WANG

VS

Procedure Language Reference

VS Procedure Language Reference

4th Edition — June, 1982

800-1205PR-04

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, 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.

This manual replaces the third edition of the *VS Procedure Language Reference* (800-1205PR-03). For a list of changes made to this manual since the previous edition, consult the Summary of Changes.

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

PREFACE

This manual is both a tutorial and a reference for the Procedure language user. Chapter 1 discusses the uses of procedures for controlling limited and noninteractive job execution. Chapter 2 lists each procedure statement and its syntactical requirements.

Before using this manual, the user should be familiar with the concepts discussed in the *VS Programmer's Introduction* (800-1101PI) and *VS Program Development Tools* (800-1307PT). To supplement this manual, the *VS Procedure Language Quick Reference* (800-6201PP) is available as a convenient syntax reference.

In addition, topics treated in the following manuals provide information that can be helpful when writing procedures.

VS File Management Utilities Reference	800-1308FM
VS Operating System Services	800-1107OS
VS System Operation Guide	800-1102SO
VS System Management Guide	800-1104SM
VS System Utilities Reference	800-1303UT

Summary of Changes for the Fourth Edition

TOPIC	DESCRIPTION	AFFECTED PAGES
New Statements	ASSIGN statement permits assignment of values to either a variable or to a substring of a string-variable.	1-20 to 1-21, 2-3 to 2-4
	DECLARE statement defines variables to be used within the procedure.	1-20, 2-5
	EXTRACT statement extracts information from the system and stores it in variables.	1-21 to 1-22, 2-10 to 2-11
	MESSAGE statement displays text on the workstation and continues execution of the procedure.	1-23, 2-17 to 2-18
	PROMPT statement displays information on the workstation and accepts data for variables.	1-22, 2-22
	USING statement declares the formal parameters to a procedure.	2-32
New Features	Enhancement to GOTO statement; now backward branching is allowed.	1-26 to 1-33
	Enhancement to IF statement; expanded to include numeric comparisons among integer-variables, integer-constants, or step-labels, and string comparisons among string-variables, string-constants, or substrings.	2-13 to 2-15
	Enhancement to RENAME statement; a new library name, as well as a new file name can be specified now.	1-24, 2-25
	Enhancement to RUN statement; a library and/or volume can be specified.	2-27
	Enhancement to SET statement; addition of SPOOLIB and PRTFCLAS.	1-9 to 1-10, 2-29
	Enhancement to SUBMIT statement; a library and/or volume can be specified.	1-38 to 1-39, 2-30 to 2-31
	Addition of string-variables, integer-variables, string-constants, and integer-constants.	1-4 to 1-6
	Parameters can be passed to and from a procedure or language.	1-30 to 1-31
Miscellaneous	This manual documents version 2.07 of the Procedure Interpreter.	

CONTENTS

CONTENTS (continued)

FIGURES

CHAPTER 1
INTRODUCTION TO THE VS PROCEDURE LANGUAGE

1.1 PROCEDURES

VS Procedure language is used to write special routines that are referred to as procedures. Procedures can run system functions and supply required parameter information with or without user interaction. The capability to automate operational sequences is useful for restricting user interaction at the workstation, and is essential for running background jobs since user interaction is not permitted in background processing. Specifically, the user can write procedures to perform the following functions:

- Run a program or series of programs
- Set default values for commonly used parameters
- Supply requested parameter information directly to a running program, thus modifying or eliminating prompts that would be displayed at the workstation during interactive processing
- Extract information from the system and store it in variables defined in the procedure
- Mount and dismount volumes
- Scratch or rename files
- Print or submit jobs

Because procedures enable the writer to automate a sequence of operations and supply relevant parameters, procedures have several important uses:

- Procedures minimize the number of keystrokes required to perform an operation or a series of operations, and thus reduce the possibility of errors. Procedures also relieve the user of remembering operational sequences and fixed parameters. In general, whenever a job involving the same operation or sequence of operations must be performed repeatedly, it is worthwhile to use a procedure.

- Procedures enable the writer to control which parameter information requests are displayed to the user, and which parameter information requests are fixed and the user cannot modify. For many applications, it is beneficial to restrict the user from making file assignments or specifying other types of parameters at runtime. In such cases, a procedure can provide those parameters that the user cannot modify.

  Prompts that would normally appear on the workstation can be suppressed when a procedure provides the information they request. Since the prompts never appear, such parameter specification is completely transparent to the user. The procedure writer, therefore, has total control over which prompts the user sees, and which responses are required whenever an application program is run. This control is particularly useful when designing applications involving data entry personnel. By providing procedures for such applications, management is assured of complete command over data security.

- Procedures can be used to modify system-provided parameters that cannot be modified in any other way. The system fully defaults certain file assignment parameters (such as work files and print files); the user never receives a prompt soliciting file assignment information for these files. In such cases, the user can modify the default parameters that the system assigns by writing a procedure specifying the desired parameters.

- Procedures can be used to run background procedures. All parameters required by a background program must be supplied by the controlling procedure, since background procedures are prohibited from any interaction with a workstation. For more information on background procedures, refer to the *VS Programmer's Introduction.*

Procedures provide a flexible, easy-to-use mechanism for running system functions and supplying required parameter information with or without interactive user communication.

1.1.1 Procedure Terminology

Procedures are constructed of a series of procedure statements, each consisting of one or more of the following: verbs, operands, constants, variables, substrings, and labels.

Verbs

Each procedure statement must begin with a Procedure language verb describing the operation to be performed. The statement verbs available in the Procedure language fall into five categories:

- Verbs that can invoke system functions (RUN, SET, SCRATCH, RENAME, PROTECT, PRINT, SUBMIT, MOUNT, DISMOUNT, LOGOFF, and EXTRACT). All these verbs, except EXTRACT, perform essentially the same operations as the corresponding options and commands issued interactively through the Command Processor. Although EXTRACT cannot be issued interactively through the Command Processor, it is a Supervisor Call (SVC) that can invoke a system function.

- Verbs used to supply parameters to a program or to modify the default parameters displayed at a workstation (DISPLAY and ENTER).

- Verbs used to perform testing and branching within a procedure (IF, GOTO, and RETURN).

- Verbs used within a procedure (DECLARE, ASSIGN, and USING).

- Verbs used for workstation I/O (MESSAGE and PROMPT).

The procedure verbs are listed as follows. The syntax and general format of each verb is discussed in both this chapter and in Chapter 2.

ASSIGN	Assigns values to variables defined in the procedure.
DECLARE	Defines the variables to be used within the procedure.
DISMOUNT	Logically dismounts a disk or tape volume.
DISPLAY	Supplies or modifies the displayed default values in a file assignment or option list prompt and generates a screen transaction.

ENTER	Supplies file assignment or option list parameters requested by a running program without generating a screen transaction.
EXTRACT	Extracts information from the system and stores it in the variables defined in the procedure.
GOTO	Specifies the next executable procedure language statement (performs either a forward or backward branch).
IF	Controls conditional execution of procedure steps.
LOGOFF	Terminates procedure execution and logs the user off the system.
MESSAGE	Displays text on the workstation and continues execution of the procedure.
MOUNT	Logically mounts a disk or tape volume.
PRINT	Places a print file in the PRINT Queue.
PROMPT	Displays variables and constants on the workstation and also allows the user to accept data for variables.
PROTECT	Protects a disk file or library.
RENAME	Renames a disk file or library.
RETURN	Terminates procedure execution.
RUN	Runs a program or another procedure.
SCRATCH	Scratches a disk file or library.
SET	Sets default values for file assignments, procedure submittal defaults, and print mode options.
SUBMIT	Submits a procedure file into the PROCEDURE Queue for noninteractive execution.
USING	Declares the formal parameters to a procedure.

Operands

Procedure language statements (except PROCEDURE, LOGOFF, and RETURN) must contain one or more operands that constitute the portion of the statement operated upon. Operands vary in syntax, depending upon the verb they follow, and are discussed in detail in conjunction with their corresponding verbs. (For further syntactical information on the operands associated with a particular verb, refer to the discussion of that verb in Chapter 2.)

Constants

There are two types of constants: string and integer. A string-constant is a sequence of 1 - 256 characters enclosed in paired quotes. Either single quotes (') or double quotes (") can be used. To use quotes as characters within the string, the user must supply two occurrences of the character, for example, "He said, ""I don't care."""

The following string-constants, in conjunction with their corresponding verb, need not be enclosed in quotes. To simplify procedure writing, the procedure interpreter allows quotes around all string-constants.

DISMOUNT	volname
DISPLAY	value1, value2
ENTER	value1, value2
MOUNT	volname, unit#
PRINT	filename, libname, volname, class, form #, copies, status, disp
PROTECT	filename, libname, volname, owner, period, fileclass
RENAME	filename1, libname1, volname, filename2, libname2
RUN	filename, libname, volname
SCRATCH	filename, libname, volname
SET	value1, value2
SUBMIT	procname, library, volume, procedure-id, class, cpu limit (ss form only), status, dump, action, disp

An integer-constant is a sequence of one or more digits whose value is in the range -99999999 to 99999999.

Variables

A variable is used for storing information. For example, a user can store in a variable the number of blocks allocated for a file. There are two types of variables: string and integer. String-variables are fixed length (between 1 - 256 characters). Integer-variables are full-word signed integers whose value is in the range of -2147483648 to 2147483647.

A string of 2 to 31 characters identifies a variable. The first character must be an ampersand (&). The other characters in a string can be: A - Z, a - z, 0 - 9, @, $, #, and _.

Variables must be declared (defined) for use in a procedure. (Refer to Subsection 1.6.1 for more information on declaring variables.)

A variable can be used as a procedure statement operand. The content of a variable used as an operand is the value of that operand. For example, if variable &A = 'FILEXYZ', then RUN &A is legal. If &A = 'FILEXYZ IN LIB1', then RUN &A is illegal (since blanks are not allowed in this context).

The following user-specified names or strings, in conjunction with their corresponding verb, can also be used as variables:

DISMOUNT	volname
DISPLAY	value1, value2

ENTER	value1, value2
MOUNT	volname, unit#
PRINT	filename, libname, volname, class, form #, copies, status, disp
PROTECT	filename, libname, volname, owner, period, fileclass
RENAME	filename1, libname1, volname, filename2, libname2
RETURN	integer
RUN	filename, libname, volname
SCRATCH	filename, libname, volname
SET	value1, value2
SUBMIT	procname, library, volume, procedure-id, class, cpu limit (ss form only), status, dump, action, disp

Substrings

A substring is a portion of a string-variable represented by the character position defined by "start" for a specified "length". Start and length can be either integer-constants or integer-variables. The three ways to specify the substring length are as follows:

- When the procedure writer specifies a length, an integer-constant or integer-variable defines the substring length.
- When the procedure writer specifies *, the substring from start to the last nonblank character is used.
- When the procedure writer does not specify either a length or *, the substring from start to the end of the string is used.

The first character position of a string-variable is 1. The defined substring must be fully contained within the string-variable, i.e., the start must be within the variable, and start plus length must not exceed the length of the variable plus one.

Labels

Any statement in a procedure can be identified with a label. A label can be up to eight characters in length. Labels can exceed eight characters in length, but the Procedure Interpreter only examines the first eight characters. Duplicate label names or labels containing the same first eight characters are permitted. To resolve problems that might occur from duplicate labels, the Procedure Interpreter selects the label according to the rules stated in Section 1.12.

A label used to identify a statement must be immediately followed by a colon (:), except where noted throughout the text. The colon does not constitute part of the label name; it is merely a separator. The first character of a label name must be alphabetic. Succeeding characters can be alphanumeric; embedded blanks are not allowed.

Multiple labels can precede a statement. For example:

```
LABELA:  LABELB:  RUN PROCA
```

Only the label closest to the statement (LABELB:) can be used for backward referencing, return code testing, or return code generation. The other labels are for branching only. Labels have the following uses:

- A labelled statement can be referenced from other statements in the procedure for the purpose of obtaining parameter information — this is known as backward referencing. (Refer to Section 1.14.)
- A labelled statement can be branched to from a GOTO statement located elsewhere in the procedure. (Refer to Section 1.11.)
- A label causes the return code generated by certain statements to be saved for user inspection or for testing using an IF statement. (Refer to Sections 1.2 and 1.11.)
- A label in an inner nested procedure can be used as a parameter reference name by an outer nested procedure. (Refer to Section 1.15.)

For the purpose of this manual, labels of certain types of statements are given different names when they are used in referencing, branching, or return code testing of those statements. For such uses, the labels of system function statements (step statements, SET, MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, PRINT, SUBMIT, and RUN) are termed step-labels; the labels of parameter-supplying statements (specification statements DISPLAY and ENTER) are termed spec-labels; and the labels of other statements (IF, GOTO, RETURN, EXTRACT, MESSAGE, PROMPT, DECLARE, ASSIGN, and LOGOFF) are termed stmt-labels.

1.1.2 Entering and Running Procedures

A user creates, edits, and runs procedures in the same manner as programs. On execution, procedures are interpreted by the Procedure Interpreter. To create or edit a procedure, the user must run the EDITOR and specify PROCEDURE as the language name. The procedure text, like source program text, is stored as a named file. (For more information on the EDITOR, refer to the *VS Program Development Tools.)*

Once created, a procedure source file can be directly executed; it does not need to be compiled into an object file. Unlike programs, procedures are executed interpretively. The Interpreter examines each statement in the procedure file and performs the specified operation. As the Interpreter runs each program specified in a procedure, the user is requested to enter any parameters the procedure does not supply. If the Interpreter detects an error, it displays an error message describing the nature of the problem. To correct the error, the user must edit and rerun the file.

1.1.3 Procedure Language Syntax

The Procedure language syntax is generally free form, subject only to the following rules:

- Each procedure must contain a line starting with the letters PROCEDURE or PROC; everything following the letters PROCEDURE or PROC on the same line is interpreted as a comment. The line containing the letters PROCEDURE or PROC must precede any procedure statement but can itself be preceded by comment lines. In the following example the Procedure Interpreter treats PROGA LISTS PERSONNEL as a comment:

  ```
  PROC PROGA LISTS PERSONNEL
  RUN PROGA
  .
  .
  .
  ```

- Other comment lines within the procedure must either begin with an asterisk (*) in column 1 or be enclosed in paired brackets. The Procedure Interpreter considers all characters within the brackets, including the brackets, as blanks. The following example illustrates the use of comments in a procedure:

  ```
  *SORTS PERSONNEL
  PROCEDURE
  RUN SORT [SORT BY STATE CODE]
  .
  .
  .
  ```

- The user can nest a comment within another comment, for example, [SORT BY STATE [TO OBTAIN ZIP]]. Brackets within a quoted constant are part of that constant and are not considered comment delimiters.

- A colon (:) must be used to separate a statement label from the verb that follows. Space between the label and the colon should not occur, e.g., step1:.

- Spaces are used to separate verbs and parts of operands in a statement. Multiple spaces between the elements of a procedure statement are ignored by the Procedure Interpreter. Blank lines are allowed between statements or comments.

- Commas must separate multiple operands in a procedure statement, if the syntax indicates commas (refer to Chapter 2). Spaces are optional.

- Uppercase and lowercase text (A – Z, a – z) can be used within a procedure. However, lowercase text automatically converts to uppercase, except when part of a constant enclosed in quotes.

- The character in column 71 of line n is adjacent to the character in column 1 of line n + 1.

- All entries in column 72 are ignored.

A procedure statement can be extended onto more than one line without special continuation characters. For example:

```
SET INLIB=MS, INVOL=VOL444, OUTLIB=
    MSCOPY, PROGLIB=MSCOPY
```

1.2 PROCEDURE STEPS AND RETURN CODES

Each procedure contains one or more functional units called steps. The procedure steps control the basic functions that a procedure performs. These steps and functions are listed as follows. The remaining procedure language statements support these functions.

Step	Function
MOUNT	Mounts disk and tape volumes
DISMOUNT	Dismounts disk and tape volume
PRINT	Prints files
PROTECT	Protects files and libraries
RENAME	Renames files and libraries
RUN	Runs programs and procedures
SCRATCH	Scratches files and libraries
SUBMIT	Executes procedures noninteractively

MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, PRINT, and SUBMIT are stand-alone statements providing all parameter information for the operations they perform. All operands for these statements are contained within the statements themselves. The procedure steps defined by these statements, therefore, consist only of the single MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, PRINT, or SUBMIT statement. The RUN statement, however, runs a program or procedure that might require the specification of many different runtime parameters. The user can supply these parameters through one or more DISPLAY or ENTER statements following the RUN statement.

An important characteristic of procedure steps is that they generate return codes. When a user runs a program at a workstation, the program generates a return code which, if nonzero, is displayed upon program completion. The return codes that the system utility programs generate indicate success or failure and, in many instances, indicate which failure occurred. Similarly, user-written application programs can use return codes to indicate specific conditions the program recognizes.

When a program is run in a procedure RUN step, its return code can be tested and used to make decisions affecting procedure execution (refer to Section 1.11). The MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, and SUBMIT statements also generate return codes upon completion. These return codes are the same as those returned by the MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, and SUBMIT SVC instructions. For the list of return codes these statements generate, refer to the appendix of this manual. For the discussion of the corresponding SVCs refer to the *VS Operating System Services.*

Although the RUN, MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, and SUBMIT steps always produce return codes indicating the status of the operation performed, the Procedure Interpreter saves those codes only if the step is labelled. (Note that the label assigned to a procedure step, when used in reference or return code testing, is called a step-label.) The Interpreter discards the return code that the unlabelled step generates. The return code can be neither displayed at the workstation nor tested with a subsequent IF statement in the procedure. Return codes are discussed further in Section 1.11.

1.3 SET STATEMENT

The SET statement is similar to the SET Usage Constants command issued interactively through the Command Processor (refer to the *VS Programmer's Introduction).* The SET statement can specify default names to be used by commands and programs to identify the input, output, the program libraries and volumes, and the work and spool volumes. SET is also used to specify job submittal defaults, a default file protection class, and print mode options. For example, the following statement sets the user's default input library, program library, and print mode:

```
SET INLIB=GS, INVOL=VOL555, PROGLIB=MS, PROGVOL=VOL444, PRNTMODE=H
```

When this statement is executed in a procedure, GS on VOL555 becomes the user's default input library with VOL555 as the input volume, MS becomes the default program library with VOL444 as the volume, and the print mode is set to HOLD. Subsequently, any procedure statement, program, or command that refers to the input library uses the default names GS and VOL555, unless different library and volume names are supplied at runtime. Similarly, all programs this procedure runs are assumed to be located in library MS on VOL444, unless otherwise specified. The parameters not defined in a SET statement are not changed. In the previous example, the output library and volume are not specified; default values for these parameters remain unmodified, unless specified in another SET statement.

When the SET Usage Constants command is invoked interactively from the Command Processor, the user must enter default values into the fields in the displays. These fields are labelled with identifying names called *keywords.* For example, the default name for the output library must be entered opposite the keyword OUTLIB. The same keywords are used to identify the corresponding parameters in a procedure SET statement. (In the previous example, INLIB identifies the input library, INVOL the input volume, etc.)

Two keywords are offered through the SET statement that are not available through the SET Usage Constants command. These keywords are PROGLIB and PROGVOL, and are used as the default program library and volume. PROGLIB and PROGVOL are used only in procedures, for programs run by those procedures. PROGLIB and PROGVOL can be specified only in a procedure SET statement. If the user runs a program interactively through the RUN command from the Command Processor, i.e., not through a procedure, the program library and volume are specified by RUNLIB and RUNVOL.

The keywords that can be used in a procedure SET statement are as follows. (Refer to Chapter 2 of this manual and the *VS Programmer's Introduction* for further information.)

INLIB	Input library	PRNTMODE	Print mode
INVOL	Input volume	PRINTER	On-line printer
OUTLIB	Output library	PRTCLASS	Print class
OUTVOL	Output volume	PRTFCLAS	Print file class
RUNLIB	Run library	FORM#	Form number
RUNVOL	Run volume	FILECLAS	File class
PROGLIB	Program library	LINES	Number of lines per page
PROGVOL	Program volume	JOBQUEUE	Job status
WORKVOL	Work volume	JOBCLASS	Job class
SPOOLIB	Spool library	JOBLIMIT	Job CPU time limit
SPOOLVOL	Spool volume		

Keywords in a SET statement must be spelled correctly. The Procedure Interpreter ignores misspelled keywords; misspellings are not flagged as errors. Keywords are discussed in greater detail in Subsection 1.5.3 and in Chapter 2.

All default values specified in a SET statement, with the exceptions of PROGLIB and PROGVOL, become the permanent defaults for the rest of the user's session (until the user logs off the system), remaining in effect even after the procedure is terminated.

1.4 RUNNING PROGRAMS WITH A PROCEDURE (RUN, DISPLAY, AND ENTER)

The most significant and useful feature of procedures is that they can be used to run programs (or other procedures) and supply parameter information to those programs (or procedures). A procedure can run a single program or a series of programs with a minimum of user intervention. The RUN statement is used for this purpose. Like the RUN command issued from the Command Processor, the RUN statement must specify the name of the program to be run. The library and/or volume can be specified. The following examples illustrate four different ways to write the RUN statement:

```
RUN COPY

RUN PROC1 IN LIB1

RUN FILEX ON VOL4

RUN FILEY IN LIB2 ON VOL6
```

Note that unlike the SET statement, the RUN statement does not use keywords to identify the individual parameters. Instead, the parameter's library and/or volume name are specified. The program name immediately follows RUN, the library name follows the connective IN, and the volume name follows the connective ON.

The RUN statement also allows parameters to be passed by reference to a program or another procedure through the USING option. The following example illustrates the RUN statement with the USING option:

```
RUN  &PROGA ON VOL1 USING  &parameter1,  "ABC"
```

1.4.1 Levels of Default

The RUN statement is used to run either user-written programs stored in a program library or system utility programs stored in the system library @SYSTEM@. In either case, the system always checks the program library first for a named program. If the program cannot be located in the program library, the system library is checked automatically. (Note that the system library is checked automatically only for the RUN statement of a RUN step; it is not checked automatically for ENTER and DISPLAY statements contained within the RUN step.) Therefore, the user should not use system program names when naming user programs. If a user program has the same name as a system program and is located in the currently specified or default program library, the user program will always be run when a RUN statement with that name is executed.

The user does not need to specify the name of the program library and/or volume in the RUN statement, if default names for the program library and/or volume have been specified in a preceding SET statement. These names are used as defaults when a RUN statement is executed. For example, the following statement defines default names for the program library and volume:

```
SET PROGLIB=DJDDATA, PROGVOL=VOL666
```

A subsequent RUN statement, such as RUN PROGA, in which no program library is specified, automatically uses DJDDATA as the library name and VOL666 as the volume. (If PROGA is not found in library DJDDATA, the system automatically searches the system library before displaying a message at the workstation indicating that the program cannot be found.)

If the user does not specify PROGLIB and/or PROGVOL in a SET statement, the library containing the running procedure is used as the default program library. For example, if a procedure is located in library PRO on volume VOL444 and PROGLIB and PROGVOL are not specified in a SET statement, any RUN statement within that procedure, which does not specify a program library, automatically uses PRO on VOL444. If the procedure is not located in PRO, then the system library is searched.

No library or volume name is required when the user runs a system utility program because the system library is automatically searched when a program cannot be found in the program library. Thus, the following statement is sufficient to run the COPY utility unless there is a program named COPY in the program library:

```
RUN COPY
```

1.4.2 RUN Step

A RUN step consists of the RUN statement used to run a specified program, followed by any DISPLAY and ENTER statements used to supply parameter information for that program. A procedure can contain zero or more RUN steps; each RUN step executes a single program or procedure, and, depending upon the amount of interaction desired, provides some or all of the runtime parameters required by that program or procedure.

The use of DISPLAY and ENTER statements within a RUN step to provide runtime parameters for a program is a particularly valuable feature of procedures. When a program is run interactively from the workstation with a RUN command, the user must supply the parameter information that the program prompts request. Such information includes the identification of files the program uses, and the selections offered by the program at runtime. In the case of file assignments, default file names and locations usually are predefined with the SET Usage Constants command; these default names appear in the display. The user can press ENTER to accept default names or type new values to modify default names.

NOTE

Unlike the case for the RUN statement, the system library is not used for default parameter values for ENTER and DISPLAY statements contained within the RUN step.

In many instances, it is advantageous to place complete control of parameter specifications (such as file assignments) with the procedure writer, so that the program user never is given the option to alter these parameters. Alternatively, it may be desirable to provide the user with a specific set of default values that can be either accepted or overridden. When a procedure runs a program with a RUN statement, the DISPLAY and ENTER statements can be used to supply required parameters at runtime. An ENTER statement provides the parameters that the program requires with no prompt displayed at the workstation. A DISPLAY statement issues a prompt displaying the modifiable default values the procedure writer specifies.

Program parameter requests not modified or satisfied by a DISPLAY or ENTER statement automatically generate a screen transaction (a prompt to the user), even if the program is run from a procedure. Thus, the procedure writer can determine which parameters will be provided and transparent to the program user, and which will be modifiable by the user at runtime.

The following RUN step might be used to run the COPY utility from a procedure:

```
RUN COPY
ENTER INPUT FILE=FILE1, LIBRARY=MS, VOLUME=VOL444
ENTER OUTPUT FILE=FILE2, LIBRARY=GS, VOLUME=VOL444
```

In this example, the input and output file names and their locations are provided; the user would not be prompted for this information when COPY is run. Note, however, that the COPY program requires the specification of several other parameters (including the options desired) that are not provided in this RUN step. (Refer to the *VS System Utilities Reference.)* Prompts requesting this information are displayed at the workstation when COPY is run.

1.4.3 Running a Series of Programs from a Procedure

A procedure can contain a series of RUN steps, each of which runs a program or another procedure. Therefore, the procedure writer can construct procedures that automatically run a number of programs in sequence with a minimum of user intervention. In this case, individual programs are run in the order in which their corresponding RUN steps appear in the procedure. Each program must complete execution before the next program is run (except when forward branching is used; refer to Section 1.11). The following example illustrates sequential RUN steps:

```
PROCEDURE   RUNNING MULTIPLE PROGRAMS
            SET INLIB=MS, INVOL=VOL444, OUTLIB=MSCOPY,
              OUTVOL=VOL444, PROGLIB=MSCOPY, PROGVOL= VOL444
STEP1:      RUN COPY
            DISPLAY INPUT FILE = FILE1
            ENTER OPTIONS
            ENTER OUTPUT FILE = FILECOPY
STEP2:      RUN FILECOPY
```

The first line identifies this file as a procedure and contains a comment the author has made. The SET statement sets the default values for the input, output, and program libraries and volumes. STEP1 is the label of the first RUN step, which runs the system program COPY. Since COPY is a system utility, no values need be supplied for its library and volume (the system library is checked automatically). The DISPLAY statement forces a screen display for the input file for COPY. The values that will be displayed on this screen are the file name, FILE1, which the DISPLAY statement supplies, and the library MS and volume VOL444, which the SET default input library and volume provides. The user can accept or override these values.

The line, ENTER OPTIONS, functions as though the user had pressed ENTER on the options screen, thereby accepting the existing defaults. The ENTER OUTPUT line specifies the output file as FILECOPY and obtains the output library and volume from the defaults specified in SET; no screen transaction occurs. (Refer to Section 1.5 for more information on DISPLAY and ENTER.) The second RUN statement is labelled STEP2 and runs the program FILECOPY. No library or volume need be specified for FILECOPY, since these values are obtained from the default program library and volume.

The technique of running one or more procedures from a procedure, i.e., nesting procedures, requires special discussion and is treated in Section 1.15.

1.4.4 Declaring the Parameters Passed to a Procedure

To declare parameters passed to a procedure, the USING statement is used. The USING statement, not to be confused with the USING option in the RUN statement, defines the parameters passed to the procedure by reference. In the following example, the variables &INPUT and &OUTPUT are passed to PROCEDURE TO DO SOMETHING:

```
PROCEDURE TO DO SOMETHING
USING &INPUT AS STRING (10), &OUTPUT AS INTEGER
.
.
.
```

Any program that uses appropriate types and lengths of parameters can call to a procedure and pass data. The number of parameters passed must equal the number of parameters declared in the USING statement.

The USING statement is optional. If the statement is present, it must immediately follow the PROCEDURE statement. (Refer to Chapter 2 for the USING syntax.)

1.5 USING DISPLAY AND ENTER STATEMENTS TO SUPPLY PARAMETER VALUES TO A PROGRAM

1.5.1 GETPARM Request

A GETPARM is a request for parameter information by a program or procedure. A GETPARM can be satisfied either by a user at a workstation or by a procedure DISPLAY or ENTER statement. A GETPARM request for parameter information is illustrated in Figure 1-1.

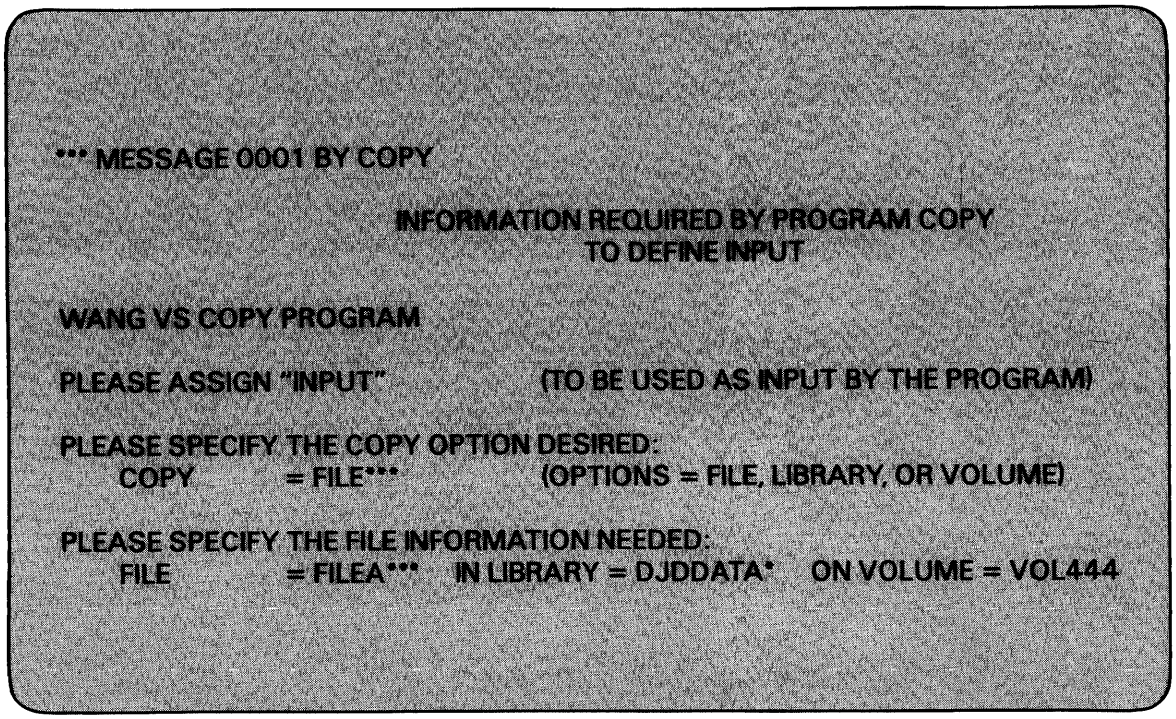


Figure 1-1. Sample GETPARM from Program COPY

In Figure 1-1, program COPY issues the GETPARM to solicit specification of the input file parameters. In this case, the user supplies the parameters interactively by running the program COPY. The user should note that each GETPARM issued by a program produces a single screen prompt soliciting one or more parameters.

The following example illustrates the same GETPARM when the parameters are supplied in a procedure using the ENTER statement:

```
PROCEDURE
RUN COPY
ENTER INPUT FILE=FILEA, LIBRARY=DJDDATA, VOLUME=VOL444
```

The relationship between a GETPARM issued interactively at the workstation (Figure 1-1), and a GETPARM supplied by a procedure (in the previous example) are as follows:

- The name of the program issuing the GETPARM is specified on line 1 after the Message number in a GETPARM request. In Figure 1-1, the program name is COPY. In a procedure, the name of the program is specified after the RUN verb.

- Each GETPARM request in a program is identified with a parameter reference name (prname). The prname is specified on the second line of the heading (line 4) in a GETPARM request. In Figure 1-1, the prname is INPUT. In a procedure, the prname is specified after the DISPLAY or ENTER verb to associate the statement with the GETPARM. (Refer to Subsection 1.5.2 for more information on parameter reference names.)

- Many GETPARM requests for information contain one or more modifiable fields into which the user (or a procedure) can enter information. Each modifiable field is referred to by a keyword. (Refer to Subsection 1.5.3 for more information on keywords.) In either a GETPARM prompt or a procedure, the keyword immediately precedes the modifiable field associated with it. In Figure 1-1, the keywords are COPY, FILE, LIBRARY, and VOLUME. In the procedure, the keywords are FILE, LIBRARY, and VOLUME; COPY is not specified since the keyword value defaults to FILE. Fields that are not assigned new values retain their default values. (Refer to Subsection 1.5.3 for more information on keyword value defaults.)

The user should note that if the GETPARM request does not receive valid parameters for all fields, it automatically generates a screen transaction. The prnames, keywords, and options must be spelled correctly.

Specifically, a GETPARM is an SVC or macro instruction that a user can write in an Assembler program; its function is to create a formatted prompt to solicit, receive, and verify parameter information. An Assembler language program can directly issue a GETPARM; a high-level language, such as BASIC, cannot directly issue a GETPARM. (The exceptions are the ACCEPT and DISPLAY statements in COBOL; refer to Chapter 2.)

When a procedure runs an Assembler program, and a GETPARM instruction is issued by the Assembler program, the prompt defined by that GETPARM is displayed on the workstation to solicit information from the user, unless there is a corresponding DISPLAY or ENTER statement in the controlling procedure RUN step. If the procedure contains an appropriate DISPLAY or ENTER statement, GETPARM obtains its parameters from the procedure. As a result, the GETPARM prompt either is never displayed or modified to contain specified default values before being displayed.

Wherever possible, all VS system utilities use GETPARM requests to solicit parameter information. It is, therefore, generally possible to provide most or all parameters a system program requires through a procedure. User-written Assembler programs that employ GETPARM to solicit parameter information can also be satisfied by DISPLAY or ENTER statements in a procedure.

DISPLAY and ENTER are restricted as to the types of parameter requests they can satisfy; they do not provide a general means of supplying information to a program. These statements cannot supply, for example, information that a COBOL DISPLAY and READ statement or a BASIC INPUT statement requests. Such high-level language features are designed to facilitate conversational interaction between a program and a user at the workstation. When one of these statements is executed, only a user or a procedure can satisfy the request for information.

Although high-level languages, such as BASIC, do not have the capability to issue GETPARM requests directly, programs written in these languages do issue GETPARM requests indirectly whenever they open a file. The Data Management System (DMS), an integral component of the VS Operating System, handles the process of opening a file. Whenever a program attempts to open a file, DMS automatically intervenes and issues a GETPARM request for the file parameter information. This special GETPARM request, called an OPEN GETPARM, can be supplied with parameters from a procedure DISPLAY or ENTER statement. It is possible, therefore, to supply any file assignment information required by a program from a procedure, even if the program itself is written in a high-level language.

In addition to GETPARM requests explicitly issued by an Assembler program and the OPEN GETPARM request issued whenever a file is opened, there is a class of GETPARMs that the system always automatically defaults. These default GETPARMs typically request information, such as file assignment parameters, for work files or print files. Since the system automatically provides default values, these GETPARM requests are never displayed at the workstation. Defaulted GETPARM requests can, however, be assigned values other than those provided by the system with a procedure. The procedure writer, therefore, can control the assignment of work files and print files that the user cannot control when a program is run from the workstation.

A brief description of GETPARMs, along with the lists of prnames, keywords, and options used in GETPARM requests can be found in the *VS System Utilities Reference* and the *VS System Management Guide* for GETPARMs issued by the system utilities, and in the *VS File Management Utilities Reference* for GETPARMs issued by the file management utilities.

1.5.2 Parameter Reference Name

A parameter solicited by a GETPARM request is identified as a parameter reference name (prname). Since each GETPARM instruction a program issues produces a single prompt soliciting one or more parameters, the prnames are collectively referred to as the parameter list for that request. The prnames associated with each GETPARM parameter list are usually unique within that program.

For easy reference, certain conventions are observed when prnames are assigned in system and file management utilities. For example, whenever a GETPARM requests parameter information for an input file, the prname for its parameter list is INPUT. Similarly, for an output file, the prname is OUTPUT. All GETPARM requests issued by system and file management programs, such as the system and file management utilities, and the associated prnames are listed in the manuals that document those programs. (Refer to the *VS System Utilities Reference, VS System Management Guide,* and *VS File Management Utilities Reference.)*

In a procedure, the prname is used in the DISPLAY or ENTER statement to associate the parameters specified in that statement with the parameter list of a particular GETPARM request. For example, INPUT is the prname in the following statement:

```
ENTER INPUT FILE=FILEA, LIBRARY=DJDDATA, VOLUME=VOL444
```

The DISPLAY and ENTER statements to be used with a particular program must be specified in the RUN step for that program, but they do not need to appear in any particular order. Whenever a GETPARM request is issued during program execution, the VS Operating System scans the list of DISPLAY and ENTER statements in the RUN step for a matching prname. If a DISPLAY or ENTER statement with a prname matching that of the GETPARM request is found, the parameters specified in the statement are assigned to the GETPARM parameter list. Otherwise, the GETPARM request produces a normal screen transaction. (Refer to the *VS System Operation Guide* for further information.)

If a program must solicit the same set of parameters more than once (for example, if several input files must be identified), the same prname is used to identify the parameter list each time the GETPARM request is issued. In this case, multiple DISPLAY or ENTER statements with the same prname are used in their order of appearance in the RUN step.

1.5.3 Keywords

Within a parameter list, individual parameters are identified by keywords. When a GETPARM prompt is displayed at the workstation, the modifiable fields that the user must enter parameter values are labelled with keywords identifying the requested parameters. These identifying keywords are used when parameters are supplied in ENTER or DISPLAY statements. For example, in the following statement the words FILE, LIBRARY, and VOLUME are keywords identifying file assignment parameters for the input file. The words FILEA, MS, and VOL444 are the keyword values, i.e., the names assigned to these parameters.

```
ENTER INPUT FILE=FILEA, LIBRARY=MS, VOLUME=VOL444
```

A keyword can be from 1 - 8 characters in length, with no embedded blanks. The keywords used in a DISPLAY or ENTER statement must be spelled correctly. The value associated with an incorrectly spelled keyword is ignored, and is not assigned to the appropriate parameter in the GETPARM parameter list. A list of the keywords used in each GETPARM request for each system program can be found in the manual that documents that program. (Refer to the *VS System Utilities Reference,* the *VS File Management Utilities Reference,* and the *VS System Management Guide.)*

Some keywords are assigned default values that a program or a SET statement supplies. In Figure 1-2, the keyword values YES, DISK, and 1 are defaulted. These default values can be accepted or overridden by a DISPLAY or ENTER statement. Fields for which no defaults are supplied must be given values by an ENTER statement in order to prevent a screen transaction. In Figure 1-2, keyword values need to be supplied for output FILE, LIBRARY, and VOLUME. If the GETPARM request does not receive valid parameters for all fields, it automatically generates a screen transaction.

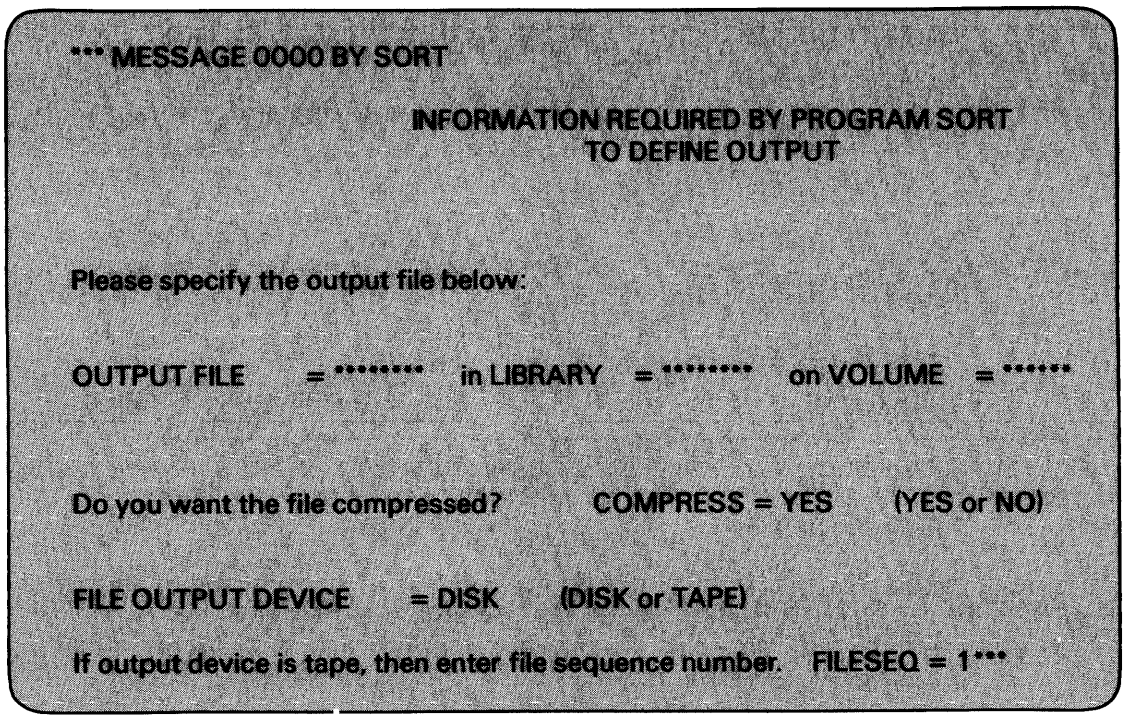


Figure 1-2. Sample SORT GETPARM

An example of an ENTER statement that provides enough parameter information to satisfy the GETPARM request in Figure 1-2 is as follows:

```
ENTER OUTPUT FILE=ABC, COMPRESS=NO, LIBRARY=DJDCTL, VOLUME=VOL666
```

In this example, the file name, library, and volume are specified. COMPRESS=NO automatically overrides the default supplied. Because no other keywords are included in the ENTER statement, default values are used for all other fields in the request. Note that the keywords do not appear in the same order in the ENTER statement as in the GETPARM request. As long as keywords are correctly spelled, they need not appear in any particular order.

To specify parameters containing spaces, the values assigned to keywords must be enclosed in paired single or double quotes. The following example illustrates the use of quotation marks when assigning multiple tab values in the EDITOR:

```
RUN EDITOR
ENTER DEFAULTS TABS= '10 16 36 68'
```

To specify a blank keyword value, the procedure writer must specify a comma (,) as a separator, or a blank enclosed in quotes as the keyword value assigned to the corresponding parameter. If the last keyword value on a line is to be blank, then the procedure writer must specify a space enclosed in paired single or double quotes, since a comma(,) at the end of a line is interpreted as a continuation. In the following example, the keyword values corresponding to FILE and LIBRARY are blank as indicated by the comma; the keyword value corresponding to VOLUME is blank as indicated by the space enclosed in paired double quotes:

```
RUN EDITOR
DISPLAY INPUT FILE=, LIBRARY=, VOLUME=" "
```

1.5.4 ENTER Statement

The ENTER statement supplies parameters for a GETPARM request. Execution of an ENTER statement is the equivalent of satisfying a prompt at the workstation by supplying requested parameters (or accepting displayed defaults), and keying ENTER or pressing a specific PF key. Each ENTER statement supplies parameter information for one GETPARM request. If the parameter information provided by ENTER is complete and correct, no screen transaction is generated by the corresponding GETPARM request.

The ENTER statement must specify the parameter reference name of the GETPARM parameter list for which values are supplied. Individual values in the ENTER statement must be identified by keywords associated with parameters in the corresponding GETPARM parameter list.

For example, the following procedure runs the COPY utility without user intervention by supplying all required parameters with ENTER statements. (The COPY utility is described in the *VS System Utilities Reference* and the GETPARM requests are listed in an appendix.)

```
PROCEDURE
RUN COPY
ENTER INPUT FILE=FILEA, LIBRARY=MS, VOLUME=VOL444
ENTER OUTPUT FILE=FILEB, LIBRARY=GS, VOLUME=VOL555
ENTER OPTIONS
ENTER EOJ
```

In the previous example, the COPY utility issues four separate GETPARM requests whose prnames are INPUT, OUTPUT, OPTIONS, and EOJ. The order in which the corresponding ENTER statements are written is unimportant. (The order in which ENTER statements are listed in the previous procedure is different from the order in which the corresponding GETPARM requests are issued by COPY.)

If the user does not specify a value for a parameter in an ENTER statement, the parameter retains its default value, if any. For example, the INPUT GETPARM for COPY requests four parameters: FILE, LIBRARY, VOLUME, and COPY. COPY is used to specify the copy option (file, library, or volume); its default is file. Because the ENTER INPUT statement in the previous procedure does not specify a value for COPY, this parameter retains its default value.

The user should note that if no default value is specified for a parameter that is omitted from an ENTER statement, the GETPARM request generates a screen transaction to solicit the missing parameter. In general, whenever an ENTER statement contains insufficient or incorrect information, the corresponding GETPARM generates a screen transaction requesting the user to correct the information. If the parameter reference name is incorrectly spelled, the entire ENTER statement is ignored.

The last two statements in the sample procedure (ENTER OPTIONS and ENTER EOJ) deserve special comment. ENTER OPTIONS satisfies the OPTIONS prompt that COPY displays. Since no options are specified in the ENTER statement, this statement accepts all default options. It is the equivalent of a user's acceptance of default values by keying ENTER without changing displayed values. To change any options, the ENTER statement must specify the appropriate keyword and the new value. For example, to specify an indexed rather than sequential file, with an eight-byte key starting at byte one, the ENTER statement would be as follows:

```
ENTER OPTIONS FILEORG=I, KEYLEN=8, KEYPOS=1
```

The ENTER EOJ statement ends the COPY program. When COPY is run from the workstation, the user must, in response to the EOJ prompt, select either PF1 to rerun the COPY program or ENTER or PF16 to end the program. If the program is to be rerun from a procedure, the following statement would be used, where 1 represents PF1:

```
ENTER EOJ 1
```

1.5.5 DISPLAY Statement

The DISPLAY statement is used to set default values for specified parameters in a GETPARM parameter list and then to force a screen transaction to display the parameter list to the user. The DISPLAY statement supplies default values that the user can accept or modify when the GETPARM request screen is displayed.

The DISPLAY statement has several uses. First, it can be used in situations where some parameters are to be provided and others are to be modifiable by the user at runtime. A file assignment, for example, might have predefined default values supplied for the library and volume, while the user must enter the file name. DISPLAY also can be used to solicit user entry of parameters, which then will be used repeatedly within the procedure. The backward reference technique (discussed in Section 1.11) makes it possible to pass the parameters entered in response to a DISPLAY statement to subsequent ENTER statements, thereby minimizing the number of times the user must be prompted to enter the same parameters.

For example, the ENTER INPUT statement used in the preceding sample procedure might be rewritten as follows:

```
DISPLAY INPUT FILE=FILEA, LIBRARY=MS, VOLUME=VOL444
```

In this case, the GETPARM request is displayed at the workstation with the values specified in the DISPLAY statement. The user has the option of accepting these values (by keying ENTER) or modifying one or more of them before keying ENTER.

Like the ENTER statement, DISPLAY specifies values for a single GETPARM request, and its parameter reference name and keywords must be spelled correctly. A value associated with an incorrectly spelled keyword is ignored; an incorrectly spelled prname causes the entire DISPLAY statement to be ignored.

NOTE

The DISPLAY statement cannot be used in a procedure run as a noninteractive, background job. Any use of DISPLAY in a background procedure causes job cancellation (except when DISPLAY is overridden by an ENTER statement in an outer nested procedure. Refer to Section 1.14.)

1.6 USING DECLARE AND ASSIGN STATEMENTS TO USE VARIABLES AS OPERANDS IN PROCEDURES

1.6.1 DECLARE Statement

The DECLARE statement defines variables that are to be used within the procedure. This statement also specifies the attributes of the variables and their initial values. In the following example, &FILE1 and &DDLIB are declared for use in PROCEDURE TWO:

```
PROCEDURE TWO
DECLARE &FILE1, &DDLIB AS STRING (8)
```

Two types of variables can be declared: string and integer. In the previous example, both variables are declared as string with a length of 8. Since the INITIAL clause is not specified, the initial value of the variable is all blanks. (Refer to Chapter 2 for the syntax of the DECLARE statement.)

When multiple variables are declared in the same statement, they all have the same type, length, and initial value.

The DECLARE statement is an executable statement; it must be executed prior to the use of any variables declared in the statement. Variables can be redeclared. If duplicate declarations are made, the most recently executed declaration is used.

1.6.2 ASSIGN Statement

The ASSIGN statement is used to assign values to either a variable, or to a substring of a string-variable. Two types of expressions are integer and string. Integer expressions are formed using + and −. For example:

```
ASSIGN  &variable1 = &variable1 + 1
```

In this example, &variable1 is assigned the value of &variable1 + 1.

String expressions are formed using the concatenation operator (!!). Concatenation of two operands takes the contents of the first operand and immediately follows them with the contents of the second operand. For example:

```
ASSIGN  &library = &proglib(1,3) !! "LIB"
```

In the previous example, &library is assigned the value of &proglib starting at position one for a length of three followed by the letters LIB.

A substring is a portion of a string-variable represented by the character position defined by "start" for a specified length. In the previous example, (1,3) designates a substring of variable &proglib starting at position one for a length of three characters.

Start and length can be either integer-constants or integer-variables. There are three ways to specify the length of a substring:

- When the procedure writer specifies a length, an integer-constant or integer-variable defines the substring length.
- When the procedure writer specifies *, the substring from start to the last nonblank character is used.

- When the procedure writer does not specify either a length or *, the substring from start to the end of the string is used.

The first character position of a string-variable is 1. The defined substring must be fully contained within the string-variable, i.e., the start must be within the variable, and start plus length must not exceed the length of the variable plus one. For example, the following statement is legal if &data_area is greater than 37 characters, i.e., the string-constant "abc" is the value in &data_area starting at character position 35 for a length of 3 (which is less than the length of &data_area plus one).

```
ASSIGN  &data_area (35,3) = "abc"
```

In an assignment to a string-variable or substring, the sending value is stored left justified. Excess character positions are truncated. If the sending length is shorter than the receiving length, the excess is blank filled. For example:

```
ASSIGN  &library (4,6) = "LIBDOC"
```

If &library is a 6 character variable with the contents CNTRLB, then when the previous statement is executed an error occurs.

In an assignment to an integer-variable, if the size of the sending value exceeds the receiving field size, high order digit truncation occurs.

If a step-label is used in an expression, but it does not have an associated value, an error is issued. (Refer to Chapter 2 for more information on the ASSIGN syntax.)

1.7 EXTRACTING INFORMATION FROM THE SYSTEM (EXTRACT)

The EXTRACT statement is used to extract information from the system and store it in variables. The information can be one of the following:

- Items available from the EXTRACT SVC
- Either the number of blocks allocated for a file or the number of records used in a file

Examples of the EXTRACT statement are as follows:

```
step1: EXTRACT &var = SYSVOL

EXTRACT &recs = RECORDS USED BY source IN srclib ON vol2
```

The EXTRACT statement function is analogous to the EXTRACT SVC executed with the specified keywords. In the first example (above), SYSVOL is the specified keyword. The keywords identifying fields for which data can be extracted are as follows:

CURLIB	PROGLIB	SYSLIB	FILECLAS	TAPEIO	USERNAME
CURVOL	PROGVOL	SYSVOL	FORM#	DISKIO	USERID
INLIB	RUNLIB	SYSWORK	LINES	PRINTIO	VERSION
INVOL	RUNVOL	WORKLIB	PRINTER	OTIO	TASK#
OUTLIB	SPOOLIB	WORKVOL	PRNTMODE	WSIO	WS
OUTVOL	SPOOLVOL	TASKTYPE	PRTCLASS		

The restrictions for the lengths of variables are as follows:

- If the receiving string-variable is too short to contain the data, the data is truncated from the right.
- If the receiving string-variable is longer than the data, the variable is blank filled on the right.
- Integer results are right justified, zero filled in the variable.

All variables must be declared in the procedure. A string-variable can receive data for all keywords (listed previously). An integer-variable can receive data for the following keywords:

TAPEIO	PRINTIO	WSIO	TASK#	LINES
DISKIO	OTIO	WS	FORM#	PRINTER

To extract the blocks allocated, the system sets the variable to the number of blocks allocated for the referenced file. If the file does not exist, the value -1 is assigned.

To extract the records used, the system sets the variable to the number of records used by the reference file. If the file does not exist, the value -1 is assigned. (Refer to Chapter 2 for more information on the EXTRACT syntax.)

1.8 PROMPT AND MESSAGE

Workstation I/O is available through the PROMPT and MESSAGE statements.

1.8.1 PROMPT

The PROMPT statement is used to display information (variables and constants) on the workstation, and then to accept data for variables.

The user can create up to 24 lines of 79 characters consisting of text and modifiable fields intermixed. Lines that are longer than 79 characters are truncated from the right. When a user creates more than 24 lines or uses an invalid attribute an error occurs. A semicolon (;) is used to mark the end of the constants and variables that are to appear on one line. Refer to Chapter 2 for the syntax of PROMPT.

To answer a PROMPT, the user presses either ENTER or a PF key. After a key is pressed, the screen is cleared automatically. Then the message "Procedure XXX In Progress" is displayed, and procedure execution continues.

If a PROMPT is issued and the workstation is open (opened, but not closed by other than the Procedure Interpreter) an error is automatically issued. If a procedure running in background issues a PROMPT, an error occurs.

The user should note that there is no relationship between PROMPT and GETPARM, i.e., the procedure writer cannot satisfy a PROMPT from a procedure.

1.8.2 MESSAGE

The MESSAGE statement displays text on the workstation and continues execution of the procedure. The procedure writer can arrange variables and constants on the screen using a simple statement format.

The user can create up to 24 lines of 79 characters consisting of text and variable values intermixed. Lines longer than 79 characters are truncated from the right. When a user creates more than 24 lines, an error occurs. A semicolon (;) is used to mark the end of the constants and variables which are to appear on one line. Refer to Chapter 2 for the syntax of MESSAGE.

When executed, the MESSAGE statement clears the current screen and displays the newly created screen, and procedure execution continues. The new screen remains until either another MESSAGE or PROMPT statement is executed, or until a RUN statement is executed. Prior to a RUN, the current screen is saved and after the RUN returns it is redisplayed on the workstation.

If MESSAGE is issued and a workstation is not available, (the workstation is opened but not closed by other than the Procedure Interpreter) or the procedure is running in background, then the message is not displayed and execution of the procedure continues.

1.9 SCRATCH, RENAME, AND PROTECT

The SCRATCH statement is used to scratch a named disk file or library, the RENAME statement is used to rename a disk file or library, and the PROTECT statement is used to change the protection parameters of a disk file or library. These statements are analogous to the SCRATCH, RENAME, and PROTECT options of the Manage FILES/LIBRARIES command issued interactively through the Command Processor.

1.9.1 SCRATCH

The procedure SCRATCH statement deletes all reference to the specified file from the disk Volume Table of Contents (VTOC) and frees the space occupied by the file for other use. For example, the following statement scratches file FILE1 in library MS on volume VOL444:

```
SCRATCH FILE1 IN MS ON VOL444
```

The user should note that this use of the IN and ON connectives is similar to that of the RUN statement. If the library and volume names are omitted from a SCRATCH statement, the default names specified for OUTLIB and OUTVOL in a preceding SET statement are used automatically.

The SCRATCH statement also can be used to scratch an entire library. Scratching a library causes files contained in that library to be scratched, except the files for which the user does not have scratch access rights or which have unexpired retention periods. For example, the following statement scratches library MS from volume VOL444:

```
SCRATCH LIBRARY MS ON VOL444
```

The user should note that the identifier LIBRARY and the connective ON are used to indicate a library scratch. To avoid ambiguity, the name LIBRARY is *not* allowed as a file name in the SCRATCH statement.

1.9.2 RENAME

The RENAME statement changes a file name in the disk VTOC to a new name, without altering the file. For example, the following statement changes the name FILE1 in library MS on volume VOL444 to FILE2. The name FILE1 can no longer be used to access the file.

```
RENAME FILE1 IN MS ON VOL444 TO FILE2
```

The RENAME statement can also change a file name in a library to a new file name and library. For example, the following statement changes the name FILE1 in library MS on volume VOL666 to FILE2 in library DJDLIB:

```
RENAME FILE1 IN MS ON VOL666 TO FILE2 IN DJDLIB
```

The user should note that this is equivalent to moving a file from one library to another on the same volume.

Like SCRATCH, RENAME uses the defaults specified for OUTLIB and OUTVOL in a SET statement if no library or volume is specified. The RENAME statement also can be used to rename a library (refer to the preceding discussion of scratching a library).

1.9.3 PROTECT

The PROTECT statement changes the protection parameters for a disk file or library. These parameters include the owner of record, the file protection class, and the retention period. For example, the following statement modifies the protection parameters of FILE1 so that DJD becomes the new owner of record, the file protection class becomes A, and the retention period becomes 21 days:

```
PROTECT FILE1 IN MS ON VOL444 TO
  OWNER = DJD, PERIOD = 21, FILECLAS = A
```

PROTECT can also be used to change the protection parameters of a library. For example:

```
PROTECT LIBRARY MS ON VOL444 TO
  OWNER = DJD
```

The SCRATCH, RENAME, and PROTECT statements can receive default parameters from the SET statement. For all of these statements, the default is the output library and/or volume. For example:

```
PROCEDURE
SET OUTLIB=DJDLIB
PROTECT FILE1 ON VOL444 TO OWNER=DJD, FILECLAS=B
```

The PROTECT statement protects the file named FILE1 in the output library DJDLIB (the default output library is used since no library was specified in the PROTECT statement) on the volume VOL444.

NOTE

Unlike the Command Processor SCRATCH, RENAME, and PROTECT options of the Manage FILES/LIBRARIES command, the SCRATCH, RENAME, and PROTECT procedure statements cause no message to be sent to the user if the statement cannot be executed. However, the SCRATCH, RENAME, and PROTECT statements do generate return codes. If the statements are labelled, these return codes, which indicate the success or the cause of failure, can be tested within the procedure and used as the basis for subsequent actions. Return codes are discussed in Subsection 1.11.1.

1.10 MOUNT AND DISMOUNT

MOUNT and DISMOUNT are used to automate mounting and dismounting of disk and tape volumes by permitting these operations to be performed from procedures. These statements are analogous to the MOUNT and DISMOUNT commands issued interactively through the Command Processor.

The MOUNT statement logically mounts a specified disk or tape volume on a specified unit prior to physical volume mounting. For disk mounting, the user may optionally specify a label type (STANDARD or NO) and a usage parameter (SHARED, PROTECTED, RESTRICTED REMOVAL, or EXCLUSIVE). Similarly, the user may optionally specify a label type (STANDARD, IBM, ANSI, or NO) and usage parameter (SHARED or EXCLUSIVE) for tape mounting. For example, the following statement mounts the disk volume named VOL444 on unit number 11:

```
MOUNT DISK VOL444 ON 11 WITH STANDARD LABEL FOR EXCLUSIVE USAGE
```

VOL444 has a standard label and is reserved for use only at the workstation from which it is mounted. The user should note that volumes mounted as EXCLUSIVE through a background processing job can be dismounted only through that background processing job. Such mounts must be dismounted before the background job can terminate. (Refer to Section 1.16 for further information.) To avoid ambiguity, the volume type (DISK or TAPE) must precede the volume name of any disk volumes named DISK and any tape volumes named TAPE in the MOUNT statement.

The DISMOUNT statement logically dismounts a specified disk or tape volume prior to physical volume dismounting. For example, the following statement dismounts the tape volume named TAPEVOL:

```
DISMOUNT TAPE TAPEVOL
```

As in a MOUNT statement, the volume type (DISK or TAPE) must precede the volume name of any disk volume named DISK or any tape volume named TAPE in the DISMOUNT statement to avoid syntax ambiguity.

The MOUNT and DISMOUNT statements are particularly useful for background processing to automate the mounting and dismounting of volumes needed in background jobs. When a MOUNT is issued from a background procedure, a message requesting the physical mounting of that volume is displayed at the Operator's Console. If there is more than one Operator's Console on the system, the message appears on the console with the highest priority. (Refer to the *VS System Operation Guide* and the *VS Programmer's Introduction.)* The MOUNT message remains on the screen until the specified volume is mounted. Similarly, a DISMOUNT issued from a background job forces an operator message, which remains on the screen until the operator keys ENTER from the Operator's Console, signifying that the DISMOUNT has been completed.

NOTE

Unlike the Command Processor MOUNT and DISMOUNT commands, the MOUNT and DISMOUNT statements cause no message to be sent to the user if the statement cannot be executed. However, MOUNT and DISMOUNT do generate return codes. If the statements are labelled, these return codes, which indicate the success or the cause of failure, can be tested within the procedure and used as the basis for subsequent actions. Return codes are discussed in the following section.

1.11 CONDITIONAL BRANCHING: INSPECTING AND TESTING RETURN CODES (IF, GOTO, AND RETURN)

Each labelled step in a procedure generates a return code upon completion. A return code is a numeric value (0 - 9999), assigned to a special register, and may be provided by user programs, system programs, or the VS Operating System. Return codes can be used in conditional branching within a procedure. When multiple labels precede a statement, only the label closest to the statement can be used for return code testing or return code generation.

1.11.1 Return Codes and Step-labels

The SCRATCH, RENAME, PROTECT, SUBMIT, MOUNT, and DISMOUNT steps generate return codes that indicate the status of the SCRATCH, RENAME, PROTECT, SUBMIT, MOUNT, or DISMOUNT operation (successful or unsuccessful; if unsuccessful, for what reason). RUN steps produce return codes generated by the program being run. In the case of system utility programs, the return code automatically indicates the status of the program upon termination. In user-written application programs, the programmer can set return codes within the program itself to indicate any special condition.

The return code generated by a labelled and executed procedure step can be tested by a procedure IF statement and used to conditionally execute a subsequent procedure statement. A procedure step return code also can be used within a RETURN statement to construct a return code for the procedure itself.

It must be emphasized that only labelled and executed procedure steps generate return codes that are retained. Although all programs produce return codes upon termination, these return codes are retained by the Procedure Interpreter for subsequent inspection and/or testing only if the corresponding RUN step has a label. The same situation holds for SCRATCH, RENAME, PROTECT, PRINT,

SUBMIT, MOUNT, and DISMOUNT. The label a procedure writer assigns to a procedure step that is used in testing and branching is called a step-label. Only steps with step-labels that are executed generate retained return codes. For example:

```
NEWFILE:  RENAME FILE1 IN MS ON VOL444 TO FILE2
```

In this RENAME statement, NEWFILE is the label of the step. This name could be used to examine and test the return code generated by the RENAME operation.

1.11.2 Testing and Branching: IF and GOTO

Return codes can be used for testing IF statements for conditional execution of procedure steps. Conditional execution is performed through the IF statement.

The IF statement is used to compare two operands and either branch to another statement or return to the invoker of the procedure if the condition is true. The comparisons are relational, i.e., equal, less than, greater than, not equal, not less than (greater or equal), and not greater than (less or equal). The comparisons can be numeric (among integer-variables, integer-constants, or step-labels), or string (among string-variables, string-constants, or substrings).

Conditional execution is accomplished by performing either a forward or backward branch to a specified step with a GOTO clause. For example:

```
COB:  RUN COBOL
      .
      .
      .
      IF COB GT 4 GOTO END
```

In this example, COB is the label of the RUN step that runs the COBOL compiler (none of the specification statements associated with this RUN step are shown in the example). Because a label is specified, the return code generated by the COBOL compiler is saved and is tested in the subsequent IF statement.

The IF statement tests the return code associated with COB to determine whether it is greater than 4 (indicating a serious problem with the compilation). If it is greater than 4, the procedure branches to a statement labelled END, which ends the procedure. If the return code is less than or equal to 4, no branch is taken, and the next sequential procedure statement is executed. This example can be expanded to a complete procedure that runs the COBOL compiler to compile a source program, then, depending upon the return code, either runs the compiled object program or terminates. For example:

```
PROCEDURE
COB:  RUN COBOL
      ENTER OPTIONS
      DISPLAY INPUT LIBRARY=GS, VOLUME=VOL444
      ENTER OUTPUT FILE=TEST, LIBRARY=GS,
        VOLUME=VOL444

      IF COB GT 4 GOTO END

      RUN TEST IN GS ON VOL444

END:  RETURN
```

It is not possible to perform multiple tests in a single IF statement. Several IF statements can be used, however, to perform multiple tests on the same return code. For example:

```
IF RCODE EQ 1 GOTO STEP1
IF RCODE EQ 2 GOTO STEP2
IF RCODE EQ 3 GOTO STEP3
```

Here, the step-label RCODE is tested for a return code of 1, 2, or 3, and a different procedure step is executed depending upon the value found. This feature provides great flexibility for the conditional execution of procedure steps, particularly when used in conjunction with application programs that set predetermined return codes.

The user should note that the GOTO clause can be used only to branch to a labelled statement either forward or backward within the procedure. (Refer to Section 1.12 for more information.)

1.11.3 RETURN Statement and Clause

In the previous procedure example, a RETURN statement, labelled END, is used to end the procedure. RETURN also can be used as a clause in an IF statement to terminate the procedure. (GOTO also functions as both a clause and a statement; refer to the discussion of GOTO in Subsection 1.11.2 and in Section 1.12.) Thus, the following statements could be replaced by the single statement: IF COB GT 4 RETURN.

```
IF COB GT 4 GOTO END

END:   RETURN
```

In this case, the procedure immediately terminates if the COBOL return code is greater than 4.

1.11.4 CODE Clause

For a procedure run from the Command Processor, a labelled procedure step that generates the return code can be displayed at the workstation upon procedure termination with the CODE clause. The CODE clause is used in conjunction with a RETURN clause (or RETURN statement) to display the return code associated with a specified labelled step. If the user does not specify a CODE clause, the return code displayed following termination of a procedure is always zero.

For example, a user can add the CODE clause to the IF COB GT 4 RETURN statement to produce the following statement:

```
IF COB GT 4 RETURN CODE=COB
```

In this case, the RETURN clause in the IF statement causes termination of the procedure if the COBOL return code is greater than 4, and CODE=COB specifies that the return code of the COBOL compilation step be displayed upon procedure termination.

The CODE clause also permits the procedure writer to specify an integer value to be displayed as a return code upon procedure termination or an integer-constant to be added to the program's return code. For example:

```
IF COB GT 4 RETURN CODE=10
IF COB LT 4 RETURN CODE=COB+100
```

This feature is a useful debugging tool for procedures that run multiple programs, since it provides a means of identifying which program caused the procedure to terminate. A different integer constant can be added to each program's return code. The constant then can be used to identify the program that caused termination, while the return code itself can be helpful in diagnosing the cause of the problem. The following short procedure illustrates this technique:

```
PROCEDURE
SCR:    SCRATCH TEST IN GS ON VOL666
        IF SCR GT 0 RETURN CODE=SCR+100
COB:    RUN COBOL
        ENTER OUTPUT FILE=TEST, LIBRARY=GS, VOLUME=VOL666
        IF COB GT 4 RETURN CODE=COB+200
        RUN TEST IN GS ON VOL666
```

A second important function of the CODE clause permits the procedure writer to test return codes when procedures are nested. A procedure can run one or more other procedures exactly as it runs programs. In this case, the return code produced by a nested or inner procedure can be tested by the outer procedure that runs it. This use of the CODE clause is further discussed in Section 1.15.

1.12 UNCONDITIONAL BRANCHING (GOTO AND RETURN)

Both the GOTO and RETURN verbs can be used as separate unconditional procedure statements. The statements function as they do in an IF statement where the relation tested is always true. Three examples are as follows:

```
GOTO STEP1

RETURN

RETURN CODE = COB
```

The GOTO statement specifies the next Procedure language statement to be executed. Both forward and backward branches are allowed.

Multiple statements can be labelled by the same label. For example:

```
RUN  COPY
INP: ENTER INPUT FILE=FILE1, LIBRARY=MS, VOLUME=VOL444
     .
     .
     GOTO INP
INP: DISPLAY INPUT
     .
     .
     .
```

Therefore, the following rules are used to determine which statement is the target of the branch:

- The first occurrence of the label following the GOTO statement in the procedure text, if it exists, is the target. In the previous example, the statement GOTO INP branches to INP: DISPLAY INPUT.

- Otherwise, the closest preceding occurrence of the label preceding the GOTO statement in the procedure text, if it exists, is the target.

- If neither rule applies, then an error has occurred.

The user should note that a backward GOTO works differently from a backward reference. The backward GOTO references the *closest textual* occurrence of a label, while the backward reference references the *most recently executed* occurrence of a labelled DISPLAY or ENTER statement. (Refer to Section 1.14 for more information on backward referencing.)

1.13 LOGOFF STATEMENT

The LOGOFF statement terminates the user's current session. All of the user's currently executing programs and procedures are terminated, all of their files are closed, and a Command Processor LOGOFF command is automatically issued.

When the EDITOR is used to run programs or procedures in a test environment, the LOGOFF statement is intercepted by the invoking program and returns to the EDITOR rather than terminating the user's session.

1.14 BACKWARD REFERENCE

Just as procedure statements defining steps can be identified for reference purposes with labels called step-labels, the specification statements DISPLAY and ENTER can be given labels termed spec-labels. A spec-label, enclosed in parentheses, can be used to identify a previously executed DISPLAY or ENTER statement for subsequent backward reference from other procedure statements. (This backward reference is not possible between statements of nested procedures.) When multiple labels precede a statement, only the label closest to the statement can be used for backward referencing.

A backward reference enables a procedure statement to obtain some or all of its required parameters from a preceding, previously executed labelled DISPLAY or ENTER statement. (Note that forward reference or reference to a preceding DISPLAY or ENTER statement which has not been executed is not allowed.)

1.14.1 Backward Reference to a Parameter List

Backward reference is a useful technique for eliminating repetitious user interaction when the same set of runtime parameters is used repeatedly. In the following example, a DISPLAY statement is used to request parameters from the user. These parameters subsequently are obtained through backward reference by other ENTER statements without additional screen transactions. For example:

```
INP:    DISPLAY INPUT
        .
        .
        .
        ENTER INPUT (INP)
```

This technique reduces redundant parameter solicitation and eliminates errors introduced by inconsistent user responses. There is no restriction on the number of statements that can reference a labelled DISPLAY or ENTER statement to obtain runtime parameters.

DISPLAY statements can be used in backward reference to permit a user to make runtime specification of files, libraries, or volumes to be run, printed, submitted, protected, scratched, renamed, mounted, or dismounted. DISPLAY is used to request parameters, which subsequently are obtained through a backward reference by one or more RUN, PROTECT, SCRATCH, RENAME, PRINT, SUBMIT, MOUNT, or DISMOUNT statements. For example:

```
SPECS:  DISPLAY INPUT
          .
          .
          .
RUN (SPECS)
```

The file specifications the user enters in response to the DISPLAY INPUT statement are later used by RUN to identify the program to be run.

Backward reference also provides a convenient form of abbreviation when a fixed set of parameters is used repeatedly within a procedure. This saves the procedure writer keystrokes, minimizes the possibility of typing errors, and makes subsequent modification or correction easier (the parameters must be changed in only one place). For example:

```
SPEC1:  ENTER OUTPUT FILE=FILE1, LIBRARY=MS, VOLUME=VOL444
          .
          .
          .
SCRATCH (SPEC1)
```

In this example, the ENTER statement, identified by the spec-label SPEC1, provides file parameters that are used by a subsequent SCRATCH statement, and can be used by any number of other statements within the procedure.

The use of backward reference simplifies the process of correcting incorrect parameters by avoiding multiple transactions when the same erroneous parameters are used in several places. For example, whenever a procedure-provided file assignment fails to anticipate the exact information required to locate or create a disk file, the system automatically displays a prompt to the user requesting the necessary corrections. These corrections can involve finding a file that has been moved, mounting a new volume, or finding a volume with sufficient available space to create a new file. When a backward reference obtains file parameters from a labelled DISPLAY or ENTER statement, the values obtained are the (possibly corrected) values actually used by the earlier procedure step, not necessarily those specified in the referenced statement itself.

For example, the wrong file name has been specified in the ENTER statement labelled INP in the sample procedure that follows:

```
PROCEDURE
START:  RUN COPY
INP:    ENTER INPUT FILE=FILE2, LIBRARY=MS, VOLUME=VOL444
          .
          .
          .
        ENTER INPUT (INP)
```

Here, the same set of input file parameters are used by two different ENTER statements. However, the file name specified in INP is incorrect (perhaps FILE2 refers to a nonexistent file). When program COPY is run, the system requests correction by the user. At that time, the user must enter the correct file name (perhaps FILE1). Subsequently, when the second ENTER statement is executed, the parameters obtained by it are those actually used by COPY. Thus, although the referenced ENTER statement specifies FILE2 as the file name, the file name obtained is the corrected one. The use of a backward reference in this case guarantees that, no matter how many times the file parameters are used, the correction must be made only once.

1.14.2 Backward Reference to Individual Parameters

In addition to obtaining the entire parameter list from a previous DISPLAY or ENTER statement, a backward reference can also be used to obtain selected parameters. Backward reference to a specific parameter is performed by means of a modified keyword called a refkey. A refkey is constructed from the spec-label of the referenced statement, followed by a period, and followed by the keyword identifying the desired parameter. The refkey, like the spec-label, is always enclosed in parentheses when used in a backward reference. The following example illustrates a backward reference to a specified parameter:

```
INP:  DISPLAY INPUT FILE=FILE1, LIBRARY=MS,
        VOLUME=VOL444
      .
      .
      .
      ENTER OUTPUT FILE=FILE2, LIBRARY=(INP.LIBRARY),
        VOLUME=(INP.VOLUME)
```

In this example, (INP.LIBRARY) and (INP.VOLUME) in the ENTER statement are refkeys that refer back to specific parameters in the DISPLAY statement. INP is the spec-label of the DISPLAY statement, while LIBRARY and VOLUME are keywords used in the DISPLAY statement to identify the library and volume names. When the ENTER statement is executed, it sets the output file name to FILE2, but obtains the library and volume names from the preceding DISPLAY statement labelled INP. (Although MS and VOL444 are specified as defaults in the DISPLAY statement, these are not necessarily the names obtained by ENTER. If the user changes the defaults when the input GETPARM prompt is displayed, the changed values and not the specified defaults are obtained by ENTER.)

The use of refkeys is not restricted to corresponding parameters in a backward reference. For example, if the procedure writer wishes to name a library with a name previously used for a volume, backward reference can be made to the volume name:

```
LIBRARY = (INP.VOLUME)
```

1.14.3 Summary of Rules for Backward Reference

To use backward reference in a procedure, a user must follow these rules:

1. Backward reference can be made only to a labelled specification statement (DISPLAY or ENTER) located within the same procedure as the referencing statement.

2. Backward reference can be used either to obtain an entire parameter list or to obtain only selected parameters.

   a. If a backward reference is used to obtain a complete set of parameters from a preceding specification statement, the parenthesized spec-label of the referenced statement is used as the operand of the referencing statement. For example:

      SCRATCH (STEP1)

      If the referencing statement is itself a specification statement, the appropriate prname must also be specified. For example:

      ENTER INPUT (INP)
      DISPLAY OUTPUT (OUT)

   b. If a backward reference is used to obtain selected parameters from a preceding specification statement, a refkey must be used to identify each referenced parameter. A refkey consists of the spec-label of the referenced statement, followed by a period, and followed by the keyword that identifies the desired parameter in the referenced statement, all enclosed in parentheses. For example:

      ENTER INPUT FILE=(INP.FILE), LIBR^.RY=(INP.LIBRARY),
        VOLUME=VOL444

      RUN (STEP1.FILE) IN MS ON VOL444

      SCRATCH (STEP2.FILE) IN SFK ON (STEP3.LIBRARY)

      NOTE

      The backward reference cannot be used to pass a PF key value from one ENTER statement to another. Each ENTER statement must directly specify its own PF key value.

3. When used in a backward reference, the spec-label and the refkey must be enclosed in parentheses.

1.15 NESTED PROCEDURES

A procedure RUN statement can be used to execute other procedures, as well as programs. The technique of executing one or more procedures from another procedure is called nesting procedures. A procedure that executes another procedure in this manner is the outer procedure. The procedure (or procedures) executed through RUN statements in the outer procedure are inner procedures. An inner procedure can, in turn, run a program or another procedure. The maximum level of nesting depends upon certain system conditions; in general, it is 16.

A RUN statement used to execute a procedure must include the same information required when running a program: the file name of the procedure file and its library and volume. For example:

```
PROCEDURE THIS IS PROCEDURE "OUTER"
RUN INNER IN MS ON VOL444

PROCEDURE THIS IS PROCEDURE "INNER"
RUN   COPY
      SPEC1:  DISPLAY INPUT LIBRARY=MS, VOLUME=VOL444
      SPEC2:  ENTER OUTPUT FILE=TEST, LIBRARY=GS, VOLUME=VOL555
      SPEC3:  ENTER OPTIONS
ENTER EOJ
```

In this example, procedure OUTER runs procedure INNER, which in turn runs the COPY utility and specifies values for the input and output libraries and volumes. (The name INNER must be the name of a procedure file in library MS on volume VOL444.)

1.15.1 Modifying Parameter Values in an Inner Procedure from an Outer Procedure

When a procedure is run with a RUN statement, DISPLAY and ENTER statements in the RUN step can be used to specify parameters that override parameters supplied in DISPLAY or ENTER statements in the procedure being run (the inner procedure). This feature, in effect, gives an outer procedure control over the parameter specifications made by all inner procedures run by that outer procedure. Thus, one or more existing procedures can be combined into a single large procedure by executing them with RUN statements from an outer procedure. Any changes required to parameter specifications in an inner procedure can be made simply by specifying the new parameters in the RUN step of the outer procedure. No modification of the inner procedures themselves is necessary.

When a procedure is executed by a RUN statement from an outer procedure, a DISPLAY or ENTER statement in the RUN step of the outer procedure can be associated with a labelled DISPLAY or ENTER statement in the inner procedure to supply parameter values to the inner statement. In this case, the parameters specified in the outer statement override those specified in the inner statement. The outer statement refers to an inner statement by specifying the spec-label of the inner statement as the prname in the outer statement. For example, consider the following ENTER statement that appears in procedure INNER:

```
SPEC2:  ENTER OUTPUT FILE=TEST, LIBRARY=GS, VOLUME=VOL555
```

An ENTER statement of the following form could be added to the RUN step in procedure OUTER to specify a new set of parameters for this statement:

```
ENTER SPEC2 FILE=NEWFILE, LIBRARY=NEWLIB, VOLUME=NEWVOL
```

The spec-label of the inner ENTER statement (SPEC2) is specified as the prname in the outer statement. When statement SPEC2 in procedure INNER is executed, the file parameters used are those specified in the outer statement (NEWFILE, NEWLIB, and NEWVOL) rather than those supplied in INNER statement SPEC2 itself.

Parameters for which no new values are specified in an outer procedure statement retain the values specified in the inner statement. For example, the ENTER statement from an outer procedure discussed in the preceding paragraph could be rewritten as follows:

```
ENTER SPEC2 LIBRARY=NEWLIB, VOLUME=NEWVOL
```

In this example, no value is specified for the file name. When statement SPEC2 is executed, therefore, its specified file name (TEST) is used; NEWLIB and NEWVOL are used as the library and volume names.

An ENTER statement in an outer procedure can modify a DISPLAY statement in an inner procedure. If an ENTER statement is used to supply parameters for a DISPLAY statement in an inner procedure, then no screen transaction occurs when the inner DISPLAY is executed, i.e., the request for user interaction generated by the inner DISPLAY statement is satisfied by the outer ENTER statement without a screen transaction.

Correspondingly, a DISPLAY statement in an outer procedure can modify an ENTER statement in an inner procedure. An outer DISPLAY always forces a screen transaction, even when it refers to an inner ENTER statement.

1.15.2 An Example of Nested Procedures

The example of nested procedures shown in the previous example can now be expanded to include a number of specification statements in the outer RUN step:

```
PROCEDURE THIS IS PROCEDURE "OUTER"
RUN INNER
        A:  ENTER SPEC1 FILE=OLDFILE, LIBRARY=OLDLIB, VOLUME=OLDVOL
        B:  ENTER SPEC2 LIBRARY=NEWLIB, VOLUME=NEWVOL
        C:  DISPLAY SPEC3

PROCEDURE THIS IS PROCEDURE "INNER"
RUN COPY
        SPEC1:  DISPLAY INPUT LIBRARY=MS, VOLUME=VOL444
        SPEC2:  ENTER OUTPUT FILE=TEST, LIBRARY=GS, VOLUME=VOL444
        SPEC3:  ENTER OPTIONS
ENTER EOJ
```

When the RUN statement executes procedure INNER from procedure OUTER, the following sequence of events occurs:

- Procedure INNER runs the COPY utility.
- The default values specified in statement SPEC1 in INNER are overridden by those specified in statement A in OUTER. Because statement A is an ENTER statement, statement SPEC1 does not generate a screen transaction.
- The values specified for library and volume in statement SPEC2 are overridden by those specified in statement B. Because statement B does not specify a file name, the file name supplied in SPEC2 is used.
- Statement SPEC3 is overridden by statement C, causing the OPTIONS prompt to be displayed for user interaction.
- Because the last statement in procedure INNER, ENTER EOJ, does not have a spec-label, it cannot be referenced by a statement in procedure OUTER. The statement ENTER EOJ, therefore, cannot be modified by OUTER.

1.15.3 Restrictions on Parameter Specifications between Nested Procedures

It is possible for an outer specification statement to supply values to an inner statement that is itself used to provide parameters to one or more other statements in the inner procedure through backward reference. In this case, the values obtained by the referencing statement(s) are those supplied in the outer statement, not those specified in the referenced statement itself.

Specification statements in an inner procedure that are not labelled, cannot be modified from an outer procedure, since it is the spec-label that is used to associate a particular inner statement with an outer statement. In this case, the parameters specified in the unlabelled inner statement are not altered by an outer procedure. Similarly, GETPARM requests for which no specification statement is provided in the inner procedure cannot be supplied values from an outer procedure. This difficulty is overcome by editing the inner procedure to supply missing specification statements or labels. The need to edit an existing procedure can be avoided; however, if the procedure writer consistently labels all specification statements and is careful to supply a specification statement for every GETPARM request the program issues. Even when no parameters or defaults are specified for a particular GETPARM request, a simple DISPLAY statement for that GETPARM provides a means for future parameterization from an outer procedure.

1.15.4 Nesting Procedures to More than One Level

When procedures are nested to more than one level, only the inner procedure(s) directly run by an outer procedure can be directly modified by the outer procedure. The procedure(s) run by an inner procedure cannot be directly affected by the outer procedure. However, dummy specification statements in the inner procedure that reference statements in a procedure run by the inner procedure can be used to pass values from the outermost procedure. For example:

```
PROCEDURE THIS IS PROCEDURE "OUTER"
RUN INNER1
        OUT1:  ENTER IN1 FILE=NEWFILE, LIBRARY=NEWLIB, VOLUME=NEWVOL


PROCEDURE THIS IS PROCEDURE "INNER1"
RUN INNER2
        IN1:  ENTER IN2


PROCEDURE THIS IS PROCEDURE "INNER2"
RUN COBOL
        IN2:  ENTER INPUT FILE=OLDFILE, LIBRARY=OLDLIB, VOLUME=OLDVOL
```

In this example, statement IN1 in procedure INNER1 is a dummy statement that specifies no values of its own, but serves as a conduit through which values are supplied to statement IN2 from statement OUT1. Statement OUT1 cannot directly modify values in IN2 because procedure INNER2 is run by INNER1 rather than by OUTER.

1.15.5 Testing the Return Code of an Inner Procedure

When one or more inner procedures are run by an outer procedure, the return code produced by an inner procedure can be tested following its termination by the outer procedure. In this way, the return code of an inner procedure can be used conditionally to determine which procedure step(s) are subsequently executed in the outer procedure. Procedures always produce zero return codes

unless the CODE clause is specified in the RETURN statement (or clause) that terminates the procedure. This is true whether the procedure is run from the Command Processor, EDITOR, or another procedure. If the CODE clause is used in a procedure run from the Command Processor or EDITOR, the return code is displayed at the workstation upon procedure termination. If the procedure is executed from a RUN step in an outer procedure, the return code is associated with the RUN step (the RUN step itself must be labelled) and can be tested in the outer procedure.

The following procedures provide an example of editing, compiling, and running of programs using nested procedures and return codes. Procedure OUTER runs procedure INNER. Procedure INNER, in turn, runs the EDITOR (to create the source program) and COBOL (to compile it). Upon completion, INNER returns the COBOL return code (indicating the status of the compilation). This code is tested by OUTER. If it is greater than 7 (indicating one or more serious compilation errors), a second procedure, FIXIT, is run. FIXIT reruns the EDITOR, permitting the programmer to make corrections to the source code, then recompiles the program. (The code for procedure FIXIT is not shown in this example.) If the return code obtained from INNER is less than 8 (indicating no serious compilation errors), a final return code of 100 is displayed to the user.

```
PROCEDURE OUTER
OUT1:  RUN INNER
       IF OUT1 LT 8 GOTO OUT3
OUT2:  RUN FIXIT
       IF OUT2 GT 7 RETURN CODE = OUT2
OUT3:  RETURN CODE = 100


PROCEDURE INNER
RUN EDITOR
     ENTER INPUT
IN1: DISPLAY OUTPUT
IN2: RUN COBOL
     ENTER OPTIONS
     ENTER INPUT IN1
     DISPLAY OUTPUT
     RETURN CODE = IN2
```

1.16 BACKGROUND PROCESSING

All background processing on the VS is implemented through procedures. Procedures used for background processing can include any of the available procedure statements, with the single restriction that *background procedures cannot generate screen transactions.* Any statements in a background procedure that attempt to force a screen transaction result in immediate abnormal termination of the procedure. Upon abnormal termination, a one-page, dump that resembles a DEBUG screen is automatically printed. Additional termination information may also be dumped, depending upon the value specified for the DUMP parameter at the time the job is submitted.

The exception to the rule against screen transactions is for the MOUNT and DISMOUNT statements, which were designed for background processing applications. These statements force screen messages at the Operator's Console requesting their specified mounts and dismounts. The volumes being mounted or dismounted, or their units for mounting, are not modifiable at this time.

Since workstation displays, except MOUNT and DISMOUNT messages, cannot be generated by background procedures, the procedure writer must include values for all parameter requests issued by background programs. For example, the following procedure results in an abnormal termination, since no values are specified for OUTPUT:

```
PROCEDURE
RUN COPY
ENTER INPUT FILE=FILE1, LIBRARY=SFK, VOLUME=SYSTEM
ENTER OUTPUT
RETURN
```

In interactive processing, this causes a workstation transaction. Similarly, any incorrect parameter values, misspelled prnames or keywords, or missing statements that otherwise cause workstation transactions result in abnormal termination. The user should note that DISPLAY statements should not be used in background procedures. DISPLAY statements can only be used in background processing in inner procedures of nested procedures when the DISPLAY statements are overridden by ENTER statements in the outer procedures. Any other use of DISPLAY attempts a workstation transaction and results in abnormal termination of the background job. Therefore, all procedures should be tested first in interactive mode to identify and eliminate any workstation transactions that cause abnormal termination, before background execution is attempted.

1.17 PRINT AND SUBMIT

The PRINT statement is used to queue a print file into the PRINT Queue, and is analogous to the PRINT option of the Manage FILES/LIBRARIES command issued interactively through the Command Processor. The SUBMIT statement is used to queue a procedure file into the PROCEDURE Queue for execution on a noninteractive basis, and is analogous to the SUBMIT Procedure command on the Command Processor.

1.17.1 PRINT

The PRINT statement submits a print file to the PRINT Queue. The file can either be printed as soon as a printer is available, or can be held for later use. After printing, the file can be deleted, requeued, or saved in the user's print library but not listed on the queue. Additionally, the print class, form number, and number of copies can be specified. For example, the following statement enters the print file EDIT0004 in library #DJDPRT on volume SYSTEM onto the PRINT Queue:

```
PRINT  EDIT0004 IN #DJDPRT ON SYSTEM, CLASS=A, FORM#=000,
       COPIES=2, DISP=SAVE
```

For printing, the print file class is specified as A, the form number as 000, and two copies will be printed. After printing, the file is saved in library #DJDPRT, but is not listed again in the PRINT Queue.

1.17.2 SUBMIT

The SUBMIT statement submits a procedure file to the PROCEDURE Queue. When specifying the file to be submitted, the user must specify the libarary and/or volume. For example:

```
SUBMIT FILE1 IN DJDLIB

SUBMIT FILE1 ON VOL2
```

If the user specifies the IN phrase, but not the ON phrase, the specified file is searched for in the specified library on PROGVOL. If that file does not exist, the specified file is searched for in the specified library on SYSVOL. If that file does not exist, a return code is issued. In the previous example, FILE1 IN DJDLIB is first searched for on the program volume, and if either the file or the program volume does not exist, then FILE1 IN DJDLIB is searched for on the system volume.

If the user specifies the ON phrase, but not the IN phrase, the specified file is searched for in PROGLIB on the specified volume. If that file does not exist, the specified file is searched for in SYSLIB on the specified volume. If that file does not exist, a return coded is issued. In the previous example, FILE1 ON VOL2 is first searched for in the program library, and if either the file or the program library does not exist, then FILE1 ON VOL2 is searched for on the system volume.

If the user specifies neither the IN phrase nor the ON phrase, the specified file is searched for in PROGLIB on PROGVOL. If not found, SYSLIB and SYSVOL are used as inputs to the SUBMIT SVC.

While in the queue, the procedure is listed with a procedure-id (a name used to identify the procedure on the queue). If the user does not supply the procedure-id, the system automatically assigns and generates one. Once in the queue, the procedure is either executed as soon as possible or is held pending later execution.

In addition, the queued procedure can be assigned a procedure class, a CPU time limit for execution, and an action to be taken if the specified limit is exceeded. In the event of abnormal termination, a dump can be made either automatically or through program control. After execution, the procedure is either removed from the PROCEDURE Queue or is listed again at the end of the queue. For example, the following statement enters the procedure file PROC1 in library PROCLIB into the PROCEDURE Queue under the job name TEST:

```
SUBMIT PROC1 IN PROCLIB AS TEST, CLASS=A, STATUS=
       HOLD, DUMP=YES, CPULIMIT=0:05:00, ACTION=WARN, DISP=
       REQUEUE
```

The procedure class is specified as A, and the procedure is held in the queue until the user releases or removes it. When executed, the operator is sent a message if the procedure exceeds five minutes of CPU time. If the procedure terminates abnormally, a dump is printed. When the job has successfully completed execution, it is listed again in the PROCEDURE Queue.

NOTE

Unlike their corresponding options on the Command Processor, the PRINT and SUBMIT statements cause no message to be sent to the user if the statement cannot be executed. However, PRINT and SUBMIT do generate return codes that can be tested (if labelled) and used as a basis for subsequent action. Refer to Section 1.11 for more information on return codes.

CHAPTER 2
PROCEDURE LANGUAGE STATEMENTS

This chapter contains the general format and syntax of each procedure statement, including a description of the purpose and function of each. The terms describing the syntax of each procedure statement are defined for each statement. Certain parameters and ranges of these syntax characteristics are common to all of the procedure statements, and are provided below for reference purposes.

fileclass	A one character value from among A - Z, #, $, ¢ or @. Fileclass can also be a string-constant, string-variable, or substring.
filename	A one to eight character alphanumeric value, that must begin with either an alphabetic or numeric character, @, $, or # and contain no embedded spaces. Filename can also be a string-constant, string-variable, or substring.
integer-constant	A whole number in the range -99999999 to 99999999.
integer-variable	A variable of type integer whose value must be in the range -2147483648 to 2147483647.
label	A one to eight character alphanumeric value, that must begin with an alphabetic character and contain no embedded spaces. A label must be followed by a colon (:), except a step-label is referenced in an IF, GOTO, or ASSIGN statement, or used in a refkey.
libname	A one to eight character alphanumeric value, that must begin with either an alphabetic or numeric character, @, $, or # and contain no embedded spaces. Libname can also be a string-constant, string-variable, or substring.
owner	A one to three character alphanumeric value, that must begin with an alphabetic character and contain no embedded spaces. Owner can also be a string-constant, string-variable, or substring.
period	A numeric value in the range 0 - 999. Period can also be a constant, variable, or substring.
pfkey	A numeric value in the range 1 - 32. Pfkey can also be a constant, variable, or substring.
prname	The parameter reference name is used in the DISPLAY or ENTER statement to identify the parameters specified in that statement.
refkey	A special type of keyword identifying a field in a labelled DISPLAY or ENTER statement preceding the current statement. The value associated with this keyword in the referenced statement is to be obtained for the current field through backward reference. A refkey consists of the label of the referenced statement, followed by a period, and followed by the keyword identifying the referenced field. Refkeys must be enclosed in parentheses.

spec-label	The label of a previous specification statement (ENTER or DISPLAY) from which parameters are to be obtained through backward reference for use in the current statement.
step-label	The label of a DISMOUNT, MOUNT, PRINT, RUN, SCRATCH, SET, SUBMIT, ASSIGN, RENAME, or PROTECT statement. Used in IF and RETURN statements to identify the return code to be tested.
stmt-label	The label of a MESSAGE, PROMPT, EXTRACT, DECLARE, ASSIGN, SET, LOGOFF, IF, GOTO, or RETURN procedure statement.
string-constant	A string of text of from 1 - 256 characters enclosed in single or double paired quotes.
substring	A portion of a string-variable represented by the character position defined by "start" for a specified length. Start and length can be either integer-constants or integer-variables. Substrings are allowed wherever a string-variable is allowed, except for the following statements: DECLARE, USING, RUN USING (as a parameter).
unit#	A numeric value in the range 1 - 099. Unit can also be a string-constant, string-variable, or substring.
variable	A string of from 2 to 31 characters. The first character must be an ampersand (&). The other characters can be chosen from the characters A - Z, a - z, 0 - 9, @, $, #, and _. Variables can be either uppercase or lowercase. Lowercase characters are converted internally to uppercase.
volname	A one to six character alphanumeric value, that must begin with either an alphabetic or numeric character, @, $, or # and contain no embedded spaces. Volname can also be a string-constant, string-variable, or substring.

The syntax of each procedure follows the format below:

Capitalized Words	=	Keywords
Lowercase Words	=	Terms (see above)
, : = ( ) + - ; !! & ' " .	=	Required syntax
{ }	=	One item must be encoded
[ ]	=	Optional item
...	=	Preceding item can be repeated

The procedure verbs with their syntax descriptions are arranged in alphabetical order in this section for easy reference.

2.1 ASSIGN

General Format:

```
[label:] ... ASSIGN  { integer-variable }  =  { integer-operand }
                     { string-variable  }     { string-operand  }
                     { substring        }

where:

integer-operand  = { [+] } { integer-variable } [ { + } { integer-variable } ] ...
                   { [-] } { integer-constant }   { - } { integer-constant }
                           { step-label       }         { step-label       }

string-operand   =  { string-variable } [ !! { string-variable } ] ...
                    { string-constant }      { string-constant }
                    { (refkey)        }      { (refkey)        }
                    { substring       }      { substring       }

substring        =  string-variable ( start [ , { length } ] )
                                                { *      }
```

The ASSIGN statement is used to assign values to either a variable or to a substring of a string-variable.

Two types of expressions are integer and string. Integer expressions are formed using + and −. String expressions are formed using the concatenation operator (!!). Concatenation of two operands takes the contents of the first operand and immediately follows them with the contents of the second operand.

A substring on the left of the = refers to the character positions within the variable into which the assignment is made. A substring on the right of the = refers to the contents of those character positions.

In an assignment to a string-variable or substring, the sending value is stored left justified. Excess character positions are truncated. If the sending length is shorter than the receiving length, the excess is blank filled.

In an assignment to an integer-variable, if the size of the sending value exceeds the receiving field size, high order digit truncation occurs.

If a step-label is used in an expression, but it does not have an associated value, an error is issued.

The four assignments possible are as follows:

- Integer-operand to integer-variable. To assign an integer-operand to an integer-variable a left to right evaluation of the integer-operand is performed. The resultant value is stored in the integer-variable.

- String-operand to integer-variable. To assign a string-operand to an integer-variable the string-operand is evaluated left to right. The resultant string must not exceed 16 characters (11 maximum of which can be a sign and digits) and must be in the following format: optional leading blanks, followed by optional sign, and followed by the digits of the integer, followed by optional trailing blanks. An error is issued if the size or format rules are violated. The string is converted to its integer value and is stored in the integer-variable. Strings must be in the range -2147483648 to 2147483647.

- Integer-operand to string-variable or substring. To assign an integer-operand to a string-variable or a substring, a left to right evaluation of the integer-operand is performed. The resultant value is converted to a string and stored in the string-variable or substring. If the receiving field is shorter than the sending field, truncation occurs from the right. The string consists only of the significant digits in the integer, and if negative, is preceded by a minus sign (if zero, then the string is '0').

- String-operand to string-variable or substring. To assign a string-operand to a string-variable or a substring, the string-operand is evaluated left to right. The resultant string is stored in the string-variable or substring. If the receiving field is shorter than the sending field, truncation occurs from the right.

2.2 DECLARE

General Format:

[label:] ... DECLARE variable [, variable] ... [AS] {STRING (n) / INTEGER} [INITIAL {string-constant / integer-constant}]

The DECLARE statement defines variables that are to be used within the procedure. It also specifies the type of the variables and their initial values. Two types of variables can be declared: STRING and INTEGER.

STRING defines fixed length string-variables with length n. The value of n must be 1 - 256. If the INITIAL clause is not specified, the initial value of the variable is all blanks.

INTEGER defines integer-variables. They are full-word signed integers. If the INITIAL clause is not specified, the initial value of the variable is zero. The integer specified for the initial value must be in the range -99999999 to 99999999.

If the INITIAL clause is specified, the initial value must agree in type with the declared variable.

When multiple variables are declared in the same statement, they all have the same type, length, and initial value.

The DECLARE statement is an executable statement. It must be executed prior to the use of any variables declared in the statement. Variables can be redeclared; if duplicate declarations are made, the most recently executed declaration is used.

2.3 DISMOUNT

Format 1:

[label:] ... DISMOUNT [DISK] volname

Format 2:

[label:] ... DISMOUNT TAPE volname

The DISMOUNT statement is used to dismount a mounted disk or tape volume. Its function is analogous to the DISMOUNT command issued interactively through the Command Processor. The DISMOUNT statement is particularly useful for automating dismounts for background processing and generally is used in conjunction with the MOUNT statement.

If the optional label is provided, a return code equal to that supplied by the DISMOUNT SVC is associated with the label. (Refer to the appendix for return code values.) This return code can be used for conditional branching by other procedure statements.

Unlike the DISMOUNT command issued interactively through the Command Processor, the DISMOUNT statement does not generate an error display if the specified volume cannot be dismounted.

NOTE

The disk volume type (DISK) is not optional for disk volumes named DISK. Therefore, a disk volume named DISK is dismounted in a procedure through the statement DISMOUNT DISK DISK.

2.4 DISPLAY

General Format:

```
[label:] ... DISPLAY {prname      } [ key1 = {value1   }  [, key2 = {value2   }] ... ]
                     {inner-label}   {(refkey1)}                   {(refkey2)}
                                     (spec-label)
```

The DISPLAY statement is used to override current default values for a specified GETPARM request. This statement is also used to display a prompt enabling the user to supply file assignments or options that are variable at runtime. A DISPLAY statement is always part of the same procedure step as the RUN statement it follows.

Any values supplied by a DISPLAY statement override defaults specified in the corresponding GETPARM request. When the GETPARM prompt is displayed, these values are displayed as defaults, along with any GETPARM-specified default values that have not been overridden. If a DISPLAY statement uses the label of an earlier ENTER or DISPLAY statement for a backward reference, any fields in the GETPARM request associated with this DISPLAY statement, whose keywords correspond to keywords in the backward-referenced statement, receive the values specified in the backward-referenced statement. (Default values in the GETPARM request are replaced by values from the corresponding fields in the referenced statement.)

Fields in the GETPARM request whose keywords have no match in the backward-referenced statement are not changed. An individual field for which a refkey is specified in the DISPLAY statement receives the value associated with the keyword referenced by that refkey in the backward-referenced statement.

In a DISPLAY statement a key may contain hyphens.

When the corresponding GETPARM request is displayed, the user can change any parameters a DISPLAY statement provides. When a DISPLAY statement is used by other statements to obtain values in a backward reference, the values obtained are those the user enters or modifies, rather than those the DISPLAY statement specifies.

2.5 ENTER

General Format:

```
[label:] ... ENTER {prname      } [[pfkey] [,]] [ key1 = {value1   } [, key2 = {value2   }] ... ]
                   {inner-label}                        {(refkey1)}          {(refkey2)}
                                                        (spec-label)
```

The ENTER statement is used to supply parameters for a GETPARM request without generating a workstation transaction. Executing an ENTER statement is the equivalent of either typing in values solicited by the GETPARM request and keying ENTER, pressing a legal PF key, or keying ENTER without modifying displayed default values. Each ENTER statement can satisfy one GETPARM request. ENTER is always part of the procedure step defined by the RUN statement it follows.

Values specified in an ENTER statement are supplied to the appropriate fields in the corresponding GETPARM request, overriding any defaults specified for those fields in the GETPARM prompt itself. Fields for which no values are provided in the ENTER statement retain the default values specified in the GETPARM prompt. If the parameter information supplied by ENTER is correct and completely satisfies the GETPARM request, then no workstation transaction occurs. If, however, the information provided by ENTER is illegal or inadequate, a request for correction is displayed at the workstation.

In an ENTER statement a key may contain hyphens.

An ENTER statement can use the label of an earlier ENTER or DISPLAY statement for backward reference. In this case, those fields in the GETPARM request associated with the current ENTER statement whose keywords correspond to keywords in the referenced statement receive values from the referenced statement. (Defaults specified in the GETPARM request are overridden.) Fields in the GETPARM request whose keywords have no match in the referenced statement are not changed. Any individual keyword for which a refkey rather than a value is specified in the ENTER statement, receives the value associated with the keyword referenced by that refkey in the backward-referenced statement.

If an erroneous ENTER statement results in a prompt requesting correction by the user, the values associated with each keyword in the ENTER statement are those modified by the user rather than those specified in the ENTER statement itself. Thus, when subsequent backward reference to this statement is made by other statements, the user-corrected values are obtained.

NOTE

The value of a PF key cannot be passed from one ENTER statement to another through backward reference. Each ENTER statement must specify its PF key value(s).

COBOL Considerations

The COBOL verbs ACCEPT and DISPLAY issue GETPARMS that can be satisfied through the procedure statement ENTER.

The prname associated with the GETPARM request issued by ACCEPT is ACCEPT. To supply values for an ACCEPT statement through an ENTER statement, the prname is ACCEPT, the keyword is the first 8 characters of the program-defined field name associated with the COBOL ACCEPT statement, and the value is any user-supplied value.

The prname associated with the GETPARM request issued by the COBOL DISPLAY statement is DISPLAY. The use of ENTER in this case restricts the DISPLAY from the workstation.

2.6 EXTRACT

Format 1:

[label:] ... EXTRACT variable = key1 [, variable = key2] ...

Format 2:

[label:] ... EXTRACT integer-variable = { BLOCKS ALLOCATED FOR / RECORDS USED BY } { filename [IN libname] [ON volname] / (spec-label) }

The EXTRACT statement is used to extract information from the system and store it in variables. The information can be one of the following:

- Items available from the EXTRACT SVC
- Either the number of blocks allocated for a file, or the number of records used in a file

The Format 1, the EXTRACT statement function is analogous to the EXTRACT SVC executed with the specified keywords. The keywords identifying fields for which data can be extracted are as follows:

CURLIB	PROGLIB	SYSLIB	FILECLAS	TAPEIO	USERNAME
CURVOL	PROGVOL	SYSVOL	FORM#	DISKIO	USERID
INLIB	RUNLIB	SYSWORK	LINES	PRINTIO	VERSION
INVOL	RUNVOL	WORKLIB	PRINTER	OTIO	TASK#
OUTLIB	SPOOLIB	WORKVOL	PRNTMODE	WSIO	WS
OUTVOL	SPOOLVOL	TASKTYPE	PRTCLASS		

The restrictions for the lengths of variables are as follows:

- If the receiving string-variable is too short to contain the data, the data is truncated from the right.
- If the receiving string-variable is longer than the data, the variable is blank filled on the right.
- Integer results are right justified, zero filled in the variable.

The procedure writer must declare all variables in the procedure. A string-variable can receive data for all the keywords (listed previously). An integer-variable can receive data for the following keywords:

TAPEIO	PRINTIO	WSIO	TASK#	LINES
DISKIO	OTIO	WS	FORM#	PRINTER

To extract the blocks allocated, the system sets the variable to the number of blocks allocated for the referenced file. If the file does not exist, the value -1 is assigned.

To extract the records used, the system sets the variable to the number of records used by the reference file. If the file does not exist, the value -1 is assigned.

In Format 2, if IN is not specified, OUTLIB is assumed. If ON is not specified, OUTVOL is assumed.

2.7 GOTO

General Format:

[label:] ... GOTO $\left\{\begin{matrix}\text{step-label} \\ \text{stmt-label}\end{matrix}\right\}$

GOTO can be used as a stand-alone statement and as a clause in an IF statement. When GOTO is used as a separate statement, it performs an unconditional branch to a specified statement. When used in an IF statement, it performs a branch conditioned by the result of the IF test.

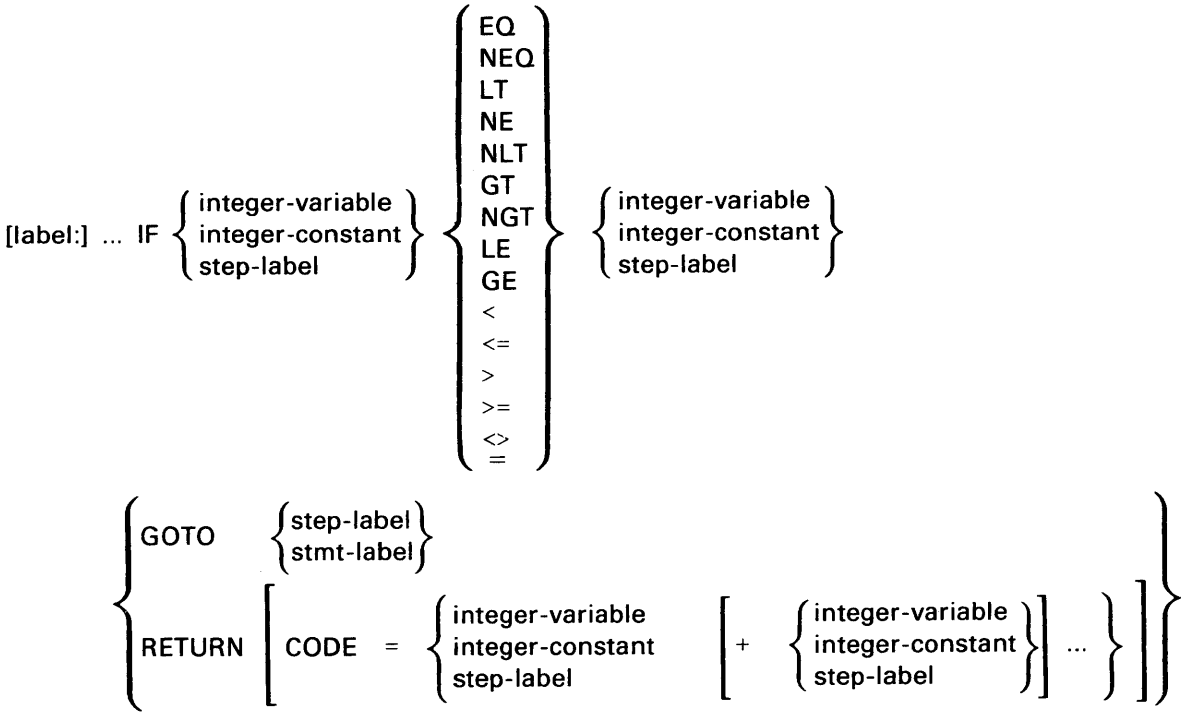
Multiple statements can be labelled by the same label. Therefore, the following rules are used to determine which statement is the target of the branch:

- The first occurrence of a label following the GOTO statement in the procedure text, if it exists, is the target.
- Otherwise, the closest preceding occurrence of the label preceding the GOTO statement in the procedure text, if it exists, is the target.
- If neither rule applies, then an error has occurred.

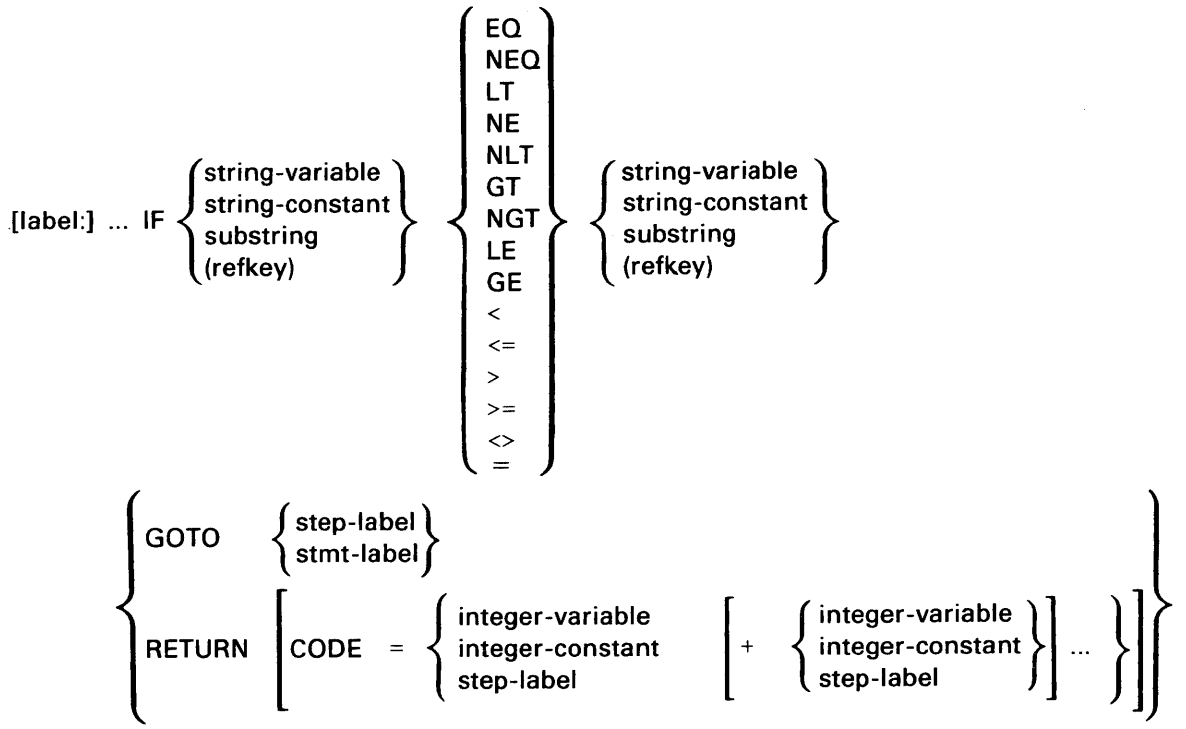
The user should note that a backward GOTO works differently from a backward reference. The backward GOTO references the *closest textual* occurrence of a label, while the backward reference references the *most recently executed* occurrence of a labelled DISPLAY or ENTER statement.

2.8 IF

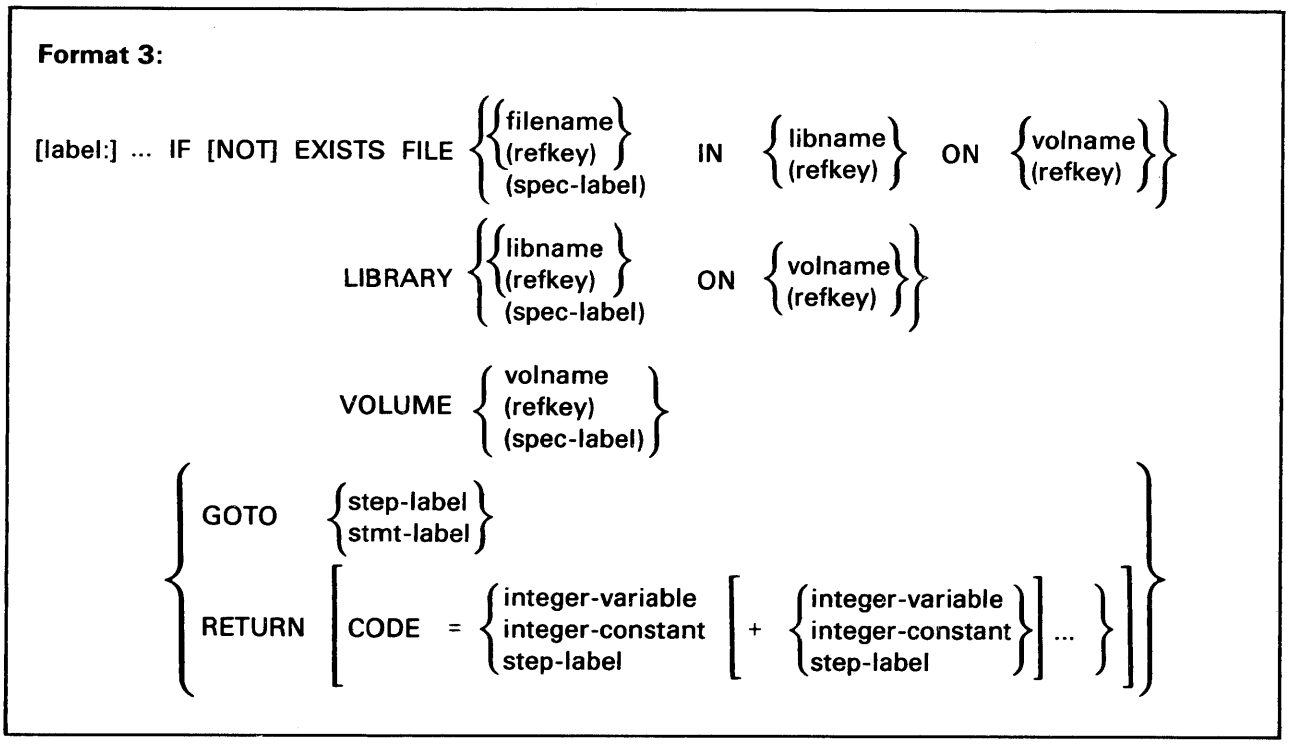
Format 1:



Format 2:



Format 3:



The IF statement is used to compare two operands and either branch to another statement or return to the invoker of the procedure if the condition is true. The comparisons are relational, i.e., equal, less than, greater than, not equal, not less than (greater or equal), and not greater than (less or equal). The comparisons can be numeric (among integer-variables, integer-constants, or step-labels), or string (among string-variables, string-constants, or substring).

The IF statement is used to test return code values and make decisions that affect the sequence of procedure step execution. The return code of the procedure step referenced by a step-label is compared, according to the specified relation, with an integer value the procedure writer furnishes. If the relation is true, the GOTO or RETURN clause is executed. Otherwise, the procedure continues with the next statement.

If the user specifies the GOTO clause, a branch is taken to the statement identified by step-label or stmt-label when the tested relation is true. If the user specifies the RETURN clause, the procedure is terminated when the tested relation is true. The CODE clause can be used to produce a nonzero return code upon procedure termination.

If the procedure is run from the Command Processor, this return code is displayed. If the procedure is run from another procedure, the return code can be used and tested by the outer procedure.

In Format 1, a numeric comparison is done.

In Format 2, a character comparison is done using the ASCII collating sequence. If the lengths of the operands are unequal, the comparison occurs as if the shorter operand is extended on the right with blanks until the length equals that of the longer operand.

In Format 3, if FILE is specified, and the file named by filename IN libname ON volname exists, then the associated GOTO or RETURN is executed. Otherwise, the next statement in the procedure is executed.

In Format 3, if LIBRARY is specified, and the library named by libname ON volname exists, then the associated GOTO or RETURN is executed. Otherwise, the next statement in the procedure is executed.

In Format 3, if VOLUME is specified, and the volume named by volname exists, then the associated GOTO or RETURN is executed. Otherwise, the next statement in the procedure is executed.

When NOT is specified in Format 3, the previous procedures for FILE, LIBRARY, and VOLUME are reversed.

In Format 3, a file, library, or volume does not exist if one of the following is true:

- The specified volume is being used exclusively by another user
- There is insufficient buffer space to perform the IF EXISTS operation
- There is a VTOC error
- A disk I/O error occurs
- The specified volume is not mounted
- The specified file and/or library cannot be found on the specified volume

2.9 LOGOFF

General Format:

[label:] ... LOGOFF

The LOGOFF statement terminates the user's current session. All programs and procedures whose execution is initiated by the user's current RUN command are terminated, their files are closed, and a Command Processor LOGOFF command is issued.

NOTE

When programs or procedures are run from the EDITOR or a menu, the LOGOFF statement returns control to the EDITOR or menu rather than cancelling the user's session. This feature is particularly useful when running programs in a test environment.

2.10 MESSAGE

General Format:

[label:] ... MESSAGE

```
{ [CENTER] [ [attribute] { string-constant       } [ , [attribute] { string-constant       } ] ... ] ; } ...
                         { integer-constant      }                 { integer-constant      }
                         { string-variable       }                 { string-variable       }
                         { integer-variable      }                 { integer-variable      }

  [CENTER] [ [attribute] { string-constant       } [ , [attribute] { string-constant       } ] ... ]
                         { integer-constant      }                 { integer-constant      }
                         { string-variable       }                 { string-variable       }
                         { integer-variable      }                 { integer-variable      }
```

The MESSAGE statement displays text on the workstation and continues execution of the procedure. The procedure writer can arrange variables, constants, and text strings on the screen using a simple statement format. The MESSAGE statement allows the procedure writer to specify any number of the following:

- Constants (either integer or string).
- Variables (either integer or string).
- End of line characters.
- Attributes for constants or variables. The attributes available are UPPER, UPLOW, NUMERIC, BRIGHT, DIM, BLINK, BLANK, and LINE. The user should note that UPPER, UPLOW, and NUMERIC are used only with variables that are to be modifiable.
- Variables (either integer or string) to be used to receive the value of PF key or ENTER.

The user can create up to 24 lines of 79 characters consisting of text and variable values intermixed. Lines longer than 79 characters are automatically truncated from the right. If the user creates more than 24 lines, an error occurs. A semicolon (;) is used to mark the end of the constants and variables which are to appear on one line.

If CENTER is not specified for a line, that line is left justified beginning in column 2. If CENTER is specified for a line, the text on that line is horizontally centered.

The MESSAGE statement produces lines that are centered vertically on the workstation. For example, if the user creates four lines then the lines are displayed on lines 11 through 14 on the workstation.

If attributes are specified, they apply only to the associated constant or variable. If conflicting attributes are specified (for example, both BRIGHT and DIM for the same field), the last specified attribute is used.

When at least one of two contiguous fields is specified with attributes, one space is introduced between them. When no attribute is specified for a field, the display is dim protect all noline. When neither of two contiguous fields is specified with attributes, no space is introduced between them.

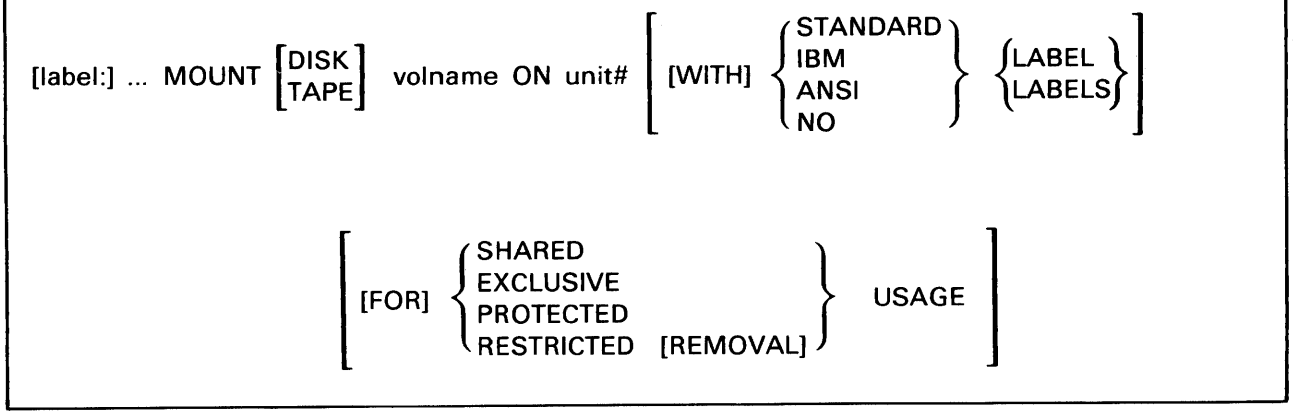
An integer-variable is displayed as a character string consisting of a minus sign if the integer is negative, followed by the significant digits in the integer. Zero is displayed as '0'.

When executed, the MESSAGE statement clears the current screen and displays the newly created screen, and execution of the procedure continues. The new screen remains until either another MESSAGE statement is executed, or until a RUN statement is executed. Prior to a RUN, the current screen is saved and, after the RUN returns, it is redisplayed on the workstation.

If MESSAGE is issued and the workstation is not available (the workstation is opened but not closed by other than the Procedure Interpreter) or the procedure is running in background, then the MESSAGE is not displayed and execution of the procedure continues.

2.11 MOUNT

General Format:



The MOUNT statement is used to mount a disk or tape volume. Its function is analogous to that of the MOUNT command on the Command Processor. The MOUNT statement is particularly useful for automating logical mounts for background processing and is used in conjunction with the DISMOUNT statement.

For disk mounting, the user can optionally specify a label type (STANDARD or NO) and a usage parameter (SHARED, EXCLUSIVE, PROTECTED, or RESTRICTED REMOVAL). Similarly, the user can optionally specify a label type (STANDARD, IBM, ANSI, or NO) and usage parameter (SHARED or EXCLUSIVE) for tape mounting. If the MOUNT statement is labelled, a return code equal to that supplied by the MOUNT SVC is associated with the label. (Refer to the appendix for the return code values.) This return code can be used for conditional branching by other procedure statements.

When mounting is performed through background processing, a display requesting the mount is forced at the highest priority Operator's Console that is available. (Refer to the *VS System Operation Guide.)* This display requests that the operator perform the physical mounting operation.

NOTE

Background processing jobs cannot mount a volume on the removable volume of a disk unit containing the system volume.

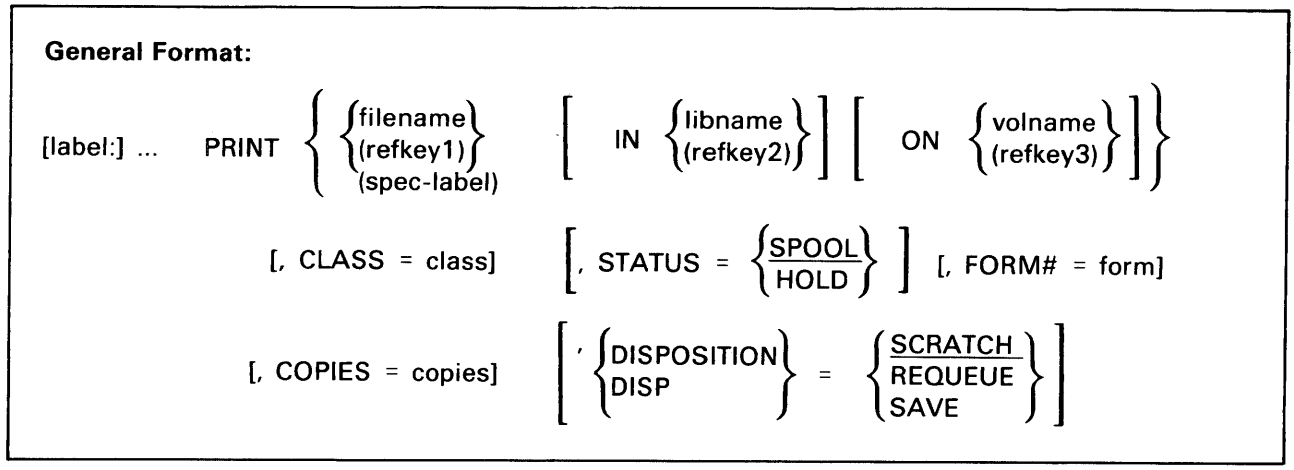
Unlike the MOUNT command issued interactively through the Command Processor, the MOUNT statement does not generate an error display if the specified volume cannot be mounted.

NOTE

The volume type (DISK or TAPE) is not optional for disk volumes named DISK or tape volumes named TAPE. For example, a disk volume named DISK is mounted in a procedure through the statement MOUNT DISK DISK ON unit#.

2.12 PRINT

General Format:



The PRINT statement permits a print file to be entered into the PRINT Queue from within a procedure. Its function is analogous to that of the PRINT option of the Manage FILES/LIBRARIES command on the Command Processor.

If the optional label is provided, a return code equal to that supplied by the PRINT SVC is associated with the label. This return code can be used for procedure statement conditional branching. If the library and volume for the print file are not specified, the user's default print library and volume are used. (The default print volume can be specified through the Command Processor or Procedure language SET statement).

In addition to the required parameters, the user can optionally specify the print class, status (HOLD or SPOOL), form number, number of copies, and disposition (SAVE, REQUEUE, or SCRATCH) of the print file. The user can also specify class, status, form number, copies, and disposition as string-variables.

2.13 PROCEDURE

General Format:

```
{PROC     }
{PROCEDURE}   [comment]
```

The PROCEDURE (or PROC) statement defines a procedure; everything following the letters PROCEDURE or PROC on the same line is interpreted as a comment. The line containing the letters PROCEDURE or PROC must precede any procedure statement but can itself be preceded by comment lines.

2.14 PROMPT

General Format:

[label:] ... PROMPT [PFKEY = variable]

```
{ [CENTER] [ [attribute] { string-constant  }                [ , [attribute] { string-constant  } ] ... ] ; } ...
                         { integer-constant }                                 { integer-constant }
                         { string-variable  }                                 { string-variable  }
                         { integer-variable }                                 { integer-variable }

  [CENTER] [ [attribute] { string-constant  }                [ , [attribute] { string-constant  } ] ... ]
                         { integer-constant }                                 { integer-constant }
                         { string-variable  }                                 { string-variable  }
                         { integer-variable }                                 { integer-variable }
```

The PROMPT statement is used to display information (variables and constants) on the workstation and then to accept data for variables. The PROMPT statement allows the procedure writer to specify any number of the following:

- Constants (either integer or string).
- Variables (either integer or string).
- End of line characters.
- Attributes for constants or variables. The attributes available are UPPER, UPLOW, NUMERIC, BRIGHT, DIM, BLINK, BLANK, and LINE. The user should note that UPPER, UPLOW, and NUMERIC are used only with variables that are to be modifiable.
- Variables (either integer or string) to be used to receive the value of PF key or ENTER.

The user can create up to 24 lines of 79 characters consisting of text and variable values intermixed. Lines that are longer than 79 characters are automatically truncated from the right. If the user creates more than 24 lines, an error occurs. A semicolon (;) is used to mark the end of the constants and variables which are to appear on one line.

If CENTER is not specified for a line, that line is left justified beginning in column 2. If CENTER is specified for a line, the text on that line is horizontally centered.

The PROMPT statement produces lines that are centered vertically on the workstation. For example, if the user creates four lines, then the lines are displayed on lines 11 through 14 on the workstation.

If the user specifies attributes, they apply only to the associated constant or variable. If conflicting attributes are specified (for example, both BRIGHT and DIM for the same field), the last specified attribute is used.

To make a field modifiable, the attributes UPPER, UPLOW, or NUMERIC must be specified. UPPER and UPLOW cannot be used on integer-variables. Only string-variables can be made modifiable.

If the same modifiable field is displayed more than once in a single prompt, e.g., PROMPT UPLOW &X, UPLOW &X, then the value supplied (by the user at the workstation) to the first field, i.e., the uppermost, leftmost, duplicate, and modifiable field on the screen, is used for the assignment.

Modifiable integer-variables display as eleven pseudoblanks. This allows room for the full range -2147483648 to 2147483647. Values entered outside of this range cause the field to blink and the workstation to beep. Data can be entered with leading or trailing signs but no space between sign and number.

When at least one of two contiguous fields is specified with attributes, one space is introduced between them. When no attribute is specified for a field, the display is dim protect all noline. When neither of two contiguous fields is specified with attributes, no space is introduced between them.

An integer-variable is displayed as a character string consisting of a minus sign if the integer is negative, followed by the significant digits in the integer.

To answer a PROMPT, the user presses either ENTER or a PF key. After the user presses a key, the screen is cleared automatically. Then the message "Procedure XXX In Progress" is displayed, and execution of the procedure continues.

When executed, the PROMPT statement clears the current screen and displays the newly created screen, and execution of the procedure continues. The new screen remains until either another PROMPT statement is executed, or until a RUN statement is executed. Prior to a RUN, the current screen is saved and after the RUN returns it is redisplayed on the workstation.

If a PROMPT is issued and the workstation is open (opened, but not closed by other than the Procedure Interpreter), an error is automatically issued. If a PROMPT is issued by a procedure running in background, an error is issued.

The user should note that there is no relationship between PROMPT and GETPARM, i.e., the procedure writer cannot satisfy a PROMPT from a procedure.

2.15 PROTECT

Format 1:

```
[label:] ... PROTECT { {filename}  [ IN {libname}  ]  [ ON {volname}  ] }
                     {(refkey1)}       {(refkey2)}       {(refkey3)}
                     {(spec-label)}
        TO  {[OWNER = owner]  [,PERIOD = period]  [,FILECLAS = fileclass]}
```

Format 2:

```
[label:] ... PROTECT LIBRARY { {libname}  [ ON {volname}  ] }
                             {(refkey1)}       {(refkey2)}
                             {(spec-label)}
        TO  {[OWNER = owner]  [,PERIOD = period]  [,FILECLAS = fileclass]}
```

The first format of the PROTECT statement is used to protect a file, and the second format is used to protect a library.

The PROTECT statement is used to modify the disk file or library protection information, and is analogous to the PROTECT option of the Manage FILES/LIBRARIES command on the Command Processor. If the optional label is provided, a return code equal to that supplied by the PROTECT SVC is associated with the label. (Refer to the appendix for the return code values.) This return code can be used for procedure statement conditional branching. If the optional library and/or volume (Format 1) or volume (Format 2) are not specified in the PROTECT statement, the OUTLIB and OUTVOL defaults from SET are used automatically.

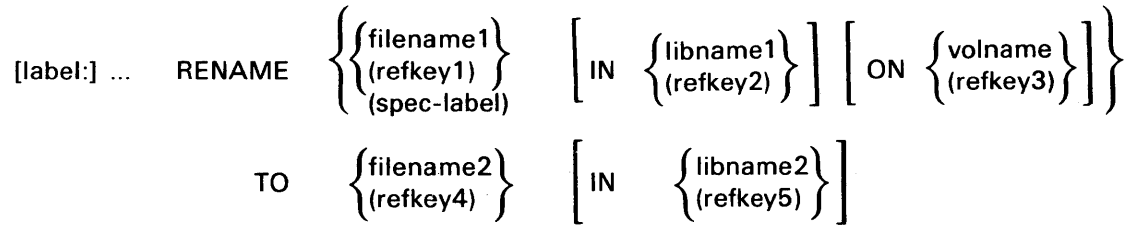
Unlike the PROTECT option of the Manage FILES/LIBRARIES command issued interactively through the Command Processor, the PROTECT statement does not generate an error display if the protection status of the file or library cannot be modified.

NOTE

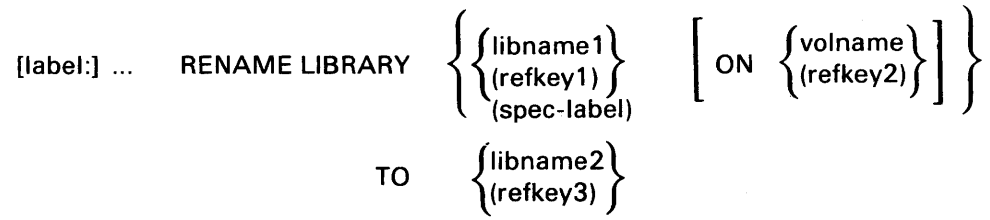
Due to syntax ambiguity, the PROTECT statement cannot be used to protect a file named LIBRARY.

2.16 RENAME

Format 1:



Format 2:



The first form of the RENAME statement is used to rename a disk file, and the second form is used to rename a library. RENAME is analogous to the RENAME option of the Manage FILES/LIBRARIES command on the Command Processor.

If the optional label is specified, a return code equal to that supplied by the RENAME SVC is associated with the label. (Refer to the appendix for the return code values.) This return code can be used for conditional branching within a procedure.

If the optional library and/or volume (Format 1) or volume (Format 2) are not specified through the RENAME statement, the OUTLIB and OUTVOL defaults from SET are used automatically.

Libname2 must follow the syntax rules for a VS library name. When libname2 is specified in Format 1, the old file and library name of the file are renamed to filename2 and libname2, respectively. The user should note that this is equivalent to moving a file from one library to another on the same volume.

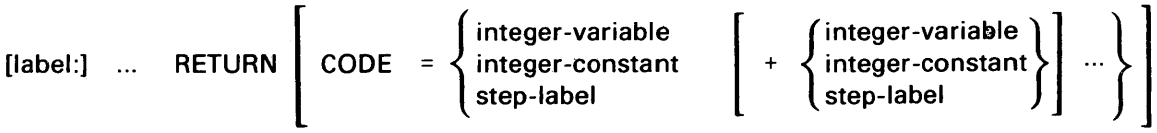
Unlike the RENAME option of the Manage FILES/LIBRARIES command issued interactively through the Command Processor, the RENAME statement does not generate an error display if the file cannot be renamed.

NOTE

Due to syntax ambiguity, the RENAME statement cannot be used to rename a file named LIBRARY.

2.17 RETURN

General Format:

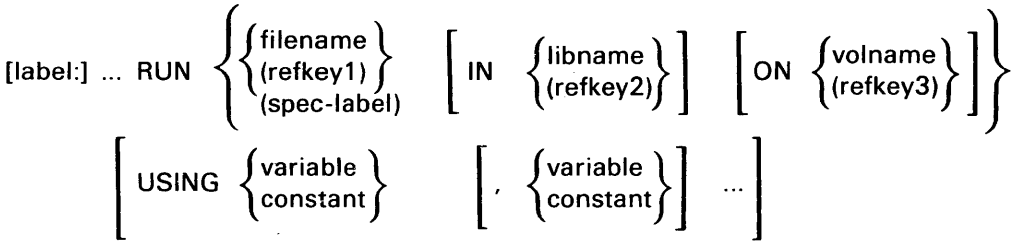


RETURN can be used either as a separate statement or as a clause in the IF statement. In either case, it unconditionally terminates procedure execution. It is not, however, required to terminate a procedure.

If the CODE clause is specified, a nonzero return code is returned upon procedure termination. The return code can be an integer value the procedure writer specifies or can be the return code a procedure step identified by step-label generates. If the procedure is run from the Command Processor, this return code (if nonzero) is displayed at the workstation upon procedure termination. If the procedure is run from another procedure, the return code is available to the outer procedure for testing in an IF statement.

2.18 RUN

General Format:



The RUN statement is used to run a program or procedure. Its function is analogous to that of the RUN command on the Command Processor.

If the IN phrase is specified, but the ON phrase is not, the specified file is searched for in the specified library on PROGVOL. If that file does not exist, the specified file is searched for in the specified library on SYSVOL. If that file does not exist, an error is issued.

If the ON phrase is specified, but the IN phrase is not, the specified file is searched for in PROGLIB on the specified volume. If that file does not exist, the specified file is searched for in SYSLIB on the specified volume. If that file does not exist, an error is issued.

The RUN statement defines a procedure step. The specification statements, DISPLAY and ENTER, can be used to supply parameters needed by a program run with a procedure RUN step. All DISPLAY and ENTER statements following a RUN statement, up to the first statement other than DISPLAY or ENTER, are part of the RUN step, and are associated with GETPARM requests issued by the program being run.

If the spec-label (or refkey) is specified in place of the name and/or location of a file, the DISPLAY or ENTER statement identified by this label is used to supply the required file parameters through backward reference.

USING allows the programmer to pass parameters by reference to a program or another procedure. When a constant is passed, a copy is made and a pointer to the copy is passed.

2.19 SCRATCH

Format 1:

```
[label:] ... SCRATCH { { filename  }          [ IN { libname  } ]   [ ON { volname  } ] }
                     { { (refkey1) }                { (refkey2) }         { (refkey3) }
                     {   (spec-label)                                                   }
```

Format 2:

```
[label:] ... SCRATCH LIBRARY { { libname  }          [ ON { volname  } ] }
                             { { (refkey1) }               { (refkey2) }
                             {   (spec-label)                          }
```

Format 1 is used to scratch a disk file, and Format 2 is used to scratch a library. SCRATCH is analogous to the SCRATCH option of the Manage FILES/LIBRARIES command on the Command Processor.

If the optional label is used, a return code equal to that supplied by the SCRATCH SVC is associated with the label. (Refer to the appendix for an explanation of the return codes.) This return code can be used for conditional branching by other procedure statements. If the optional library and/or volume are not specified in the SCRATCH statement, the OUTLIB and OUTVOL defaults from SET are used automatically.

Unlike the SCRATCH option of the Manage FILES/LIBRARIES command issued interactively through the Command Processor, the SCRATCH statement does not generate an error display if the file cannot be scratched.

NOTE

Due to syntax ambiguity, the SCRATCH statement cannot be used to scratch a file named LIBRARY.

2.20 SET

General Format:

[label:] ... SET { setkey1 = value1 [,setkey2 = value2] ... }

The SET statement is used to specify default names for commonly used libraries and volumes, to specify the default file class, and to select print mode and job submittal options. Default values set through the SET statement are used automatically as defaults in all subsequent procedure statements.

The SET statement is analogous to the SET Usage Constants command on the Command Processor. All parameters specifiable with the SET statement can, with the exception of PROGLIB and PROGVOL, also be specified with the SET Usage Constants command. For a discussion of these parameters, refer to the *VS Programmer's Introduction.*

PROGLIB and PROGVOL are unique to the SET statement. They identify the library and volume that are to serve as the default program library and volume for all programs the procedure runs. (RUNLIB and RUNVOL identify the default program library and volume to be used for programs run with the RUN command from the Command Processor.) If PROGLIB and PROGVOL are not specified, the library and volume in which the procedure itself is located serve as the default program library and volume for all programs run by the procedure.

Setkey is a keyword identifying a field for which a default parameter value is to be specified. The legal keywords are as follows:

INLIB	PROGLIB	PRNTMODE	JOBQUEUE
INVOL	PROGVOL	PRINTER	JOBCLASS
OUTLIB	WORKVOL	PRTCLASS	JOBLIMIT
OUTVOL	SPOOLIB	FORM#	
RUNLIB	SPOOLVOL	FILECLAS	
RUNVOL	PRTFCLAS	LINES	

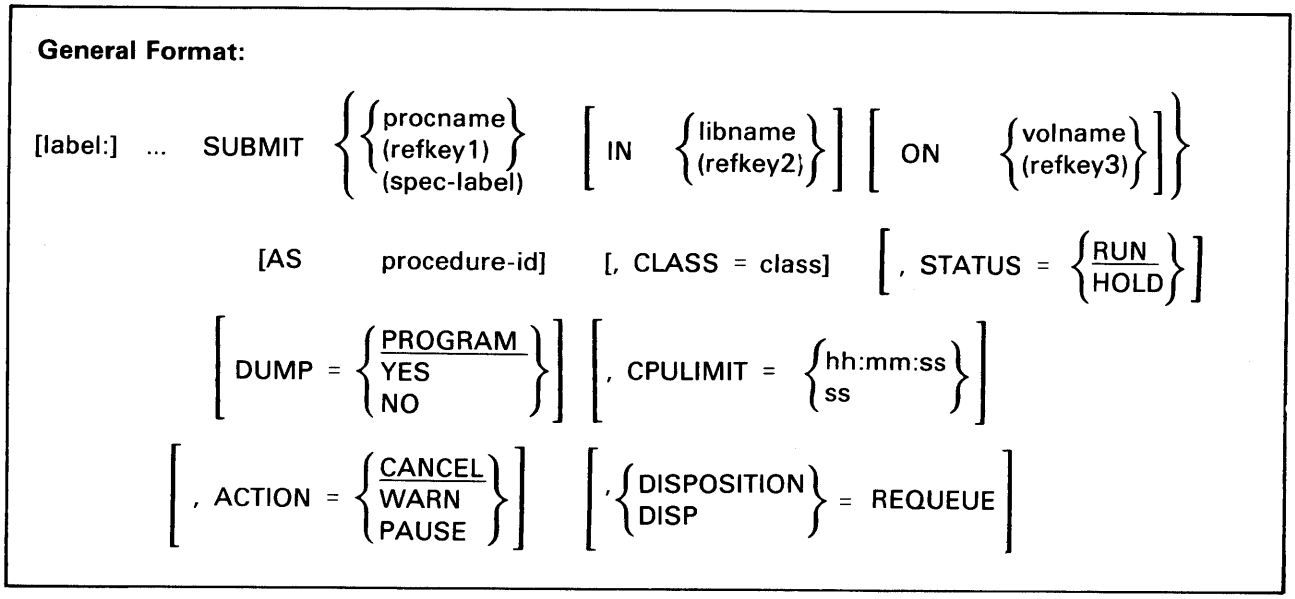
All parameters except PROGLIB and PROGVOL become default parameters for the remainder of the user's session, remaining in effect after the procedure is terminated.

NOTE

The system defines the work library default name. The default name for this library *cannot* be modified by the user with a SET statement. For this reason, WORKLIB is not listed as a legal keyword even though it appears in the File Defaults display of the SET Usage Constants command of the Command Processor.

2.21 SUBMIT

General Format:



The SUBMIT statement is used to submit a procedure into the PROCEDURE Queue for execution on a noninteractive basis. Its function is analogous to that of the SUBMIT command on the Command Processor.

If the optional label is provided, a return code equal to that supplied by the SUBMIT SVC is associated with the label. This return code can be used for procedure statement conditional branching. (Refer to the appendix for the return code values.)

The SUBMIT statement allows the library and/or volume to be specified. If the IN phrase is specified, but the ON phrase is not, the specified file is searched for in the specified library on PROGVOL. If that file does not exist, the specified file is searched for in the specified library on SYSVOL. If that file does not exist, a return code is issued.

If the ON phrase is specified, but the IN phrase is not, the specified file is searched for in PROGLIB on the specified volume. If that file does not exist, the specified file is searched for in SYSLIB on the specified volume. If that file does not exist, a return code is issued.

If neither the IN phrase nor the ON phrase is specified, the specified file is searched for in PROGLIB on PROGVOL. If it is not found, SYSLIB and SYSVOL are used as inputs to the SUBMIT SVC.

Class, status, dump, cpulimit, action, and disposition can be specified as string-variables.

In addition to the required parameters, the user can optionally specify a number of characteristics: a queueing status for the submitted procedure (RUN or HOLD); whether or not a program dump is produced on abnormal termination (PROGRAM, YES, or NO; PROGRAM indicates program default); a CPU time limit, an action to be taken if the CPU time limit is exceeded (CANCEL, WARN, or PAUSE, valid only when the CPU time limit is specified and is nonzero); and whether or not the procedure is to be queued again upon completion.

The maximum CPU processing time allowed can be specified in hours, minutes, and seconds, or in seconds alone. The values specified must fall within the following ranges: hours: 0 – 99; minutes: 0 – 59; seconds: 0 – 59; seconds only: 0 – 359999.

NOTE

If an ACTION is specified and the CPULIMIT is 0 or is not specified, the procedure cancels and displays an error message.

2.22 USING

General Format:

```
USING  variable [, variable] ... AS { STRING (n) }
                                    { INTEGER    }
     [ , variable [, variable] ... AS { STRING (n) } ] ...
                                      { INTEGER    }
```

The USING statement declares the formal parameters to a procedure. This statement is optional. If the statement is present, it must immediately follow the PROCEDURE or PROC statement.

The USING statement defines the parameters passed to the procedure by reference. Any program that uses appropriate types and lengths of parameters can call to a procedure and pass data.

The number of parameters passed must equal the number of parameters declared in the USING statement.

The value n is an integer between 1 and 256, inclusive.

APPENDIX
VS SYSTEM UTILITIES AND PROCEDURE LANGUAGE STATEMENTS RETURN CODE VALUES

All programs run on the VS generate return codes which, if other than zero, can be displayed on the workstation after program completion. Some system utility programs, along with the procedure statements DISMOUNT, MOUNT, PROTECT, RENAME, SCRATCH, and SUBMIT generate return codes that indicate success or failure and can indicate the type of failure that occurred. These non-zero return codes and their meanings are listed below.

VS SYSTEM UTILITIES

Program Name	Return Code	Meaning
BACKUP		Number of errors found
COPY	100	No copy took place
	104	Some program privileges lost
	108	Disk error: run LISTVTOC
	112	No space in output volume
	116	I/O error on output
	120	Boundary violation
	124	Key out of sequence
	128	Duplicate key
	132	The primary extent was exceeded
	136	There was a DMS error
	140	Duplicate file name encountered
	144	Copy completed in I/O mode after primary extent was exceeded
DISKINIT	4	Termination by user
	8	Insufficient space in I/O buffer
	12	Mount operation unsuccessful
	16	Bad disk sector encountered
	20	Bad MOUNT SVC return code
	24	Bad FREEBUF return code
	28	Bad XIO return code
	32	Disk I/O error
EZFORMAT	1-4	Warning; program will probably run
	5-7	Severe error; program will not run correctly
	8-16	Fatal error; object code not generated

VS SYSTEM UTILITIES

Program Name	Return Code	Meaning
LINKER	4	Either unsatisfied external reference(s), or labeled COMMON sections of unequal lengths encountered (FORTRAN only)
	8	1. Multiple entry points with the same name encountered
		2. Static section in segment 0 address space encountered
		3. Initialized blank COMMON static section encountered (FORTRAN only)
		4. Illegal replacement attempted (possible only in object programs generated in FORTRAN). This error occurs when a code or static section and a labeled or blank COMMON section of the same name are encountered
LISTVTOC	4	Extent lost on disk
	8	Error on file label
	12	Error in library directory
	16	Error in VTOC
SORT	4	No records to be sorted: input records did not meet selection criteria, or input file specified by user was empty
	8	Insufficient space in stack or I/O buffer
	12	Record size is more than 1024 bytes long
	16	Invalid sort key
	20	Unexpected program check
	24	Input records out of order in file to be merged; program cannot proceed
All compilers	1-4	Warning
	5-8	Severe error; program will not execute correctly
	9-16	Fatal error; program will not execute

Statement Name	Return Code	Meaning
DISMOUNT	4	Input volume name is blank, or bytes 0 - 1 in the input are nonzero
	8	Volume cannot be found
	12	Volume cannot be dismounted
	16	Device detached
	20	Volume in use by a user or the operating system
	24	Volume reserved by another user
	28	GETMEM (SVC) pool failure
	32	Device is reserved by another task
MOUNT	4	Successful mount, but new volume label type does not agree with the input parameters
	8	Sucessful mount, but the new volume name is not the volume name requested
	12	Disk or tape I/O error detected while reading the new volume label, or the new volume has a bad VTOC. VCBSER is set to blank
	16	Device not a disk or tape, or the device number is not valid
	20	Device detached
	24	Volume type (REM or FIX) not found
	28	Request to mount unlabelled volume on disk other than diskette
	32	Input volume name is blank
	36	Volume already mounted. Also set for a duplicate volume name
	40	Volume is currently in use by a user or the operating system
	44	Volume reserved by another user for exclusive use
	48	I/O buffer is insufficient to perform the mount
	52	Unable to allocate space for tape I/O control blocks
	56	Invalid request: work and/or spool filing requested in a nonlabeled volume
	60	Invalid request: nonstandard addressing attempted with standard label option or on a hard-sectored device
	64	Wrong media: soft-sectored diskette inserted into a device for hard-sectored diskettes only
	68	Wrong media: hard-sectored diskette inserted into a device for soft-sectored diskettes only
	72	Wrong media: hard-sectored diskette inserted for a nonstandard addressing request
	76	Wrong addressing mode: caller requested MOUNT for standard addressing but diskette is nonstandard addressing
	80	Device reserved by another user
	84	PF16 was entered when the mount message was displayed

PROCEDURE LANGUAGE STATEMENTS

Statement Name	Return Code	Meaning
PROTECT	4	Volume not mounted
	8	Volume used exclusively by other user
	12	All buffers in use, no protection change
	16	Library not found
	20	File not found
	24	Update access denied, no protection change
	28	(Unused)
	32	File in use, no protection change
	36	VTOC error
	40	VTOC error
	44	Invalid argument list address
	48	I/O error; VTOC unreliable
	52	Open or protected files bypassed in protecting library
	56	Invalid new protection data
RENAME	4	Volume not mounted
	8	Volume used exclusively by other user
	12	All buffers in use, no rename
	16	Library not found
	20	File not found
	24	Update access to file protection class denied, no rename
	28	Unexpired file, no rename
	32	File in use, no rename
	36	VTOC error
	40	VTOC error
	44	Invalid argument list address
	48	I/O error. VTOC unreliable
	52	New filename or library name already exists, no rename
	56	New filename invalid (or first character #), no rename
	60	VTOC is currently full
	64	Reserved bits in the parameter list options byte are nonzero
SCRATCH	4	Volume not mounted
	8	Volume used exclusively by other user
	12	All buffers in use, no scratch
	16	Library not found
	20	File not found
	24	Update access to file protection class denied (single-file scratch only)
	28	Unexpired file, no scratch (single-file scratch only)
	32	File in use, no scratch
	36	VTOC error
	40	VTOC error
	44	Invalid argument list address
	48	I/O error; VTOC unreliable
	52	Open, protected, and/or unexpired file(s) bypassed in scratching library

Statement Name	Return Code	Meaning
SUBMIT	4	Volume not mounted
	8	Volume in exclusive use
	12	All buffers in use; unable to perform verification
	16	Library not found
	20	File not found
	24	Improper file type
	28	File access denied
	32	VTOC error
	36	VTOC error

DOCUMENT HISTORY

Summary of Changes for the Third Edition

TYPE	DESCRIPTION	AFFECTED PAGES
New Statements	PRINT statement; permits a print file to be queued from within a procedure.	1-4, 2-11
	SUBMIT statement; permits a procedure to be queued for noninteractive processing from within a procedure.	1-4, 2-21, 2-22
New Features	Enhancement to SET statement; addition of JOBCLASS, JOBQUEUE, and JOBLIMIT.	1-5, 2-20
	Enhancement to CODE clause of RETURN and IF statements; allows a series of values to be added to an initial CODE value.	2-17
	Modified syntax in DISMOUNT statement; requires keyword TAPE for tape volumes.	2-3
Miscellaneous	For a listing of system utility return codes, refer to the *VS Utilities Reference* (800-1303UT).	

DOCUMENT HISTORY

Summary of Changes for the Second Edition

TYPE	DESCRIPTION	AFFECTED PAGES
New Statements	MOUNT statement; permits mounting of disk and tape volumes from within a procedure.	4, 16, 38
	DISMOUNT statement; permits dismounting of disk and tape volumes from within a procedure.	4, 16, 31
New Features	Background processing; provides ability to run procedures as background jobs.	26–27
	Enhancement to SET statement; addition of the LINES options.	5–7, 47

GLOSSARY

backward GOTO — The backward GOTO references the closest textual occurrence of a label.

backward reference — A backward reference enables a procedure statement to obtain some or all of its required parameters from a preceding, previously executed DISPLAY or ENTER statement.

comment — Any user-written message. The Procedure Interpreter interprets comments as blanks.

fileclass — The protection class of the specified file or library. A protection class must be one of the following: A – Z, #, $, ɸ, or @.

filename — The name of the program or procedure file to be acted upon by the current statement.

GETPARM — A request for parameter information by a program or procedure.

inner-label — The label of an ENTER or DISPLAY statement in the inner procedure of two nested procedures, which functions as the prname of an ENTER or DISPLAY statement in the outer procedure. An inner-label, therefore, passes parameter values to the inner statement from the outer statement.

integer-constant — A whole number in the range -99999999 to 99999999.

keyword — Identifies a field in the GETPARM request, the value of which is supplied or solicited by the current ENTER or DISPLAY statement.

label — An optional name up to eight characters in length that identifies a procedure statement for reference by other statements. A label must begin with an alphabetic character and must be followed by a colon (:), except when a step-label is referenced in an IF, GOTO, or ASSIGN statement, or used in a refkey.

libname — The name of the library containing the file to be operated upon, or the name of the library to be operated upon.

operand — Data that is to be operated upon by the current procedure statement.

owner — The one to three-character user ID that identifies the owner-of-record of the specified file or library. Must be alphanumeric, begin with an alphabetic character, and contain no embedded spaces.

period — The retention period in days (0 – 999) for the specified file or library.

pfkey — A Program Function key (1 – 32) for the GETPARM request.

prname — A parameter reference name is a parameter solicited by a GETPARM request.

program library The library containing user-written files.

refkey A special type of keyword identifying a field in a previous ENTER or DISPLAY statement whose value is to be obtained for the current field through backward reference. A refkey consists of the label of the referenced specification statement, followed by a period, and followed by the keyword identifying the referenced field. Refkeys must be enclosed in parentheses.

setkey A keyword identifying a field for which a default parameter value is to be specified. The legal keywords are as follows:

INLIB	PROGVOL	PRTFCLAS
INVOL	WORKVOL	FORM#
OUTLIB	SPOOLIB	FILECLAS
OUTVOL	SPOOLVOL	LINES
RUNLIB	PRNTMODE	JOBQUEUE
RUNVOL	PRINTER	JOBCLASS
PROGLIB	PRTCLASS	JOBLIMIT

spec-label The label of a previous specification statement (ENTER or DISPLAY) from which parameters are to be obtained through backward reference for use in the current statement.

step-label The label of a DISMOUNT, MOUNT, PRINT, RUN, SCRATCH, SET, SUBMIT, RENAME, or PROTECT statement. Used in IF and RETURN statements to identify the return code to be tested.

stmt-label The label of a MESSAGE, PROMPT, EXTRACT, DECLARE, ASSIGN, SET, LOGOFF, IF, GOTO, or RETURN procedure statement.

string-constant A string of text of from 1 - 256 characters enclosed in quotes.

substring A portion of a string-variable represented by the character position defined by "start" for a specified length. Substrings are allowed wherever a string-variable is allowed, except for the following statements: DECLARE, USING, RUN USING (as a parameter).

system library The library containing system program files (@SYSTEM@).

unit# The number of the device on which a volume is being mounted.

value A user-specified value to be supplied to the keyword, which is legal for that particular application.

variable A string of from 2 to 31 characters. The first character must be an ampersand (&). The other characters can be chosen from the characters A - Z, a - z, 0 - 9, @, $, #, and _. Variables can be either uppercase or lowercase. Lowercase characters are converted internally to uppercase.

volname The name of the volume on which the specified library is located, or the name of the volume that is being mounted or dismounted.

INDEX

WANG

Customer Comment Form

Publications Number 800-1205PR-04

Title VS PROCEDURE LANGUAGE REFERENCE

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other _____

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ **Street** _____

Title _____ **City** _____

Dept/Mail Stop _____ **State/Country** _____

Company _____ **Zip Code** _____ **Telephone** _____

Thank you for your help.

Printed in U.S.A. 14-3140 3-82-5C

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1369
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-1205PR-04
11-82-10M