 9 operands.	Correct the KEYS statement.
DUPLICATE FIELD NAME <field name>	A field name defined in a RECORD or WORK statement is a duplicate of a field name already defined, or is a reserved keyword.	Rename the field in error.
TOO MANY NAMES (<field name>)	A maximum of 100 field names may be defined. The rejected field name is in parentheses.	Restructure the RECORD and WORK statements such that the total number of fields does not exceed 100.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
SORT WITH NO SORT KEYS	The KEYS statement must be included whenever SORT is specified.	Eliminate the SORT or insert a KEYS statement.
NAME <field name> NOT DEFINED	The referenced field name has not been defined in a RECORD or WORK statement.	Define the field name in a RECORD or WORK statement or delete the reference to it.
INVALID SUBSCRIPT	The subscript exceeds the WORK array dimensions, or is not a valid number. It may be subscripting a scalar variable.	Correct the subscript to conform to the array dimensions. Subscripts may not be used with scalar variables.
NAME <field name> INVALID	The field name is not valid in the context. For example, COND statements require RECORD names, numeric expressions require numeric field names, &<name>s may only appear in TOTAL statements, RECORD names may not appear in TOTAL or START statements, etc.	Insert a field name appropriate to the context in which it is used.
LIMIT 30 &NAMES	A maximum of 30 automatic totals is allowed. Count 1 for each total at each level.	Rewrite the TOTAL statement using no more than the maximum number of &<numeric RECORD field name>s.
INVALID LINE NUMBER	The line number must be an integer from 1 to 2000.	Correct the line number.
OBJECT PROGRAM SPACE EXCEEDED	The space allocated for an object module (probably the report module, <name>R) is not large enough to hold the generated program.	Scratch and rename the existing <name>R module. Create a new, larger file with a DATASAVE DC OPEN <name>R, then recompile the source program.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
INVALID SYNTAX	The print format may have been mixed with the substatement format in one program line, or the statement syntax is incorrect in some other respect.	Refer to the syntax rules concerning punctuation and format. Correct the statement line accordingly.
INVALID LINE NUMBER REFERENCE	A GOTO or GOSUB references a line number between 2000 and 8000.	Correct the GOTO or GOSUB reference (1 to 2000 or 8000 to 9999).
NO ROOM ON DISK FOR OBJECT PROGRAM <object module name>	The compiler is unable to allocate enough space for an object module on the specified disk.	Store the object program on a different disk or free up more space on the currently used disk. Remember that the source program must be recompiled.
LIMIT OF 39 COND STATEMENTS EXCEEDED	There may be a maximum of 39 COND statement lines.	Restructure the program so that the number of COND statements does not exceed 39.
TOO MANY FIELDS ON INPUT RECORD	An input record may contain up to 55 fields if the data file is unblocked, or up to 38 fields if the data file is blocked.	Restructure the data file and rewrite the RECORD statements involved.
UNRESOLVED REFERENCE TO LINE <number>	A GOTO or GOSUB statement refers to a nonexistent line number, or to a line number outside the Print group but not greater than 2000. If there are more than 255 lines in the Print group, it may only mean that the capacity of the compiler to check line number references is exceeded. If a GOSUB refers to a subroutine in another Print group, the program has compiled correctly but this error message was nevertheless generated.	Correct the GOTO or GOSUB reference if necessary.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
SORT KEY TOO LONG	The total length of all fields specified as SORT keys may not exceed 64 bytes. Numeric fields each count as 8 bytes.	Restructure the SORT keys in the KEYS statement such that they do not exceed 64 bytes.

Run Time

The messages below may appear on the CRT screen during the running of an object program. If any RPL system error message other than these appears consistently, be sure there has been no user modification of the object program. If other SORT-3 error messages appear, notify Wang Laboratories. (Other System 2000 error messages may occur.)

ERR 72	Disk read error.	Rerun the program. If the error persists, the file may have to be recreated from backup data.
ERR 85	Disk write error. This is also possible during the SORT phase.	Rerun the program. If the error persists, there may be physical damage to the disk containing the SORT work file. Recreate the SORT work file on another disk.
INVALID DEVICE ADDRESS	The disk device address entered was not 310, B10, 350, 320, B20, 330, or B30.	Enter a valid disk device address.
ERR 80	File not found.	Recheck the software configuration.
STOP INPUT INVALID	No END record was found on a Format 0 input file, or the input file is not classified as a data file in the disk catalog. The system will not sort this file as requested.	Recheck the software configuration.
STOP WRONG INPUT FILE	The header record of a Format 1 (BAS-1) data file does not contain the correct file name.	Rerun the program using the correct input file, or modify the FILE statement to correct either the file name or the format number.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
STOP ERROR OPENING KFAM FILE	An error was detected in the KFAM OPEN subroutine.	Rerun the program. If the error persists, check the key file number specified in the INDEX statement.
STOP KFAM FINDFIRST ERROR	There are no records in the input data file. The system will not sort this file as requested.	Recheck the software configuration.
STOP INVALID RECORD FORMAT	The sample record being examined (first record of the input file) is not in the proper format for a data record. The control bytes for the data file are invalid for use with RPL. The system will not sort this file as requested.	Be sure that the format specified in the FILE statement is correct. This error message could mean the data file is damaged, in which case it must be recreated from backup. It could also signify that the data file was written in BA mode, and hence cannot be used with RPL. Refer to Chapter 4.
STOP NOT BLOCKED AS SPECIFIED	The sample block of records being examined does not have the same number of records per block as specified in the BLOCK statement. The system will not sort this file as requested.	Correct the BLOCK statement and be sure array-type blocking is used.
STOP NO ROOM TO SORT	The system will not sort very large records (or blocks of records) if there is not enough work space in main (user) memory.	Set the memory size larger and/or use a System 2200 with more memory. Redefining the data file may help. Refer to Chapter 3.
STOP WORK SPACE TOO SMALL	The SORT work file is too small.	Create a larger work file. Refer to subsection 5.4.3.
STOP INVALID RECORD LAYOUT	The record layout is not the same as specified in the RECORD statements. This error message may occur during sorting.	Correct the RPL RECORD statements, recompile, and rerun. This error arises frequently during sorting due to fields at the end of the data record being omitted in the RECORD statements. If the object program runs without the SORT statement, this is definitely the case.

ERROR MESSAGE	CAUSE	POSSIBLE RECOVERY
STOP NO RECORDS	There are no records to be sorted.	Be sure the data file contains valid data. If it does, the appearance of this error message may signify that the COND statements exclude all of the data file records.
ERR 01	Memory capacity exceeded.	Correct the main memory size. Refer to Chapter 3.
ERR 43	Input records or blocks of records do not all have an identical format. The system will not sort this file.	Correct the data file format.
ERR 20	Character data is not in the proper form for a CONVERT or an UNPACK substatement.	Correct the input data or process it as a character string.
ERR 22	An array subscript is out of bounds. See the Object Program Variable Listing to identify the variables.	Correct the source program (or, if applicable, the BASIC subroutine).
System hangs	The printer is not turned on or not manually selected.	Turn on the printer, make sure it is supplied with paper sufficiently wide for the report, and press the SELECT button.

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-3766C

TITLE OF MANUAL **REPORT PROGRAM LANGUAGE (RPL) USER MANUAL**

COMMENTS:

Fold

Fold

(Please tape, Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Lowell, Mass.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

— POSTAGE WILL BE PAID BY —

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Technical Writing Department

Fold

Cut along dotted line.

Printed in U.S.A.
13-1019

North America:

Alabama
Birmingham
Mobile

Alaska
Anchorage

Arizona
Phoenix
Tucson

California
Fresno
Inglewood
Los Angeles
Sacramento
San Diego
San Francisco
San Mateo
Sunnyvale
Tustin
Ventura

Colorado
Denver

Connecticut
New Haven
Stamford
Wethersfield

District of Columbia
Washington

Florida
Jacksonville
Miami
Orlando
Tampa

Georgia
Atlanta

Hawaii
Honolulu

Illinois
Chicago
Morton
Park Ridge
Rock Island

Indiana
Indianapolis
South Bend

Kansas
Overland Park
Wichita

Kentucky
Louisville

Louisiana
Baton Rouge
Metairie

Maryland
Rockville
Towson

Massachusetts
Boston
Burlington
Littleton
Lowell
Tewksbury
Worcester

Michigan
Grand Rapids
Okemos
Southfield

Minnesota
Eden Prairie

Missouri
Creve Coeur

Nebraska
Omaha

Nevada
Reno

New Hampshire
East Derry
Manchester

New Jersey
Howell
Mountainside

New Mexico
Albuquerque

New York
Albany
Buffalo
Lake Success
New York City
Rochester
Syracuse

North Carolina
Charlotte
Greensboro
Raleigh

Ohio
Cincinnati
Columbus
Middleburg Heights
Toledo

Oklahoma
Oklahoma City
Tulsa

Oregon
Beaverton
Eugene

Pennsylvania
Allentown
Camp Hill
Erie
Philadelphia
Pittsburgh
Wayne

Rhode Island
Cranston

South Carolina
Charleston
Columbia

Tennessee
Chattanooga
Knoxville
Memphis
Nashville

Texas
Austin
Dallas
Houston
San Antonio

Utah
Salt Lake City

Virginia
Newport News
Richmond

Washington
Seattle
Spokane

Wisconsin
Brookfield
Madison
Milwaukee

Canada
Wang Laboratories (Canada) Ltd.
Don Mills, Ontario
Calgary, Alberta
Edmonton, Alberta
Winnipeg, Manitoba
Ottawa, Ontario
Montreal, Quebec
Burnaby, B.C.

International Subsidiaries:

Australia
Wang Computer Pty. Ltd.
Sydney, NSW
Melbourne, Vic.
Canberra, A.C.T.
Brisbane, Qld.
Adelaide, S.A.
Perth, W.A.
Darwin, N.T.

Austria
Wang Gesellschaft M.B.H.
Vienna

Belgium
Wang Europe, S.A.
Brussels
Erpe-Mere

Brazil
Wang do Brasil Computadores Ltda.
Rio de Janeiro
Sao Paulo

China
Wang Industrial Co., Ltd.
Taipei, Taiwan

France
Wang France S.A.R.L.
Bagnolet
Ecully
Nantes
Toulouse

Great Britain
Wang Electronics Ltd.
Northwood Hills, Middlesex
Northwood, Middlesex
Harrogate, Yorkshire
Glasgow, Scotland
Uxbridge, Middlesex

Hong Kong
Wang Pacific Ltd.
Hong Kong

Japan
Wang Computer Ltd.
Tokyo

Netherlands
Wang Nederland B.V.
Ijsselstein

New Zealand
Wang Computer Ltd.
Grey Lynn, Auckland

Panama
Wang de Panama (CPEC) S.A.
Panama

Republic of Singapore
Wang Computer Pte., Ltd.
Singapore

Republic of South Africa
Wang Computers (South Africa) (Pty.) Ltd.
Bordeaux, Transvaal
Durban
Capetown

Sweden
Wang Skandinaviska AB
Solna
Gothenburg
Arloev
Vasteras

Switzerland
Wang S.A./A.G.
Zurich
Bern
Pully

West Germany
Wang Laboratories GmbH
Berlin
Cologne
Duesseldorf
Fellbach
Frankfurt/M.
Freiburg/Brsg.
Hamburg
Hannover
Kassel
Munich
Nuernberg
Stuttgart

International Representatives:

Argentina
Bolivia
Canary Islands
Chile
Colombia
Costa Rica
Cyprus
Denmark
Dominican Republic
Ecuador
Finland
Ghana
Greece
Guatemala
Iceland
India
Indonesia
Iran
Ireland
Israel
Italy
Jamaica
Japan
Jordan
Kenya
Korea
Lebanon
Liberia
Malaysia
Mexico
Morocco
Nicaragua
Nigeria
Norway
Pakistan
Peru
Philippines
Portugal
Saudi Arabia
Spain
Sri Lanka
Syria
Thailand
Tunisia
Turkey
United Arab Emirates
Venezuela
Yugoslavia

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
700-3766C
10-79-1M
Price: see current list