SYSTEM/32

IBM System/32
System Control Programming
Reference Manual

IBM System/32 Programming Information

GC21-7593-4
File No. S32-36

Program Number
5725-SC1

IBM System/32
System Control Programming
Reference Manual

Fourth Edition (May 1977)

This is a major revision of, and obsoletes, GC21-7593-2 and Technical Newsletter GN21-7879. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition. Significant additions are in Part 5, *System Configurations, Modification, and Installation*; these additions include system control programming support for three program products: FORTRAN IV, Basic Assembler, and the File Conversion Utility. In addition, system control programming now supports the 1255 Magnetic Character Reader attachment, Word Processing Communications utility, overlay linkage editor, and queued job stream. Enhancements to system control programming include the compress function of $MAINT utility program (CONDENSE procedure), the Queued Job Stream Card-to-Library utility program ($QJOB), and the JOBSTR and APCHANGE procedures. A new appendix, Appendix H, *System Sharing,* has been added showing examples of sharing a System/32 and the procedures recommended for each method. Miscellaneous changes and additions are not extensive.

This edition applies to version 6, modification 0 of IBM System/32 (Program 5725-SC1), IBM System/32 Utilities Program (Program Products 5725-UT1 and 5725-UT2), IBM System/32 RPG II (Program Product 5725-RG1), IBM System/32 FORTRAN IV (Program Product 5725-FO1), IBM System/32 Basic Assembler (Program Product 5725-AS1); and to all subsequent versions and modifications unless otherwise indicated in new editions or technical newsletters. Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest *IBM System/32 Bibliography,* GC20-0032, for the editions that are applicable and current.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. Use this publication only for the purposes stated in the *Preface.*

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

This reference manual provides system programmers with information needed to establish administrative and operating procedures for an IBM System/32. Information is provided for programmers to run application programs on IBM System/32 and use the system procedures and utility programs provided with IBM System/32.

This manual contains:

- A summary of IBM System/32 operation control language (OCL) statements and a detailed description of each OCL statement
- A general description of IBM System/32 system procedures and a detailed description of each procedure. A detailed description of the command statements that evoke the procedures and a summary of command statement formats
- A description of how to use OCL statements to create data files and run application programs. An example of how to use OCL statements and procedures to run applications
- A description of each system utility program provided with IBM System/32 and a description of associated utility control statements
- A description of how to create, install, and modify IBM System/32 system control programming and how to install IBM System/32 program products

Appendixes describe:

- The relationship of disk records, blocks and sectors
- Decimal and hexadecimal conversion
- Diskette data formats for IBM System/32
- The IBM service procedures
- The OCL and utility control statements contained in the system procedures
- Standard characters for IBM System/32 printers
- Polling and address characters for IBM System/32 tributary stations
- System sharing examples

A glossary at the back of the manual defines data processing terms used in the manual. New terms in the manual are italicized the first time they are used.

Note: This manual follows the convention that *he* means *he or she.*

Prerequisite Publication

IBM System/32 Introduction, GC21-7582, provides an overview of the system and its characteristics

Related Publications

IBM System/32 Operator's Guide, GC21-7591, provides detailed instructions for operating IBM System/32

IBM Diskette General Information Manual, GA21-9182 , describes the diskette data format for basic data exchange

IBM System/32 SCP Command Statement Reference Summary, GX21-7687, provides a brief description and the format of command statements used for system functions.

Word Processor/32 Installation and Procedures Manual, SH30-0114, provides instructions for installing the Word Processing/32 Program Product and shows basic typing and work flow procedures for various applications of the program product.

Titles and abstracts of related publications are listed in the *IBM System/32 Bibliography,* GC20-0032.

Contents

This page intentionally left blank

List of Abbreviations and Acronyms

The following abbreviations and acronyms are used in the text of this manual.

BSC	Binary synchronous communication
BSCA	Binary synchronous communications adapter
CE	Customer engineer
DTF	Define the file
EBCDIC	Extended binary coded decimal interchange code
I/O	Input/output
IOB	Input/output block
IOS	Input/output supervisor
IPL	Initial program load
K	1024 bytes
MIC	Message identification code
MICR	Magnetic ink character reader
MRJE	MULTI-LEAVING remote job entry
MRJE/WS	MULTI-LEAVING remote job entry work station
OCL	Operation control language
PID	Program information department
PLCA	Program level communication area
PTAM	Pseudo tape access method
PTF	Program temporary fix
RIB	Request indicator byte
SCA	System communication area
SCP	System control programming
SDLC	Synchronous data link control
SIS	Scientific instruction set
SNA	Systems network architecture
SVC	Supervisor call
SWA	Scheduler work area
VTOC	Volume table of contents

How to Use This Manual

This manual has five parts. Part 1 describes operation control language (OCL) statements. Part 2 describes system procedures and command statements. Part 3 describes the OCL and procedures to use applications. Part 4 describes system utility programs. Part 5 describes system configuration, installation, modification, and program product installation.

Part 1

Refer to part 1 if you want to know:

- What an OCL statement is
- What each OCL statement is used for and when it is needed
- Where each OCL statement is placed in relation to others
- How each statement must be coded
- What each statement must contain

Part 2

Refer to part 2 if you want to know:

- What a procedure is
- What a command statement is and how it is used
- How to create, evoke, or modify a procedure
- What procedures are supplied with IBM System/32 and the function of each
- The format and contents of the command statements that evoke the procedures supplied with IBM System/32

Part 3

Refer to part 3 if you want to know:

- How to use OCL to build disk files and to load and run programs
- How to use OCL and procedures to perform applications

Part 4

Refer to part 4 if you want to know:

- What system utility programs are supplied with IBM System/32 system control programming
- What the function of each utility program is
- What OCL statements and utility control statements are necessary to load and run each utility program

Part 5

Refer to part 5 if you want to know about:

- Configuration and installation of IBM System/32 system control programming at initial system installation or subsequent system update
- Installing IBM System/32 program products (and verifying that they are installed correctly)
- Modifying an installed system by deleting certain system control programming components or program product functions from the library

Reader's Comments

If you find an error, please tell us about it by using the Reader's Comment Form at the back of this publication.

Part 1

OCL Statements

WHAT IS OCL?

The IBM System/32 system control programming (SCP) controls program execution. The SCP must be in main storage before your programs can be run. It is located on the disk and is brought into main storage by a process called *initial program load (IPL),* which is performed by the operator after the system power is turned on.

Operation control language (OCL) is your means of communicating with the SCP. Every job requires OCL statements identifying a job and describing that job's requirements to the SCP. OCL statements for a job can be stored together as a set, called a *procedure*, and can be stored in and evoked from the *system library.*

The system library is contained in a disk file named #LIBRARY. Besides areas required by the SCP, the system library contains:

- *Load members:* A load member is a collection of instructions that can be loaded directly into main storage for execution.

- *Procedure members:* A procedure member is a collection of related OCL statements. Procedures can also contain *utility control statements,* statements required by the system utilities (see index entry: *writing utility control statements* for more information on utility control statements).

- *Source members:* A source member is a collection of records used as input to a program. For example, RPG II specifications and sort sequence specifications can be stored in source members. Source members cannot, however, contain data to be processed.

- *Subroutine members:* Subroutine members contain subroutines that can be combined with user and system control programs for execution.

You can enter OCL statements in two ways: (1) key OCL statements to create a procedure stored in the library, then evoke the entire procedure when those OCL statements are required; (2) key the OCL statements at the time the system requires them.

OCL STATEMENTS AND THE JOB

To run a job, the necessary OCL statements must be supplied from the keyboard or called from the system library. To call OCL statements (procedures) from the system library, enter an INCLUDE OCL statement. A simplified form of the INCLUDE statement is called a *command statement.* Command statements make it easier to call procedures (see index entry: *command statements*).

When *system utility programs* (programs that perform a variety of routine tasks to keep the system and files in order) are to be run, *utility control statements* may be needed in addition to OCL statements. *Utility control statements* pass information, such as filenames, to utility programs. Utility control statements can be included with OCL statements in procedures.

OCL statements, utility control statements (when required), and data, form the *job stream.*

SYSTEM CONFIGURATION

IBM System/32 system control programming runs on all models of System/32 and is compatible with all available System/32 features.

TYPES OF INFORMATION CONVEYED IN OCL STATEMENTS

OCL statements contain two types of information, an *identifier* and *parameters.* An identifier distinguishes one OCL statement from another; a parameter supplies information to a program. Figure 1 shows the general form of OCL statements.

```
// IDENTIFIER Parameter-1,Parameter-2,...,Parameter-n
```

Figure 1. General Form of OCL Statements

Identifiers

Every OCL statement except a command statement requires a statement identifier. A command statement uses a procedure name. Command statements are discussed in Part 2 of this manual.

Most OCL statements begin with //. OCL statement identifiers that require // are:

COMPILE	FORMS	LOAD	PAUSE	SYSLIST
DATE	IMAGE	LOG	RUN	* (message)
FILE	INCLUDE	MEMBER	SWITCH	

For example, in the statement

```
// LOAD $COPY
```

the statement identifier is LOAD.

Identifiers that do not require // are:

```
* (comment)

/* (end of data)
```

For example, in the statement

```
* END OF JOB
```

the statement identifier is *. Because // does not precede the *, the * indicates the statement is a comment. (// * at the beginning of a statement indicates the statement is a message.)

Also, // is not required with a command statement (a simplified form of the INCLUDE OCL statement).

Parameters

Parameters are either *symbolic* or *keyword* parameters. In the following statement, $COPY is a *symbolic parameter*—the name of a system utility program:

```
// LOAD $COPY
```

NAME-COPYIN, UNIT-F1, and LABEL-filename are *keyword parameters* in the following statement:

```
// FILE NAME-COPYIN,UNIT-F1,LABEL-filename
```

A keyword parameter contains a *keyword* (NAME, UNIT, and LABEL are the keywords in the preceding OCL statement) that distinguishes the parameter from other parameters, just as statement identifiers distinguish one OCL statement from another. In addition to a keyword, a keyword parameter usually contains a *value* (COPYIN and F1, are values in the preceding sample OCL statement).

GENERAL OCL CODING RULES

OCL statement formats described in this manual can include special characters, such as //, and words written in capital letters, such as the FILE statement parameter, LABEL. These special characters and words must be entered exactly as shown in the statement descriptions given in this manual. Words written in lowercase letters, such as filename, represent information that you must supply. OCL statements cannot exceed 120 characters, except the FILE statement. (See *Continuation* description on the following page.)

Additional coding rules are:

- The first character (* or /) of an OCL statement must be keyed in position 1. For example, // must be entered in positions 1 and 2.
- One or more positions must be blank between the // and the statement identifier. For example:

```
// LOAD
// *
```

- One or more positions must be blank between the statement identifier and the first parameter. For example:

```
// LOAD $COPY
// * 6666
```

- If you need to include more than one parameter, use a comma to separate them. No blanks are allowed within or between parameters. Anything following the first blank after a parameter is considered a comment (see index entry: *comments*).
- If you are writing keyword parameters, place the keyword first and use a hyphen (-) to separate the keyword from the value.

Continuation

Expressing a single statement in two or more records is called *continuation.* The only OCL statement that can use continuation is the FILE statement. (See index entry: *// FILE statement* for a description of FILE statements.)

A record can consist of a maximum of 120 characters, including blanks and commas, when expressing an OCL statement. Because of the many parameters possible in FILE statements, FILE statements can be composed of more than one record to express a single FILE statement. All other OCL statements must not exceed one record.

Rules for using continuation are:

- Place a comma after the last parameter in every record except the last. The comma, followed by a blank, tells the system that the statement is continued in the next record.
- Begin each new record with // in positions 1 and 2.
- Leave one or more blanks between the // and the first parameter in the record.

In the first of the following two examples of continued FILE statements, five records are used to express a single FILE statement. In the second example, two records express one FILE statement.

Example 1:

```
// FILE NAME-TRANS,
//      UNIT-F1,
//      LABEL-TRANS1,
//      RECORDS-225,
//      RETAIN-T
```

Example 2:

```
// FILE NAME-TRANS,UNIT-F1,LABEL-TRANS1,
// RECORDS-225,RETAIN-T
```

Comments

Comments can contain any character but should not contain a question mark (?). The question mark has a special meaning in procedures and certain control statements. Any combination of valid characters can be included in the following places:

- Following the * on the OCL comment statement.

```
*THIS IS AN EXAMPLE OF A COMMENT STATEMENT
```

  In the example above the comment is THIS IS AN EXAMPLE OF A COMMENT STATEMENT.

- After the last parameter in a statement or in an OCL record that is continued (*continuation* is described in the preceding paragraph). Leave one or more blanks between the last parameter and your comment.

```
// LOAD $COPY LOAD THE DISK COPY UTILITY
```

  In this example the comment is after the last parameter. The comment is LOAD THE DISK COPY UTILITY.

  In the following example, the comments are in an OCL record that is continued:

```
// FILE NAME-TRANS,UNIT-F1,LABEL-TRANS1,      COMMENT A
// RECORDS-225,RETAIN-T                       COMMENT B
```

- After the identifier on statements without parameters. Leave one or more blanks between the identifier and your comments.

```
// RUN RUN THE DISK COPY UTILITY
```

  The comment here is RUN THE DISK COPY UTILITY.

- After an identifier where parameters are optional, such as on a command statement (see index entry: *command statement*), leave a blank after the identifier, code a comma, leave a blank after the comma, and enter the comment.

```
// INCLUDE PROC , MAIN PROCEDURE
```

  The comment here is MAIN PROCEDURE.

The following two tables are intended for quick referencing. The tables are: table of OCL statements (Figure 2) and table of parameters (Figure 3).

The table of OCL statements (Figure 2) gives the identifier, function, placement, and restrictions for each OCL statement.

The table of parameters (Figure 3) describes the contents (identifier and related parameters) of the OCL statements.

When using Figure 3, remember that words written in lowercase letters, such as filename or value, require information you must supply, depending on the functions you want the statement to perform. Refer to Figure 3 to determine which parameters are valid. Keyword parameters that are capitalized must be coded along with the appropriate keyword value.

If you are not familiar with an entry, or you do not know when to use or omit it, refer to the proper statement in the next section, *OCL Statement Descriptions.*

Statement	Function	Placement in Job Stream	Restrictions on Use
// COMPILE	Tells the system the source program to be compiled	Must follow LOAD statement and precede the RUN statement	
// DATE	Supplies the system with a date, which is given to disk files being created and printed on printed output	Must follow LOAD statement and precede RUN statement *except* for performing an IPL, when it must precede the first LOAD statement	Only one DATE statement is allowed between a LOAD and a RUN statement
// FILE	Supplies file information to the system	Must follow LOAD statement and precede the RUN statement	
// FORMS	Instructs the system to change the number of lines printed per page	Can be placed anywhere among the OCL statements	
// IMAGE	Tells the system to replace the print belt image area with characters keyed in or read from a member in the source library	Can be placed anywhere among the OCL statements	Mandatory if the print belt was changed
// INCLUDE	Identifies the procedure member to be merged into job stream	Can be placed anywhere among the OCL statements	Can include sixteen levels of nested procedures
// LOAD	Identifies the program to be run	Must precede the RUN statement	Required in the job stream for the program to be run. Only one LOAD per RUN
// LOG	Instructs system to start or stop printing OCL statements and messages on the printer, and whether to skip to line 1 of the next page at end of job	Can be placed anywhere among the OCL statements	
// MEMBER	Identifies the message load member from which messages come	Can be placed anywhere among the OCL statements	

Figure 2 (Part 1 of 2). Table of OCL Statements

Statement	Function	Placement in Job Stream	Restrictions on Use
// PAUSE	Tells the system to stop so that the operator can perform a function. Operator must indicate when program is to continue.	Can be placed anywhere among the OCL statements	
// RUN	Indicates the end of the OCL statements for a program and tells system to run the program	Must be the last OCL statement within the set of OCL statements for each job	Required in the job stream for the program to be run
// SWITCH	Sets one or more external indicators on or off or to leave the indicator as it is	Can be placed anywhere among the OCL statements	Only one SWITCH statement is allowed between a LOAD and a RUN statement
// SYSLIST	Changes the output medium (printed copy or display on the display screen) or specifies that output be neither printed nor displayed	Can be placed anywhere among the OCL statements	
* Comment	Explains the job; does not affect the program in operation	Can be placed anywhere among the OCL statements	The * must be in position 1
/*	Indicates the end of a data file read from the keyboard	Last record of an input data file	Not recognized in a procedure
// * Message id or 'message'	Indicates a message to be displayed to the operator	Can be placed anywhere among the OCL statements	

Figure 2 (Part 2 of 2). Table of OCL Statements

Statement	Parameter	Meaning of Parameter
// COMPILE	SOURCE-name	Name of source program
// DATE	mmddyy or yymmdd or ddmmyy	System date or date for a particular job within a set of statements (job date) mm = month dd = day yy = year *Note:* Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.
// FILE (Disk)	NAME-filename or	Name the program uses to refer to the file
	NAME-COPYIN or	For certain utility program, names the input file when used with the LABEL parameter
	NAME-COPYO	For certain utility programs, names the output file when used with the LABEL parameter
	UNIT-F1	Location of the file is, or will be, the disk. If the parameter is not specified, default is F1
	LABEL-filename	Name you specify to identify the file on the disk
	RECORDS-number or BLOCKS-number	Amount of space needed on the disk for a file
	LOCATION-blocknumber	Number of the block where the file begins or will begin
	RETAIN-S or RETAIN-T or RETAIN-P	Scratch file Temporary file Permanent file
	DATE-mmddyy or DATE-ddmmyy or DATE-yymmdd	Date the file was created
// FILE (Diskette)	NAME-filename	Name the program uses to refer to the file
	UNIT-I1	Location of the file is, or will be, a diskette
	LABEL-filename	Name you specify to identify the file on the diskette

Figure 3 (Part 1 of 3). Table of Parameters

Statement	Parameter	Meaning of Parameter
// FILE (Diskette) (continued)	RETAIN-retention-days	The number of days a file is retained before it expires. Maximum is 998. If 999 is specified the expiration date is set to a value that cannot be met and the file is considered permanent
	DATE-mmddyy or DATE-ddmmyy or DATE-yymmdd	Date the file was created
	PACK-vol-id	Volume identification of the diskette
// FORMS	LINES-value	Number of lines to be printed per page
// IMAGE	HEX	Characters that follow are in hexadecimal form
	or	
	CHAR	Characters that follow are in EBCDIC form
	or	
	MEM or MEMBER	Characters that are identified as HEX or CHAR and located in a source member in the library
	number	Number of characters
	name	Name of the library source member that contains the print belt image characters
// INCLUDE	procedure-name	Name that identifies the procedure member in the library
	procedure parameters	Parameters (as many as 10) to be used by the procedure
// LOAD	program-name	Name of program to be loaded from the library
// LOG	CRT or PRINTER	Use only the display screen for logging. Use the printer and the display screen for logging
	EJECT or NOEJECT	Skip to line 1 of the next page at end of job. Do not skip to line 1 of the next page at end of job
// MEMBER	PROGRAM1-name	Name of load member used for program product level 1 messages. If 0 is specified, the member name is cleared
	PROGRAM2-name	Name of load member used for program product level 2 messages. If 0 is specified, the member name is cleared

Figure 3 (Part 2 of 3). Table of Parameters

Statement	Parameter	Meaning of Parameter
// MEMBER (continued)	USER1-name	Name of load member used for user program's level 1 and OCL message statements. If 0 (zero) is specified, the member name is cleared
	USER2-name	Name of load member used for user program's level 2 messages. If 0 (zero) is specified, the member name is cleared
// PAUSE	none	
// RUN	none	
// SWITCH	nnnnnnnn where n can be 0, 1, or X	See index entry: *// SWITCH statement*
// SYSLIST	CRT	Use the display screen for SYSLIST output
	PRINTER	Use the printer for SYSLIST output. (The printer is assigned during IPL.)
	OFF	Ignore request for SYSLIST output
* Comment	none	
/*	none	
// * message id or 'message'	msg-id	The identification of a message in the assigned USER1 message member
	'message'	A character string that is the actual message (The character string must be enclosed in single quotes.)

Figure 3 (Part 3 of 3). Table of Parameters

In this section, each OCL statement is described separately. The following information is given for each statement:

- Its function
- Its placement in relation to other statements and the circumstances under which it is needed
- Its format
- Its contents (the parameters that can be used with it)

COMPILE Statement

Function

The COMPILE statement identifies the library member that contains the *source program* to be compiled. A source program is a collection of statements, such as RPG II specifications, that can be translated into an *object program.* An object program is a program that can be loaded into main storage and run. Object programs are stored in the library as load members. Source programs are stored in the library as source members.

Placement

The COMPILE statement must be within the set of OCL statements that apply to the compilation. The COMPILE statement must follow the LOAD statement and precede the RUN statement. If the source program to be compiled follows the RUN statement in the jobstream, the COMPILE statement must not be used.

Format

```
// COMPILE SOURCE-name
```

Contents

SOURCE: This parameter specifies the name of the source member that contains the source program to be compiled.

Example

The following sample COMPILE statement tells the system that the source member with the name PROG3 is the name of the program to be compiled. (LOAD loads the RPG II compiler program and RUN executes the RPG II compiler program.) For additional information about how to compile an RPG II program, see index entry: *OCL (operational control language)* in the *IBM System/3 RPG II Reference Manual*, SC21-7595.

```
// LOAD #RPG
.
.
.
.
// COMPILE SOURCE-PROG3
// RUN
```

DATE Statement

Function

A DATE statement establishes the system date if it is given after IPL and before the first LOAD statement. If a DATE statement is not given during IPL, the system date remains unchanged from what it was set to by a previous DATE statement, DATE procedure, or SET procedure (see index entries: *DATE procedure* and *SET procedure*).

A DATE statement between the LOAD and RUN statements (see index entries: *// LOAD statement* and *// RUN statement*) changes the job (program) date, but only for the program being run. When the program ends, the program date is reset to the system date. If a DATE statement is not given between LOAD and RUN, the system date is used as the program date.

The date established for the program is used to determine file retention periods for diskette files (see the RETAIN parameter for diskette files under index entry: *// FILE statement)* and is printed on printed output. The data is also used for the creation date of the disk or diskette files created by the program.

Placement

A DATE statement can be given after IPL and before a LOAD statement. It can also be included anywhere within the OCL statements for a given program, provided it follows the LOAD statement and precedes the RUN statement. Only one DATE statement can be given between a LOAD and a RUN statement.

Format

// DATE mmddyy or yymmdd or ddmmyy

Contents

The system date can be in either of three formats: month-day-year (mmddyy), year-month-day (yymmdd) or day-month-year (ddmmyy). However, you must use the current system date format. The STATUS procedure (see index entry: *STATUS procedure*) can be used to determine the current format.

Note: Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems. The SET procedure can be used to change the system date and the system date format.

Month, day, and year must each be 2-digit numbers, but leading zeros in month and day can be omitted when punctuation is used. The date can be entered with or without punctuation. For example, July 24, 1975 could be specified in any one of the following ways:

7-24-75	mm-dd-yy
75-7-24	yy-mm-dd
24-7-75	dd-mm-yy
072475	mmddyy
750724	yymmdd
240775	ddmmyy

In the punctuated form, any characters except commas, quotes, numbers, and blanks can be used as punctuation.

Example

The DATE statement for the day of July 1, 1975 could be : // DATE 07-01-75 or // DATE 7-1-75.

FILE Statement

Function

The FILE statement supplies the system with information about disk and diskette files. The system uses this information to read records from and write records on the disk and diskettes.

Placement

A FILE statement is used for each new disk or diskette file that a program creates, and for each of the existing disk or diskette files that a program uses. The FILE statement must follow the LOAD statement and precede the RUN statement.

CAUTION
Be careful when using files during inquiry. (See index entry: *inquiry interrupt.*)

Format

// FILE parameters

Contents

The contents section of the FILE statement description is divided into two sections, one section for files on the disk and one section for files on diskettes.

Contents of FILE Statement for Disk Files

All of the parameters are keyword parameters (keywords are in capital letters), as follows:

- NAME-filename (in program)
- UNIT-F1
- LABEL-filename (on the disk)
- RECORDS-number or BLOCKS-number
- LOCATION-block number
- RETAIN-T or RETAIN-S or RETAIN-P
- DATE-mmddyy or DATE-ddmmyy or DATE-yymmdd

The NAME parameter is always required. The others are required only under certain conditions.

NAME: The NAME parameter is always required. It tells the system the name that the program uses to refer to the file. The filename can be any combination of characters (numeric, alphabetic, and special) except commas, single quotes ('), and blanks. The question mark (?), slash (/), and hyphen (-) should not be used in filenames because they have special meanings in procedures (see index entry: *procedure parameters*). The first character of a filename must be alphabetic, #, $, or @. The number of characters in a filename must not exceed eight.

UNIT: The UNIT parameter tells the system whether the file is on the disk or on a diskette. The code for the unit parameter on a FILE statement for the disk is F1. This keyword and value need not be specified for a disk file because F1 is the default value for UNIT parameter.

LABEL: The LABEL parameter tells the system the name by which a file is identified on the disk. If the file is being created, the filename supplied in the LABEL parameter is used to identify the file on the disk. If the LABEL parameter is omitted from a disk FILE statement, the filename from the NAME parameter is used. If a program refers to an existing file by a filename that differs from the filename by which the file is identified on the disk, a LABEL parameter must be supplied. The filename can be any combination of characters (numeric, alphabetic, and special), except commas, single quotes (') and blanks. The question mark (?), slash (/), and hyphen (-) should not be used in filenames because they have special meanings in procedures (see index entry: *procedure parameters*). The first character must be alphabetic, #, $, or @. The number of characters must not exceed eight.

RECORDS or BLOCKS: The RECORDS or BLOCKS parameter tells the system the amount of space needed on the disk for a file being created. (See Appendix A for determining the number of records or blocks.)

When using the BLOCKS keyword, the number of disk *blocks* needed for the file is specified. There are 2560 bytes in 1 disk block—1 block = ten 256-byte sectors. A sector is the smallest quantity of information that can be read from or written to the disk in one read/write operation. Disk blocks available to the user vary with disk size (one *megabyte* = one million bytes):

3.2 Megabyte Disk	5.0 Megabyte Disk	9.1 Megabyte Disk	13.7 Megabyte Disk
1248 blocks	1968 blocks	3576 blocks	5376 blocks

#LIBRARY, the name of the file containing the system library, must be included in the user blocks available. You can use the CATALOG command statement (see index entry: *CATALOG command statement*) to determine the number of disk blocks actually available for other files.

When using the RECORDS keyword, the approximate number of records for the file must be specified. The total space allocated is rounded up to the next block, allowing space to accommodate at least the number of records indicated. The smallest allocatable unit is one block. For example, if you specify ten 50-byte records, 2560 bytes (one block) are allocated.

If the RECORDS parameter is used, the number can be up to six digits long. If the BLOCKS parameter is used,the number can be up to four digits long.

Either of these two keywords, RECORDS or BLOCKS, can appear in the FILE statement, but not both. The keyword must be followed by a number indicating the amount of space needed.

LOCATION: This parameter tells the system the number of the block where a file begins. LOCATION can be used for allocating new output files and identifying existing input files. The keyword for the parameter is LOCATION. A valid entry for LOCATION must meet two requirements. It must be:

- Greater than the sum of 17 plus the number of blocks used by #LIBRARY, and
- Less than or equal to the following values as determined by the disk size:

3.2 Megabyte Disk	5.0 Megabyte Disk	9.1 Megabyte Disk	13.7 Megabyte Disk
1265	1985	3593	5393

LOCATION is required in only two cases:

- You are creating another version of an existing file. To create such a file, a file that has the same name and size (RECORDS or BLOCKS) as an existing file, you must specify a location that is different from the location of the existing file(s) of the same name and size. The creation date of the new file must also be different from the creation date of any existing file of the same name and size.

- You are writing over an existing file. To write over, or overlay, an existing file you must specify the name of the existing file, its size (RECORDS or BLOCKS) when it was created, and its location. (The LOCATION number can be up to four digits long.) A different creation date, taken from the current job date, will exist for the new file.

Note: Use LOCATION with caution. Three procedures—the COMPRESS, RESTORE, and APCHANGE—change file locations. Both the COMPRESS procedure and the RESTORE procedure move files from previous locations on the disk to new locations, thereby invalidating LOCATION parameters specified before the COMPRESS or RESTORE procedure was run. These three procedures do not display a message to notify the operator of the new locations. (For more information on the COMPRESS, RESTORE, and APCHANGE procedures, see index entries: *COMPRESS procedure, RESTORE procedure, and APCHANGE procedure.*) To determine the current location of a file, use the CATALOG procedure (see index entry: *CATALOG procedure*).

The APCHANGE procedure contains an option that changes file locations, also invalidating LOCATION parameters.

RETAIN: The RETAIN parameter classifies files as scratch, temporary, or permanent.

The keyword for the parameter is RETAIN. It must be followed by a code that indicates the classification of the file. The codes are:

Code	Meaning
S	Scratch file
T	Temporary file
P	Permanent file

A scratch file can be used only by the program creating it, and does not exist after the program that created it has ended.

A temporary file is usually used more than once. The area containing a temporary file can be given to another file only under one of the following conditions:

- A FILE statement containing the RETAIN-S parameter is supplied for the temporary file to identify the file as a scratch file, which will not exist after the program has ended.

- Another file with the same LABEL name is loaded into the area occupied by the temporary file, changing only the data. The RECORDS or BLOCKS and LOCATION parameters must be provided and must be the same as the original file.

- The DELETE procedure is used to delete the file.

The area containing a permanent file cannot be used for any other file until the DELETE procedure is used to delete the permanent file (see index entry: *DELETE procedure*).

The system supports up to 200 permanent or temporary files at any one time on the disk (199 user files plus the system file #LIBRARY).

A disk file is classified as scratch, temporary, or permanent when it is created. If the RETAIN parameter is omitted from the FILE statement when the file is created, the file is assumed to be a temporary file.

The RETAIN parameter can be omitted when accessing an existing file. If an existing permanent file is referenced by a FILE statement with RETAIN-T, it will remain a permanent file. If an existing temporary file is referenced by a FILE statement with a RETAIN-P, it will remain a temporary file. No message is issued by the system to reflect the above situations. However, a message is issued if an existing permanent file is referenced by a FILE statement with RETAIN-S. If processing is continued, the file remains permanent.

DATE: The DATE parameter identifies the creation date of the file. Though the date is not used when creating a file, it is used to ensure that the proper version of a file is referred to. When a file is created on disk, its LABEL name and creation date are written on the disk as identification. The job (program) date is the date used. More than one file can be given the same name. However, the creation dates of these files must be different. To refer to such a file, you can use its name and date, its name and location on disk, or its name and size if the size is unique. If neither the date nor the location is given, the file having the latest date is the one automatically referred to.

The date can be entered in one of three forms: month-day-year (mmddyy), day-month-year (ddmmyy) or year-month-day (yymmdd). However, the form chosen must conform to that of the current system date format.

Sample FILE Statement for a Disk File

A program is creating a disk file; therefore, it must have a FILE statement. Assume the following facts about the file:

- The name the program uses to refer to the file is TRANS
- The name of the file on the disk is TRANS1
- The file is to be saved for use at the end of the month but it can be deleted at the first of the next month
- The file contains 225 records
- The system is to choose the disk area to contain the file

A FILE statement that could be entered to define the file is:

```
// FILE NAME-TRANS,UNIT-F1,LABEL-TRANS1,
// RETAIN-T,RECORDS-225
```

Contents of FILE Statement for Diskette Files

All of the parameters are keyword parameters, which follow (keywords are in capital letters):

- NAME-filename (in program)
- UNIT-I1
- LABEL-filename (on diskette)
- RETAIN-retention-days
- DATE-mmddyy or DATE-ddmmyy or DATE-yymmdd
- PACK-vol-id

The NAME and UNIT parameters are always required. The others are required only under certain conditions.

NAME: The NAME parameter is always needed. It tells the system the name that the program uses to refer to the file.

The keyword for the parameter is NAME. It must be followed by the filename used by the program. The filename can be any combination of characters (numeric, alphabetic, and special) except commas, single quotes ('), and blanks. The question mark (?), slash (/), and hypen (-) should not be used in filenames because they have special meanings in procedures (see index entry: *procedure parameters*). The first character of a filename must be alphabetic, #, $, or @. The number of characters in a filename must not exceed eight.

UNIT: The UNIT parameter tells the system whether the file is on the disk or on a diskette.

The code for the UNIT parameter on a FILE statement for the diskette is I1. This keyword and code must be specified on a diskette FILE statement. If omitted, the UNIT parameter will default to disk.

LABEL: The keyword for the parameter is LABEL. It must be followed by the name of the file on the diskette. The filename can be any combination of characters (numeric, alphabetic, and special) except commas, single quotes ('), and blanks. The comma, single quote ('), question mark (?), slash (/), and hyphen (-) should not be used in filenames because they have special meanings in procedures (see index entry: *procedure parameters*). The first character must be alphabetic, #, $, or @. The number of characters must not exceed eight.

The LABEL parameter tells the system the name by which the file is identified on a diskette. If the file is being created, the name supplied in the LABEL parameter is used to identify the file on a diskette. If the LABEL parameter is omitted from a diskette FILE statement, the name from the NAME parameter is used. If the file is an existing file, a LABEL parameter is required when the name the program uses to refer to the file differs from the name with which the file is identified on a diskette.

If multiple files are to be created on a single diskette, each file LABEL must be unique. Duplicate file labels on the same diskette are not permitted.

RETAIN: The RETAIN parameter specifies duration—the number of days a file is to be retained. It is used to compute an expiration date. Whenever RETAIN is given for a file, the system determines the expiration date of the file by adding to the job (program) date the number of days specified by the RETAIN parameter. The RETAIN parameter can be from 0 to 999. When creating a new file, if RETAIN is omitted, 1 is assumed. If up to 998 is specified, the file is retained for this number of days, if 999 is specified, the file is considered permanent but can be deleted by the DELETE procedure (see index entry: *DELETE procedure*).

When creating a diskette file, the system writes the expiration date of the file in the same format as that of the current system date. If an existing nonpermanent diskette file is referenced by a FILE statement with a RETAIN parameter, the expiration date of the file is changed to the date determined by the RETAIN parameter. The new expiration date is written in the format of the current system date regardless of the format of the file creation date. .

If an existing permanent diskette file is referenced by a FILE statement with a nonpermanent RETAIN parameter, an error message is issued. If you decide to continue processing after the message is displayed, the file remains as a permanent file.

Whenever the system is creating a file on a diskette or adding to an existing file on a diskette, all files on the diskette whose expiration dates were met and all files with blank (hex 40s) expiration dates are deleted automatically. When the expiration dates are checked for having been met, each expiration date is checked for the international format (yymmdd). If an expiration date is not in the international format, it is assumed to be in the same format of the system date. If the expiration date for a diskette file is not in the international format and is not in the format of the system date, the expiration date may be misinterpreted by the system and the file might be deleted before the expiration date is actually met.

When a new file is created on a diskette, the new file starts at the first available sector beyond the last unexpired existing file.

DATE: The DATE parameter is the creation date of an existing file. It is used to ensure that the proper version of a file is referred to. The format specified must be the same as the format of the creation date of the diskette file referred to.

Note: When a file is created on diskette, its label, filename, expiration date, and creation date (job date) are written on the diskette as identification. The job (program) date is the date described under *DATE statement* (see index entry: *// DATE statement*). This date can be in one of three formats: month-day-year (mmddyy), day-month-year (ddmmyy), or year-month-day (yymmdd). However, the creation date of each file on a diskette must be in the same format as every other creation date on the diskette, or the file might be deleted before the intended expiration date.

PACK: A PACK parameter is required when creating a file or adding to a file on a diskette. The PACK parameter provides the system the volume identification (vol-id) of the diskette associated with this FILE statement. The vol-id is put on the diskette (pack) by the INIT procedure (see index entry: *INIT procedure).* PACK must be followed by the vol-id of the diskette associated with this file. The vol-id can be any combination of six or less alphameric characters.

The PACK parameter vol-id will be compared with the vol-id of the inserted diskette. If they are unequal, a message is displayed to the operator who then has the option to continue processing (ignore vol-id), to insert the correct diskette, or to cancel the job.

If the PACK parameter is not supplied on the diskette FILE statement for an output file or when adding to a file, an error message is displayed to the operator with a cancel option only.

The PACK parameter is not required for a diskette input file; however, it is recommended that you ensure that the proper diskette is inserted.

Sample FILE Statement for a Diskette File

Assume the following facts about a file to be created on a diskette.

- The program that creates the file refers to the file as TRANS
- The name of the file once it is on a diskette will be TRANS1
- The file is to be saved for use at the end of the month but can be deleted the first of the next month. There are seven days left in the month
- The file contains 225 records
- The file will be on the diskette identified by 666666

The FILE statement for the file could be:

```
// FILE NAME-TRANS,UNIT-I1,LABEL-TRANS1,
// RETAIN-8,PACK-666666
```

FORMS Statement

Function

The FORMS statement changes the number of lines that the printer will print per page. This number of lines is effective until another FORMS statement or LINES procedure or SET procedure is used (see index entries: *LINES procedure* and *SET procedure)* or an RPG II program specifies some other number. During IPL the number of lines per page is set to the value existing in the system configuration record.

Placement

The FORMS statement can be placed anywhere among the OCL statements.

Format

```
// FORMS LINES-value
```

Contents

LINES: Value is used to indicate the number of lines per page. The maximum number of lines that can be specified per page is 84. The value specified must not exceed two digits. The LINES parameter remains in effect until a SET procedure (see index entry: *SET procedure*) is used to change the variable, another FORMS statement is received, an IPL is performed, or a 3 option (immediate cancel) is taken in response to a message. If a line counter specification is used in an RPG II program, it remains in effect only for the duration of that program.

The printer will skip (overflow) to a new page when six less than the number of lines specified are printed. For example, if LINES-84 is specified, the printer skips to a new page after printing line 78. If LINES-6 is specified, there would be one line printed per page. When five or less lines are specified, there is printing on every line (no overflow).

Note: RPG II programs can specify their own overflow rules to override the values specified in a // FORMS statement.

Example

The following statement tells the system that the forms length is 50 lines per page:

```
// FORMS LINES-50
```

IMAGE Statement

Function

To operate correctly, the printer requires that the characters matching those on the print belt be in a special area of main storage called the print belt image area. When one print belt is replaced for another with different characters, the contents of the print belt image area must also be changed.

The IMAGE statement instructs the system to replace the contents of the print belt image area with the characters indicated by the statement.

The characters can be entered from the keyboard or read from a source member in the library on disk. The effect of the IMAGE statement is temporary and the system print belt image is returned to the print belt image area during IPL. The SET procedure can also change the print belt image (see index entry: *SET procedure*).

Placement

The IMAGE statement can appear anywhere among the OCL statements.

Format

// IMAGE parameters

Contents

The IMAGE statement tells the system either:

- The new print belt characters are to be read from the keyboard, or
- The new print belt characters are to be read from a source member in the library.

Characters from the Keyboard

To indicate that the new print belt characters are to be entered from the keyboard, use the following parameters:

CHAR or HEX: Use the word CHAR to indicate that the characters are in alphameric form. Use the word HEX to indicate that the characters are in hexadecimal form. (See Appendix F for the hexadecimal form of standard characters.)

Note: Two characters, the reverse slash (\) and the grave accent (`) are not on the keyboard but are on the print belt. If these characters are to be entered from the keyboard, all characters must be entered in hexadecimal form.

Number: The number parameter must be used with HEX and CHAR. This parameter is the number of characters following the IMAGE statement (after the IMAGE statement is entered, the entry of the characters is prompted for). This number must not exceed 384 when the characters are hexadecimal, 192 and characters are alphameric.

Following are the rules for entering the new characters from the keyboard:

- The characters must begin in position 1
- Consecutive positions must be used and characters must be entered in the sequence in which they appear on the new print belt

  The sequence for entering characters on the 48-character print belt is:

  1234567890#@/STUVWXYZ&,%JKLMNOPQR-$*ABCDEFGHI+.'

  The sequence for entering characters on the 48-character FORTRAN (48HN) print belt is:

  1234567890=)/STUVWXYZ&,(JKLMNOPQR-$*ABCDEFGHI+.'

  The sequence for entering characters on the 64-character print belt is:

  1234567890#@/STUVWXYZ&,%JKLMNOPQR-$*ABCDEFGHI+.'¢<(|!);¬|_>?:=''

  The sequence for entering characters on the 96-character print belt is:

  1234567890#@/STUVWXYZ&,%JKLMNOPQR—$*ABCDEFGHI+.'
  ¢[(] !)½¼;_°?:=''±§abcdefghi^{2} ^{3}¶jklmnopqr®£stuvwxyz

  Note: The question mark (?) has a special meaning in procedures, therefore if you use a procedure to enter the // IMAGE statement you must use the HEX form for all characters.

- A line must be filled before characters can be continued on the succeeding line (beginning in position 1 of the new line)

Characters from Source Member in the Library

MEM,name or MEMBER,name may be entered to indicate that new print belt characters are to be read from a source library member. Name identifies the source member containing the characters.

In the following example, the name parameter indicates that the characters are to be found in a source member named BELT48:

```
// IMAGE MEM,BELT48
```

A // IMAGE statement specifying the format as either hexadecimal (HEX) or alphameric (CHAR) and specifying the number of characters in the source member is required as the first record within the source member. Because BELT48 is the source member that contains the image of the standard 48-character print belt, BELT48 contains the following:

```
// IMAGE CHAR,48

1234567890#@/STUVWXYZ&,%JKLMNOPQR-$*ABCDEFGHI+.'
```

Examples

The IMAGE statements in Examples A and B tells the system that the new characters are to be entered from the keyboard. The HEX parameter in example A indicates that the new characters are in hexadecimal form; the number parameter indicates that there are 128 positions containing the new characters.

Example A

```
// IMAGE HEX,128
```

In example B, the new characters entered from the keyboard are alphameric. The number parameter indicates that there are 48 positions containing the new characters.

Example B

```
// IMAGE CHAR,48
```

INCLUDE Statement

Function

The INCLUDE statement identifies the procedure member containing the OCL to be merged into the job stream, and any utility control statements (see index entry: *writing utility control statements*) to be merged into the job stream. The INCLUDE statement also enables you to pass parameters to the identified procedure member. In effect, the INCLUDE statement causes system input to come from a procedure. See index entry: *procedures* for more information on procedures.

Placement

The INCLUDE statement can be placed anywhere within a set of OCL statements.

Format

// INCLUDE procedure-name parameters

Contents

The // and the INCLUDE can be omitted. Procedures are usually evoked by command statements. Command statements consist only of the procedure name followed by the parameter values to be passed to the procedure.

The // with only the procedure name (no INCLUDE statement identifier) is also allowed. However, if the procedure name is the same as an OCL statement identifier or is IF or ELSE, then // INCLUDE must be present. For example, if the procedure name is LOAD, then the following format is correct:

```
// INCLUDE LOAD parameter(s)
```

Procedure-name: The procedure name is the name of the procedure member to be merged into the job stream.

Parameters: Parameters may or may not be required, depending on the particular included procedure they are passed to. Parameters are separated by commas. A parameter can be omitted. See the example that follows. The parameters required for IBM-supplied procedures are found in Part 2 of this manual.

The parameters can be any combination of characters except question marks, commas, quotation marks (single and double), slash (/), hyphen (-), or blanks. The number of characters per parameter must not exceed eight. The number of parameters must not exceed ten per INCLUDE statement.

Parameters passed in an INCLUDE statement must be interpreted by the procedure (see index entry: *modifying a procedure job stream*).

Example

In the following example, parameter number 2 was omitted. JOE and SAM are two parameters that will be interpreted by the PAYROLL procedure:

```
// INCLUDE PAYROLL JOE,,SAM
```

In the following example, procedure FILE1 is included between the LOAD and RUN statements and the name of the file (WEEKLY) is being passed to the procedure. Procedure FILE1 contains only the FILE statements necessary to execute the program PAYROLL.

An INCLUDE statement

```
// LOAD PAYROLL
FILE1 WEEKLY
// RUN
```

Assuming that PAYROLL requires only two FILE statements and the procedure FILE1 contains these two FILE statements, the effect of the preceding three OCL statements would be the following sequence of OCL statements entered into the system:

Merged into the job stream in place of the INCLUDE statement

```
// LOAD PAYROLL
// FILE LABEL-WEEKLY,...
// FILE...
// RUN
```

LOAD Statement

Function

The LOAD statement identifies the program to be executed.

Placement

The LOAD statement must be the first statement in a set of statements for a program.

Format

```
// LOAD program-name
```

Contents

Program-name: The program-name parameter is the name of the program to be loaded.

Example

In the following sample LOAD statement, $COPY is the symbolic parameter that identifies the Disk Copy/Display Utility Program:

```
// LOAD $COPY
```

LOG Statement

Function

The LOG statement tells the system where to display messages and OCL statements and whether to skip to line 1 of the next page at end of job.

Note: The LOG statement can be used to tell the system to display OCL statements and messages on the printer as well as on the display screen. IPL assigns only the display screen for displaying messages and OCL statements.

Placement

The LOG statement can be used anywhere within the set of OCL statements for a program.

Format

```
// LOG CRT       ,EJECT
       or         or
       PRINTER  ,NOEJECT
```

Contents

Parameter	Meaning
CRT	Use only the display screen.
PRINTER	Use the printer and the display screen.
EJECT	Skip to line 1 of the next page at end of job. EJECT is assumed if neither EJECT nor NOEJECT is specified.
NOEJECT	Do not skip to line 1 of the next page at end of job.

Example

The following example specifies that messages and OCL statements are to be displayed on both the display screen and the printer, and that EJECT is assumed:

```
// LOG PRINTER
```

MEMBER Statement

Function

The MEMBER statement allows the user to identify the message load member from which messages are to come.

There are four types of message load members: PROGRAM1, PROGRAM2, USER1, and USER2.

PROGRAM is used by IBM program products to assign names to associated message load members.

USER means that the messages are for user-generated programs and OCL statements.

Level 1 messages are 40 characters in length and do not give the detail found in level 2 messages which can be 200 characters in length. A level 2 message can be displayed only after the level 1 message of the same MIC (message identification code) is issued. (See index entry: *$MGBLD utility program* for a description of creating a message load member.)

Placement

The MEMBER statement can be placed anywhere among OCL statements.

Format

// MEMBER parameters

Contents

All the parameters are keyword parameters (keywords are in capital letters) as follows:

PROGRAM1-name — The name of the load member used for IBM program product level 1 messages. Each IBM program product has its own set of names for related message load members.

If 0 (zero) is specified for name, the system will not look for requested PROGRAM1 messages but will display a message indicating that the requested message was not found.

PROGRAM2-name — The name of the load member used for IBM program product level 2 messages. Each IBM program product has its own set of names for related message load members.

If 0 (zero) is specified for name, the system will not look for requested PROGRAM2 messages but will display a message indicating that the requested message was not found.

USER1-name — The name of the load member used for level 1 and OCL statement messages for a program supplied by the user.

If 0 (zero) is specified for name, the system will not look for requested USER1 messages but will display a message indicating that the requested message was not found.

USER2-name — The name of the load member used for level 2 messages for a program supplied by the user.

If 0 (zero) is specified for name, the system will not look for requested USER2 messages but will display a message indicating that the requested message was not found.

The MEMBER statement is in effect until the user enters another MEMBER statement or an IPL is performed. At IPL, the member names are cleared.

After an included procedure is executed, the load member names are reset to the names used when the INCLUDE statement was read. The following is an example of a MEMBER statement used with an included procedure.

Examples

Procedure A

```
// MEMBER USER1-JOE
// INCLUDE B
// * 6666
```

Procedure B

```
// MEMBER USER1-SAM
// * 7777
// LOAD PAYROLL
// RUN
```

When the MEMBER statement is executed in procedure A, the message associated with MIC 6666 comes from the message load member named JOE. The message associated with MIC 7777 in procedure B comes from the message load member named SAM.

PAUSE Statement

Function

The PAUSE statement causes the SCP to suspend processing. It usually is used to give the operator time to insert a diskette. A message telling the operator which diskette to insert usually precedes a PAUSE statement.

When ready, the operator can restart the SCP by taking the 0 (zero) option to continue. The SCP then continues reading the OCL statements that follow the PAUSE statement.

Placement

The PAUSE statement can be placed anywhere among the OCL statements.

Format

```
// PAUSE
```

Contents

None

RUN Statement

Function

The RUN statement indicates the end of the OCL statements for a program. After the system reads the RUN statement, it executes the program named in the LOAD statement.

Placement

A RUN statement is needed for each of the programs the system will run. In the job stream, it must be the last statement within the set of OCL statements for each job.

Format

```
// RUN
```

Contents

None

SWITCH Statement

Function

The SWITCH statement sets one or more external indicators on or off. If a switch statement is used to set an indicator on, the indicator remains on until:

- Another SWITCH statement sets it off,
- A system IPL is performed (turns all indicators off), or
- A user program sets the indicator off.

Note: If an IBM SCP procedure sets a switch, at the end of the procedure the switch is restored to its original setting.

Placement

The SWITCH statement can be placed anywhere among the OCL statements for a job. However, only one SWITCH statement is allowed between a LOAD and a RUN statement.

Format

```
// SWITCH indicator settings
```

Contents

Indicator settings: The indicator settings parameter consists of eight characters, one for each of the eight external indicators (U1-U8). The first, or leftmost, character gives the setting of indicator U1; the second character gives the setting of U2; and so on.

The parameter must always contain eight characters. For each indicator, one of the following characters must be used:

Character	Meaning
0	Set the indicator off
1	Set the indicator on
X	Leave the indicator as it is

Example

```
// SWITCH 1X0110XX
```

The example shown causes the following results:

Indicator	Result
U1	Set on
U2	Unaffected
U3	Set off
U4	Set on
U5	Set on
U6	Set off
U7	Unaffected
U8	Unaffected

SYSLIST Statement

Function

The SYSLIST (system list) statement changes the method for listing output. Output can be listed on the printer or on the display screen, or specified not to be listed at all.

Placement

The SYSLIST statement can be placed anywhere among OCL statements.

Format

```
// SYSLIST parameter
```

Contents

The parameter can be:

Parameter	Meaning
CRT	Display output on the display screen.
	Note: If CRT is specified on a SYSLIST statement, the ROLL↑ without the SHIFT key (roll up) must be pressed after each system list output record is displayed to advance to the next record.
PRINTER	Print output on the printer. (The printer is assigned during IPL.)
OFF	Do not list output.

Example

The following is an example of assigning the printer for listing output:

```
// SYSLIST PRINTER
```

Comment Statement

Function

Comment statements are usually used to explain the purpose of the OCL statements and utility control statements stored in a procedure. (See index entries: *writing utility control statements* and *procedures* for a description of utility control statements and procedures.) Comments in a procedure are displayed when the procedure is displayed. Comments are not displayed when the procedure is being executed.

Placement

Comment statements can be placed anywhere among the OCL statements, except between IF and ELSE expressions. (See index entry: *parameters, statement* for statement parameter rules of the IF and ELSE expressions.)

Format

```
* comment
```

Contents

Comment statements must contain an asterisk (*) in position 1. The text of the comment itself can be any combination of words and characters except the question mark. Because the question mark (?) has a special meaning in procedures (see index entry: *procedure parameters*) and certain control statements, comments should not contain a question mark.

/* End of Data Statement

Function — /* statements indicate the end of data files entered from the keyboard.

Placement — A /* statement must be the last record of an input data file.

Format — /*

Note: An end of data statement is not recognized in a procedure.

// * Message Statement

Function — The message statement provides a means of displaying messages to the operator from a procedure.

Placement — The message statement can be placed anywhere among OCL statements.

Format — // * msg-id or 'message'

Contents — The parameter can be in either of two forms:

msg-id — This is the identification of a message in the USER1 message member specified on the // MEMBER OCL statement (one to four numerics). (See index entry: *// MEMBER statement* for a description of the USER1 message member.)

'message' — A character string enclosed by single quotation marks is the actual message. Any character can be used in the character string except a single quote or a single question mark (?). (See index entry: *substitution in procedures* for the use of the question mark in an OCL message statement.) The maximum number of characters in the character string is 120 characters minus the OCL statement characters (//, *, ', and blanks) not included in the actual message. However, when the message is displayed on the display screen, only the first 40 characters of the character string are displayed. When the message is printed, all characters in the character string are printed.

The message is always displayed to the operator when the statement is processed in the job stream.

Example — In the following example, the message statement would very likely be followed by a PAUSE statement to allow the operator to change the diskettes:

```
// * 'INSERT THE PAYROLL MASTER DISKETTE'
```

Part 2

Procedures

A procedure is a set of related OCL statements and, possibly, utility control statements (see index entry: *writing utility control statements* for a description of utility control statements). A procedure is stored in the system library as a procedure member. Each procedure, and thereby each procedure member, must have a unique name. This name is the name by which a procedure is evoked.

One procedure can cause more than one job to be run. That is, a single procedure may contain more than one LOAD statement and RUN statement (see index entries: *// LOAD statement* and *// RUN statement*).

The ability to store sets of frequently used OCL statements and utility control statements makes it possible to avoid recoding and rekeying the statements each time they are required.

IBM SCP PROCEDURES

The following list of names identifies the procedures supplied with IBM System/32 system control programming to provide you with an easy method of using system functions.

ALTERBSC	CREATE	LINES	RESTORE
ALTERSDL	DATE	LISTLIBR	SAVE
APCHANGE	DELETE	LOG	SET
BACKUP	DISPLAY	ORGANIZE	SETMICR
CATALOG	FROMLIBR	OVERRIDE	SPECIFY
COMPRESS	HISTORY	REBUILD	STATUS
CONDENSE	INIT	RELOAD	SYSLIST
CONVERT	JOBSTR	REMOVE	TOLIBR
COPYI1		RENAME	TRANSFER

Notes:

1. The ALTERSDL and SPECIFY procedures are intended for data communication programming that uses SDLC (synchronous data link control). The ALTERBSC and OVERRIDE procedures are intended for data communication programming that uses BSC (binary synchronous communications). Data communication programming using SDLC and BSC is described in the *IBM System/32 Data Communications Reference Manual,* GC21-7691.
2. The SETMICR procedure is used with the 1255 Magnetic Character Reader attachment and is described in *IBM System/32 1255 Magnetic Character Reader Reference and Logic Manual,* GC21-7692.

IBM also provides SCP service procedures to help you and IBM service personnel solve system problems that may arise. The service procedures provided are:

APAR	DUMP	TRACE
BUILD	PATCH	

The service procedures are described in Appendix D. Three other procedures, APPLYPTF, CNFIGSCP, and INSTALL, are part of the installation steps described in Part 5.

Some of the IBM SCP procedures call and use other IBM SCP procedures that you cannot evoke directly. Though you cannot evoke these procedures directly, their names may appear on listings you request.

You can create your own procedures to use in addition to those provided by IBM. The information contained in this part of the manual, Part 2, will help you create and evoke your own unique procedures as well as use those provided by IBM.

CREATING A PROCEDURE

A procedure can be created and stored in the library by keying statements from the keyboard and using the $MAINT utility program (see index entry: *$MAINT utility program*) or another program such as the Source Entry Utility (described in *IBM System/32 Utilities Program Product Reference Manual—Source Entry Utility,* SC21-7605). An existing set of OCL statements and utility control statements can be read from a diskette to the disk by using one of the procedures or utilities described in this manual.

EVOKING A PROCEDURE

Procedures can be evoked in three ways:

- By keying an INCLUDE OCL statement (command statement)
- By using a command key
- By calling a procedure from another procedure

Keyboard Entry of the INCLUDE Statement

A procedure usually is called by a simplified form of the INCLUDE OCL statement known as a *command statement.* Command statements are formed by deleting the // and INCLUDE from the format of INCLUDE statements. That is, the general format of a command statement is:

```
Procedure name Parameter-1,Parameter-2,...Parameter-n
```

A command statement can begin in any position—a command statement does not have to begin in position 1.

For example, keying

```
PAYROLL
```

and pressing the ENTER key is sufficient to call a procedure named PAYROLL, provided no parameters need to be passed to the procedure.

For a description of the other two formats permitted for an INCLUDE OCL statement, see index entry: *// INCLUDE statement.*

Note: The // and INCLUDE cannot be omitted from the INCLUDE statement if you want to evoke a procedure whose name is IF, IFT, IFF, ELSE, RETURN, or CANCEL, or if the procedure name is the same as an OCL statement identifier.

Using a Command Key

The command key is another way of evoking a procedure. By pressing the CMD key in response to READY and then an upper or lowercase assigned command key (the command keys are the 12 keys in the top row of the typewriter keyboard). You can request one procedure for each uppercase key and one procedure for each lowercase key. Therefore, you can request 24 procedures using command keys, (see *Assigning Command Keys* following). The procedure can then be evoked by pressing the ENTER key.

Note: Command keys can be used to evoke OCL statements in the same way they can be used to evoke procedures.

When you request a procedure by pressing a command key, the procedure name (and any parameters previously specified for that procedure) is displayed on the display screen. For example, if you are requesting a previously created PAYROLL procedure and PAYROLL has no parameters specified for it, the procedure name appears on the display screen as:

```
PAYROLL
```

The display screen cursor would be positioned at the second position after PAYROLL. If no parameters are required, pressing the ENTER key evokes the PAYROLL procedure. If parameters are to be entered, you must key them before pressing ENTER.

Assigning Command Keys

If you wish to request a procedure by pressing a command key rather than keying in the command statement each time for commonly used procedures, you must create a message source member, a level 2 message load member named ##MSG3; use the CREATE procedure or the $MGBLD utility program to put the load member into the library; and then perform an initial program load (IPL).

The message source member contains a message control statement and a message text statement for the message load member ##MSG3. (See index entry: *message source member* for a description of message control statements and message text statements for assigning a command key to a procedure.)

Command Key Message Identification Codes

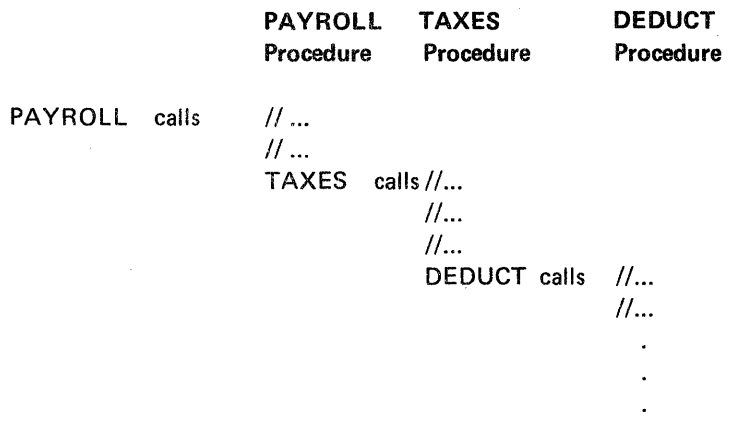
The message load member (##MSG3) must contain one or more of the following twenty-four *message identification codes* (MICs). The MICs are shown with the data characters on the corresponding command keys.

MIC	Command Key (Lowercase)	MIC	Command Key (Uppercase)
0001	1	0013	|
0002	2	0014	@
0003	3	0015	#
0004	4	0016	$
0005	5	0017	%
0006	6	0018	¬
0007	7	0019	&
0008	8	0020	*
0009	9	0021	(
0010	0	0022	)
0011	- (minus)	0023	_ (underscore)
0012	=	0024	+

For example, if the command key message load member contains a MIC of 0010 and the COPYI1 command statement as text, COPYI1 is executed after the CMD key and the 0 data key are pressed.

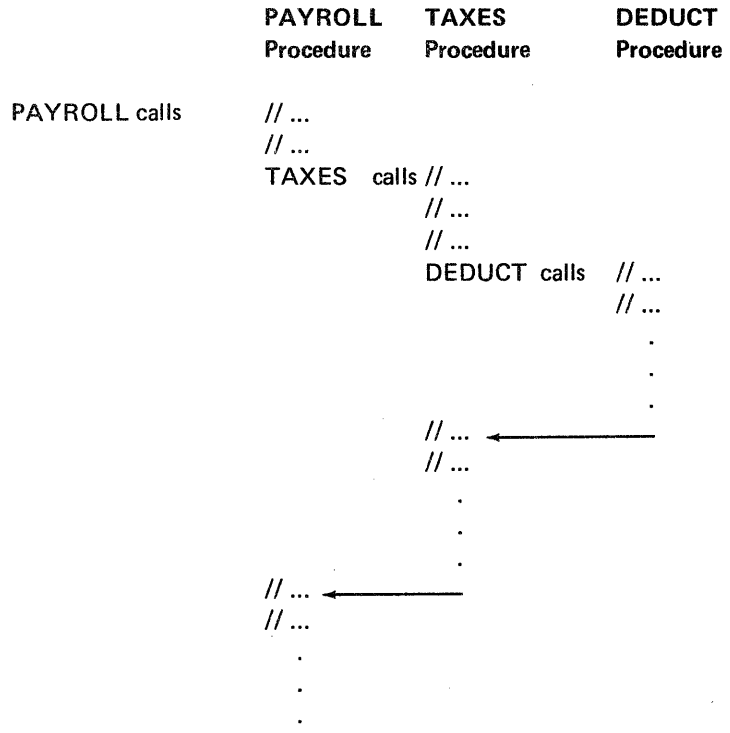
Evoking a Procedure from Another Procedure

A procedure requested by a command statement (INCLUDE OCL statement) or a command key can also request another procedure. For example, suppose a procedure named PAYROLL contains, besides other OCL statements, a TAXES command statement, and the procedure named TAXES contains a DEDUCT command statement. Both the TAXES procedure and the DEDUCT procedure are called and executed when the operator enters the PAYROLL command statement.



A procedure evoked by another procedure is called a *nested procedure.* In the preceding example TAXES and DEDUCT are nested procedures.

A nested procedure normally returns to the procedure that called it. That is:



The preceding example contains three levels of procedures: the first level contains PAYROLL, the second contains TAXES, and the third contains DEDUCT. One level can contain more than one command statement, but a maximum of 16 levels of procedures is allowed in one job stream.

Procedure Execution

When a procedure name is recognized by the SCP the following action occurs:

1. The procedure member corresponding to the procedure name is found in the library.

2. The OCL statements, utility control statements, and/or nested procedures are read, one statement at a time, by the SCP. Parameters are substituted for substitution variables (see index entry: *modifying a procedure job stream)*, IF and ELSE expressions are processed (see index entry: *modifying a procedure job stream*), and the resultant OCL and utility control statements from the original procedure and any nested procedures are executed as a normal job stream.

PROCEDURE PARAMETERS

Some procedures require parameters when the procedures are requested, other procedures do not. A maximum of 10 parameters can be passed when evoking a procedure. Most parameters passed to procedures are *positional parameters*. A positional parameter is a parameter that, whenever it appears in a statement, must appear in the same position in relation to other parameters in the statement. If a valid positional parameter is omitted from a statement requesting a procedure but a following parameter is used, a comma must indicate the position reserved for the omitted parameter.

For example:

```
// INCLUDE PROCEDUR FILEA,,NO
```

FILEA is the first parameter, the second parameter is omitted, and NO is the third parameter. A fourth parameter, XYZ, is omitted, but is not indicated by a comma because it was the last parameter.

Some parameters have *defaults.* A default is a parameter that is substituted for an omitted parameter. You can write defaults in your procedures (see index entry: *modifying a procedure job stream*). Defaults are underlined when shown in command statement formats in this manual.

The comma, single quote ('), question mark (?), slash (/), and hyphen (-) have special meanings in procedures and in OCL and utility control statements (see index entry: *writing utility control statements* for information on utility control statements) and should not be used in parameters for a procedure. The ?, /, and - should not be used in any control statement unless the format of the statement given in this manual indicates that one or more of the symbols is required. (For example, OCL and utility control statements begin with //; a hyphen is required to separate keywords and values in keyword parameters.)

Note: If sequence numbers are used on procedure statements, the sequence numbers are considered comments and the rules for coding comments apply. (See index entry: *comments, rules for using.*)

Modifying a Procedure Job Stream

Within a procedure member, special kinds of expressions can modify the job stream generated when the procedure is invoked. The two types, called substitution expressions and conditional expressions, can be used to substitute values in the generated job stream and to conditionally generate OCL and utility control statements. Following are detailed descriptions of substitution and conditional expressions and examples of how they are used.

Substitution Expressions

Substitution expressions within a procedure allow the programmer to substitute information in the statements generated when the procedure is run. Substituted values can be:

- Passed to the procedure as positional parameters on the command statement that invoked the procedure
- Supplied by the operator in response to prompts issued from within the procedure

Following are descriptions of each of the substitution expression formats that can be used in a procedure member.

Note: In the following forms of substitution, the n can be any number between 01 and 11 (the leading zeros can be dropped). The first 10 parameters on the statement that evokes the procedure are positional. Only 10 parameters can be passed; the eleventh must be supplied by default (for example, ?11,'FILEIN'?) or prompting (for example, ?11R?).

Substitution Format — **Meaning**

?n?

Substitutes the value of the n^{th} positional parameter. If the n^{th} parameter is not defined, no value is substituted. For example, a procedure member contains the following statement:

```
// * '?3? WAS DELETED'
```

If the third parameter is not defined (that is, it was not specified on the command statement and was not assigned a value by a previous statement within this procedure), the following statement is generated:

```
// * ' WAS DELETED'
```

If the value of the third parameter is FILEX, the following statement is generated:

```
// * 'FILEX WAS DELETED'
```

Substitution Format	Meaning

?n'default'?	Substitues the value of the n^{th} positional parameter; or, if the n^{th} parameter is not defined, permanently assigns a default value to the parameter, and then substitues the value. 'default' specifies the permanent default value. If the default value is assigned, subsequent references to the n^{th} parameter within the procedure member use that same default value. For example, a procedure member contains the following statement:
	
	// FILE NAME-?2'FILEIN'?
	
	If the second parameter was not defined, the following statement is generated:
	
	// FILE NAME-FILEIN
	
	Subsequent references to the second parameter use FILEIN. For example, if the next or succeeding statement is:
	
	// FILE NAME-X,LABEL-?2?
	
	the following statement is generated:
	
	// FILE NAME-X,LABEL-FILEIN

?nT'default'?	Substitutes the value of the n^{th} positional parameter; or, if the n^{th} parameter is not defined, temporarily assigns a default value to the parameter, and then substitues that value. This expression format is the same as ?n'default'? except T means temporary and 'default' specifies the temporary default value. A temporary default value is used only for the current substitution expression. For subsequent references to the n^{th} parameter within the procedure, the parameter is undefined. For example, a procedure member contains the following statement:
	
	// FILE NAME-?2T'WEEKLY'?
	
	If the second parameter was not defined, the following statement is generated:
	
	// FILE NAME-WEEKLY

Substitution Format	Meaning

?nR'msg-id'?	Substitutes the value of the n^{th} positional parameter; or, if the n^{th} parameter is not defined, displayes a message from the USER1 message member and waits for the operator to enter from the keyboard the value to be substituted. Subsequent references to the n^{th} parameter within the procedure member use the value entered by the operator. R indicates that an operator's reply is required if the parameter is not defined. Msg-id identifies the MID of the message to be displayed if the n^{th} parameter is not defined. For example, a procedure member contains the following statement:
	
	// FILE NAME-?2R'6666'?
	
	If the second parameter is not defined, the following message (message 6666 from the USER1 message member) is displayed:
	
	ENTER NAME OF THE REQUIRED INPUT FILE
	
	The operator then enters the word PAYROLL from the keyboard, and the following statement is generated:
	
	// FILE NAME-PAYROLL
	
	Subsequent references to the second parameter use PAYROLL.

?nR?	Substitues the value of the n^{th} positional parameter; or, if the n^{th} parameter is not defined, displayes a system message (ENTER MISSING PARAMETER) and waits for the operator to enter the value to be substituted. Subsequent references to the n^{th} parameter within the procedure member use the value entered by the operator. R indicates that an operator's reply is required if the parameter is not defined. For example, a procedure member contains the following statement: PAYROLL WEEKLY,?1R? If the first parameter is not defined, ENTER MISSING PARAMETER is displayed. The operator then enters the word SALARY from the keyboard, and the following statement is generated: PAYROLL WEEKLY,SALARY *Note:* Within the procedure member named PAYROLL, references to the second positional parameter use the word SALARY.

Substitution Format	Meaning

?R?	Displays a system message (ENTER REQUIRED PARAMETER), and waits for the operator to enter the value to be substituted from the keyboard. R indicates that an operator reply is required. For example, a procedure member contains the following statement:
	
	// FILE NAME-?R?
	
	When the statement is being generated, ENTER REQUIRED PARAMETER is displayed. The operator then enters the word INFILE, and the following statement is generated:
	
	// FILE NAME-INFILE

This page intentionally left blank

Substitution Format	Meaning

?R?	Displays a system message (ENTER REQUIRED PARAMETER), and waits for the operator to enter the value to be substituted from the keyboard. R indicates that an operator reply is required. For example, a procedure member contains the following statement:
	
	// FILE NAME-?R?
	
	When the statement is being generated, ENTER REQUIRED PARAMETER is displayed. The operator then enters the word INFILE, and the following statement is generated:
	
	// FILE NAME-INFILE

Conditional Expressions: IF and ELSE

Conditional expressions are used among OCL statements to modify procedures. There are two types of conditional expressions: IF expressions and ELSE expressions.

IF Expression

The IF expression can only be used in a procedure (can be anywhere in the procedure).

The IF expression tests to find out whether a condition is as specified (true or false); if it is as specified, the statement parameter is executed; if not, the SCP goes to the next statement in the procedure.

There are three formats of the IF expression:

```
// IF Condition-parameter  Statement-parameter
// IFT Condition-parameter  Statement-parameter
// IFF Condition-parameter  Statement-parameter
```

IF or IFT means that if the condition is true, the statement is to be executed. IFF means that if the condition is false, the statement is to be executed.

Condition Parameter: There are two types of *condition parameters: existence testing* and *comparison.*

The existence testing parameter is a keyword parameter. The keywords and meanings are:

Keyword	Meaning
BLOCKS-value	Is the available disk file space equal to or greater than the value specified?
DATAI1-'name,date' or DATAI1-name	Is there a file on the diskette with the name and creation date (optional) as specified?
DATAF1-'name,date' or DATAF1-name	Is there a file on the disk with the name and creation date (optional) as specified?
SOURCE-name	Is there a source member of the specified name in the library?
LOAD-name	Is there a load member of the specified name in the library?
PROC-name	Is there a procedure member of the specified name in the library?
SUBR-name	Is there a subroutine member of the specified name in the library?
SWITCH1-0	Is SWITCH1 a 0 (off)?
SWITCH1-1	Is SWITCH1 a 1 (on)?
•	
•	(SWITCH2 through SWITCH8 can also be tested)
•	

The comparison parameter format and meaning is:

Format	Meaning
parameter1/parameter2	Is parameter1 equal to parameter2? (Each parameter has a maximum length of eight characters.)

Statement Parameter: The *statement parameter* of the IF expression can be an OCL statement, (except the comment statement or end-of-data statement) a utility control statement, another IF statement, or a part of an OCL or utility control statement (continuation). Drop the initial // of OCL and utility control statements used as statement parameters.

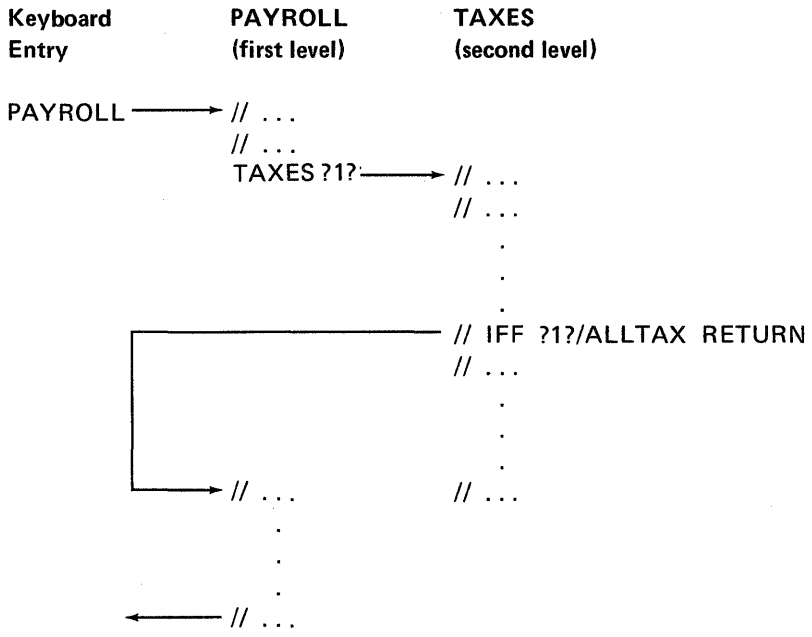
Also allowed in the statement parameter of the IF expression are the keywords CANCEL and RETURN. The meaning of these two keywords are:

CANCEL	Cancel the job and return to the keyboard for the next OCL statement.

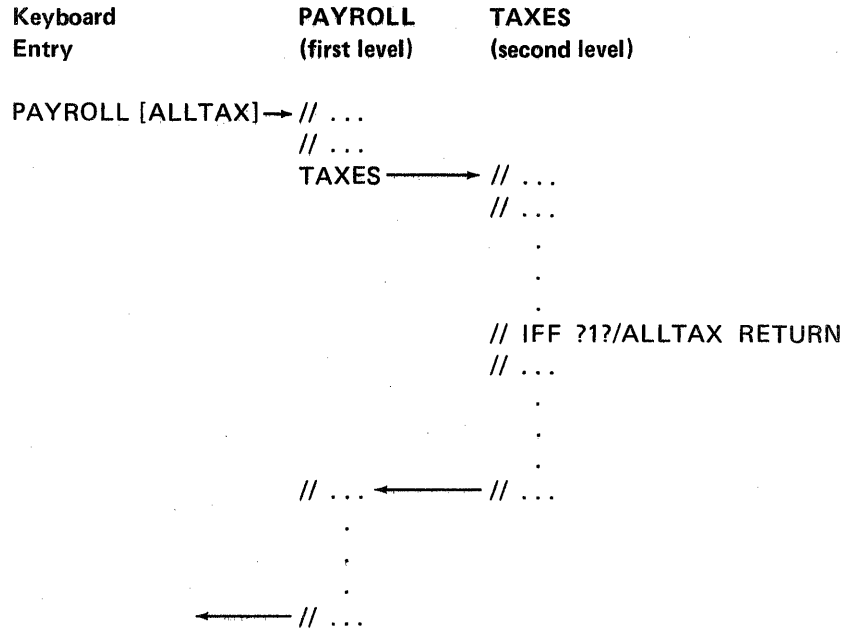
RETURN Encountered in a first level procedure causes an immediate return to the keyboard for the next OCL statement. Encountered in a nested procedure causes an immediate return to the calling procedure for the next OCL statement.

The following example shows two levels of procedures, PAYROLL (first level) and TAXES (second level). The TAXES procedure represents a nested procedure that contains the keyword RETURN in a conditional expression.

In this example, the keyboard entry is PAYROLL. PAYROLL contains an optional parameter ALLTAX which is not specified so that the conditional expression is executed and the keyword RETURN is encountered. RETURN encountered in a nested procedure causes a return to the calling procedure (PAYROLL). The remaining OCL statements in the TAXES procedure are not executed. The example:



If the ALLTAX parameter was specified in the keyboard entry, the RETURN keyword in the conditional expression would not be encountered, all OCL statements in the TAXES procedure would be executed, and then processing returns to the PAYROLL procedure. For example:



Examples of the IF Expression: Following are some examples of the IF expression.

- Existence testing

```
// IFF DATAF1-?1? CREATEF1
```

This expression checks to see if the file label substituted for parameter 1 is on the disk. If it is not, the condition is satisfied and the CREATEF1 procedure is evoked. If the file is on the disk, the condition is not satisfied and the next statement or expression in the procedure is read; CREATEF1 is not executed.

- Comparison

```
// IF ?1?/PAYROLL PAYROLL
```

This expression says that if the first parameter on the statement that evokes the procedure is PAYROLL, then evoke the procedure named PAYROLL; otherwise, go to the next statement or expression in the procedure. An expression equivalent to the preceding comparison example is:

```
// IF PAYROLL/?1? PAYROLL
```

- There can be more than one IF expression on a line. A line is a maximum of 120 characters.

```
// IF PROC-PAYROLL1 IF SWITCH3-1 PAYROLL
```

This expression says that if the procedure member PAYROLL1 is in the procedure library, and if UPSI (user program status indicator) switch 3 is on (1), then the procedure PAYROLL is evoked. If the procedure PAYROLL1 is not in the procedure library, or if the UPSI switch is not on, the expression is ignored and the procedure PAYROLL is not evoked.

ELSE Expression

The ELSE expression requires the IF expression and the following restrictions apply:

- The ELSE expression must immediately follow the IF expression. The ELSE expression is ignored if it does not immediately follow an IF expression.
- Comments cannot be entered between the IF expression and the ELSE expression. The ELSE expression is ignored if comments are entered between the IF expression and the ELSE expression.

For example:

```
// IF ?1?/ RETURN
// ELSE DELETE ?1?
```

The example tests whether the first parameter in the statement that evoked the procedure is a *null entry.* A null entry is an entry that contains no value. In the statement

```
NAME PARM1,,PARM3
```

the second parameter is a null entry.

The IF and ELSE statements in the preceding sample say: if the first parameter in the statement that evoked the procedure is a null entry, RETURN to the previous procedure if this is a nested procedure or to the keyboard if this procedure is not nested; if the first parameter in the statement is not a null entry, evoke the DELETE procedure to delete the file specified by the first parameter.

The ELSE expression can be used with all forms of the IF expression (IF, IFT, and IFF).

There can be only one ELSE expression per line and it must be the first expression in that line. An IF expression can follow an ELSE expression in a conditional statement. For example:

```
// IF ?1?/ PAYROLL
// ELSE DELETE ?1?
// IF ?2?/ RETURN
// ELSE IF ?2?/YEAREND PAYROLL YEAREND
// ELSE PAYROLL WEEKLY
```

In this example, if parameter 1 was passed by the statement that evoked the procedure, the file specified by the parameter would be deleted by the DELETE procedure. If the second parameter is a blank, the PAYROLL procedure is not evoked. If the second parameter is YEAREND, YEAREND is passed to PAYROLL and the PAYROLL procedure is run. If the second parameter is neither a blank nor YEAREND, the parameter WEEKLY is passed to PAYROLL and PAYROLL is run.

EXAMPLE OF PROCEDURE CODING

FILEBKUP Procedure

Note: This is an example of a user coded procedure not provided with the SCP.

The FILEBKUP procedure copies a file from the disk onto a diskette. The following is the command statement format:

FILEBKUP filename-1,vol-id [,filename-2 / ,filename-1]

This procedure demonstrates conditional operator prompting for required parameters, and conditional building of a file statement LABEL parameter.

The prompting text is stored in the procedure rather than in a user library member.

FILEBKUP Parameters

Filename-1 This is a required parameter. It is the name of the disk file to be copied.

Vol-id This is a required parameter. It is the volume identification of the diskette.

Filename-2 This is an optional parameter. If entered, it becomes the name of the file on the diskette. If omitted, the disk and diskette file names are the same.

If either or both of the required parameters (filename-1 and vol-id) were entered with the command, the procedure assumes that filename-2 was intentionally omitted; the procedure defaults to filename-1.

If no parameters were entered with the command, the procedure assumes that the optional filename-2 parameter may be desired; the procedure prompts for filename-2.

The statements for the FILEBKUP procedure example are shown here with each statement numbered for reference to the corresponding explanation:

```
1.  // * 'FILEBKUP IS EXECUTING'
2.  // IF ?1?/ IF ?2?/ IF ?3?/ * 'ENTER YES TO ALTER DISKETTE FILE LABEL'
3.  // IF ?1?/ IF ?2?/ IF ?3?/ * 'PRESS ENTER FOR SAME AS DISKETTE LABEL'
4.  // IF ?1?/ IF ?2?/ IF ?3?/ IF ?11R?/YES * 'ENTER DISKETTE FILE LABEL'
5.  // IF ?11?/YES IF ?3R?/ * 'PARAMETER 3 OMITTED-PROC CANCELED'
6.  // IF ?11?/YES IF ?3?/ CANCEL
7.  // IF ?1?/ * 'ENTER THE DISK FILE LABEL'
8.  // IF ?1R?/ * 'PARAMETER 1 OMITTED-PROC CANCELED'
9.  // IF ?1?/ CANCEL
10. // IF ?2?/ * 'ENTER THE DISKETTE VOLUME ID'
11. // IF ?2R?/ * 'PARAMETER 2 OMITTED-PROC CANCELED'
12. // IF ?2?/ CANCEL
13. // LOAD $COPY
14. // FILE NAME-COPYIN,LABEL-?1?
15. // FILE NAME-COPYO,UNIT-I1,PACK-?2?,RETAIN-999,
16. // IF ?3?/ LABEL-?1?
17. // IFF ?3?/ LABEL-?3?
18. // RUN
19. // COPYFILE OUTPUT-DISK
20. // END
```

The following procedural comment informs the operator of the system activity:

```
1.  // * 'FILEBKUP IS EXECUTING'
```

The statements 2 through 6 prompt the operator for filename-2 if all three parameters were omitted. Statements 2 and 3 display the prompting text. Statement 4 halts the machine after issuing the system message ENTER MISSING PARAMETER. When parameter number 11 is coded as required, the machine stops for key entry.

```
2. // IF ?1?/ IF ?2?/ IF ?3?/ * 'ENTER YES TO ALTER DISKETTE FILE LABEL'
3. // IF ?1?/ IF ?2?/ IF ?3?/ * 'PRESS ENTER FOR SAME AS DISK FILE LABEL'
4. // IF ?1?/ IF ?2?/ IF ?3?/ IF ?11R?/YES * 'ENTER DISKETTE FILE LABEL'
5. // IF ?11?/YES IF ?3R?/ * 'PARAMETER 3 OMITTED-PROC CANCELED'
6. // IF ?11?/YES IF ?3?/ CANCEL
```

Parameter 11 cannot be passed in a command statement: 10 is the maximum number of parameters allowed in a command statement. However, a procedure can supply an eleventh parameter by default, or prompt the operator for the eleventh parameter. The IF expression in statement 4 is equated to YES so that prompting for filename-2 will occur if the operator enters YES. If the operator responds to statement 4 with YES, then statement 5 must have a name entered. If statement 5 receives a null response from the operator, who presses only the ENTER key, the message PARAMETER 3 OMITTED-PROCEDURE CANCELED is displayed and statement 6 cancels the procedure.

The statements, 7 through 12, prompt the operator for filename-1 and vol-id if they were omitted when the command was entered. Statement 7 displays the prompt ENTER THE DISK FILE NAME. Statement 8 or 11 issues the system message ENTER MISSING PARAMETER below the prompt coded in statement 7 or 10, respectively, and then halts the machine for operator key entry. Statement 9 or 12 executes the CANCEL function, similar to statement 6.

```
7.  // IF ?1?/ * 'ENTER THE DISK FILE LABEL'
8.  // IF ?1R?/ * 'PARAMETER 1 OMITTED-PROC CANCELED'
9.  // IF ?1?/ CANCEL
10. // IF ?2?/ * 'ENTER THE DISKETTE VOLUME ID'
11. // IF ?2R?/ * 'PARAMETER 2 OMITTED-PROC CANCELED'
12. // IF ?2?/ CANCEL
```

The next statements, 13 through 20, build the OCL for execution:

```
13. // LOAD $COPY
14. // FILE NAME-COPYIN,LABEL-?1?
```

If the operator omits filename-2, the disk file name is also used for the diskette file name:

```
15. // FILE NAME-COPYO,UNIT-I1,PACK-?2?,RETAIN-999,
16. // IF ?3?/ LABEL-?1?
```

If the operator enters filename-2, it becomes the diskette file name:

```
17. // IFF ?3?/ LABEL-?3?
18. // RUN
19. // COPYFILE OUTPUT-DISK
20. // END
```

The COPYO file statement, statement 15, is incomplete; continuation is shown by the comma following the RETAIN-999 parameter. Depending on the absence or presence of filename-2, either parameter statement 16 or 17 selects the LABEL parameter.

Each IBM SCP procedure can be evoked by a command statement. Figure 4, is a table showing the formats of the command statements that evoke the IBM SCP procedures. (See Appendix D for the format of the command statements that evoke the IBM service procedures.)

Figure 4 is meant for quick reference. It shows the procedure name and parameters (if any) in each command statement. For more information about each of the command statements shown and for a description of the procedures they evoke, see *IBM SCP Procedure Descriptions,* following Figure 4.

Note: The following command statements are intended for data communications programming using either BSC (binary synchronous communications) or SDLC (synchronous data link control):

- ALTERBSC and OVERRIDE using BSC
- ALTERSDL and SPECIFY using SDLC
- MRJE using BSC
- BWSUR and BWSUD using SDLC
- DCPRINT using either BSC or SDLC output

Data communication programming that uses BSC or SDLC is described in *IBM System/32 Data Communications Reference Manual,* GC21-7691.

```
ALTERBSC [BRATE-{F}] [,CLOCK-{Y}] [,DEBUG-{Y}] [,ERC-{number}]
         {H}       {N}          {N}          {7}
         [,SLINE-{Y}] [,TEST-{Y}] [,TONE-{Y}]
                  {N}        {N}         {N}

ALTERSDL [BRATE-{F}] [,CLOCK-{Y}] [,DEBUG-{Y}]
         {H}       {N}         {N}
         [,SLINE-{Y}] [,TEST-{Y}] [,TONE-{Y}]
                  {N}        {N}         {N}

APCHANGE [blocks][,filename [,CLEAR]]

BACKUP vol-id, [retention-days] [,filename    ]
               [1             ] [,#LIBRARY   ]

BWSUD sluname,host

BWSUR sluname
```

Figure 4 (Part 1 of 5). IBM SCP Command Statement Formats

```
CATALOG [ALL     ] [,I1]
        [filename] [,F1]

COMPRESS

CONDENSE

CONVERT

COPYI1 [ALL],,vol-id,[DELETE],[PRESERVE] [,number of copies]
                                         [,1               ]
  or
COPYI1 filename, [mmddyy] ,vol-id,,[PRESERVE] [,number of copies]
                 [ddmmyy]                     [,1               ]
                 [yymmdd]

CREATE sourcename [,REPLACE]

DATE {mmddyy}
     {ddmmyy}
     {yymmdd}

DCPRINT [filename]

DELETE filename, [F1] [SCRATCH] [,mmddyy]
                 [I1] [REMOVE ] [,ddmmyy]
                      [ERASE  ] [,yymmdd]

DISPLAY filename [,mmddyy]
                 [,ddmmyy]
                 [,yymmdd]
  or
DISPLAY filename, [mmddyy] ,RECORD,value-1 [,value-2]
                  [ddmmyy]
                  [yymmdd]

FROMLIBR library-name, [SOURCE ] [,filename-1  ] [,I1] [,ADD      ] [,vol-id]
                       [PROC   ]                 [F1]  [retention-days]
                       [LOAD   ]                       [1           ]
                       [SUBR   ]                  [P] [,blocks]
                       [LIBRARY]                  [T] [,8     ]
                                                  [S]
                                                  [,ADD]
  or
FROMLIBR {name,ALL}, [SOURCE ] [,filename-2] [,I1] [,ADD      ] [,vol-id]
         {ALL     }  [PROC   ]               [F1]  [retention-days]
                     [LOAD   ]                     [1           ]
                     [SUBR   ]                [P] [,blocks]
                     [LIBRARY]                [T] [,8     ]
                                              [S]
                                              [,ADD]
```

Figure 4 (Part 2 of 5). IBM SCP Command Statement Formats

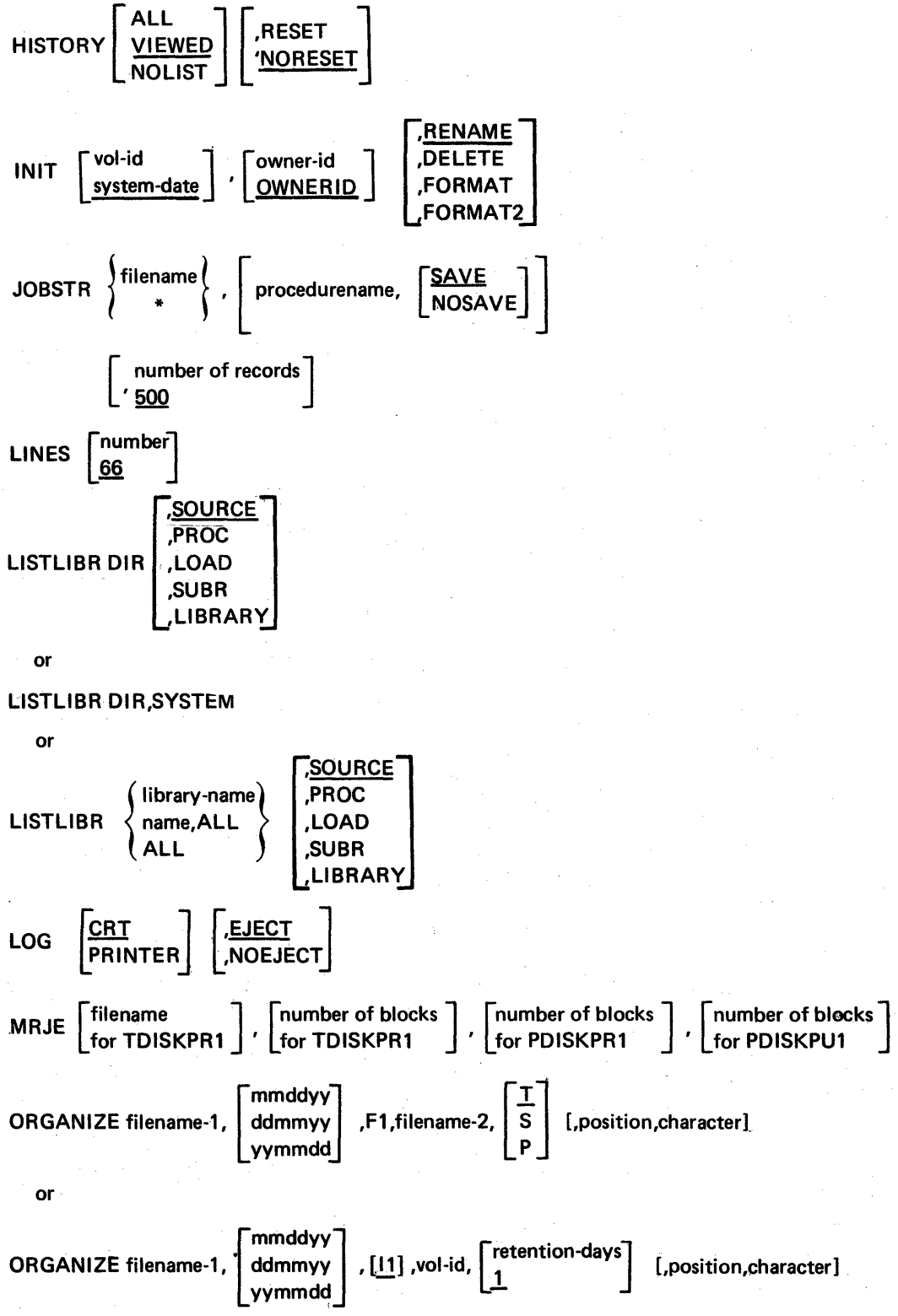


Figure 4 (Part 3 of 5). IBM SCP Command Statement Formats

```
OVERRIDE [ADDR-nn]  [ ,LINE- { C }  ]  [ ,SWTYP- { AA }  ]
                             { P }                { MA }
                             { R }                { MC }
                             { S }
                             { T }
```

Note: At least one parameter must be given in each OVERRIDE command statement.

```
REBUILD

RELOAD [vol-id]  [ mmddyy ]  [ ,filename    ]
                 [ ddmmyy ]  [ ,#LIBRARY    ]
                 [ yymmdd ]

REMOVE { library-name }  [ ,SOURCE  ]
       { name,ALL     }  [ ,PROC    ]
       { ALL          }  [ ,LOAD    ]
                         [ ,SUBR    ]
                         [ ,LIBRARY ]

RENAME filename-1,filename-2  [ ,mmddyy ]
                              [ ,ddmmyy ]
                              [ ,yymmdd ]

RESTORE [ALL],  [ filename-1 ]  [ ,mmddyy ]
                [ #SAVE      ]  [ ,ddmmyy ]
                                [ ,yymmdd ]
  or
RESTORE filename-2,  [ mmddyy ]  [ ,RECORDS,value-1 ]
                     [ ddmmyy ]  [ ,BLOCKS,value-2  ]
                     [ yymmdd ]

SAVE [ALL],  [ retention-days ]  [ filename-1 ]  ,vol-id
             [ 1              ]  [ #SAVE      ]

SAVE filename-2,  [ retention-days ]  [ mmddyy ]  ,vol-id
                  [ 1              ]  [ ddmmyy ]
                  [ ADD            ]  [ yymmdd ]

SET [value], [source-name],  [ MDY ]  [ ,mmddyy ]
                             [ DMY ]  [ ,ddmmyy ]
                             [ YMD ]  [ ,yymmdd ]
```

Note: At least one parameter must be given in each SET command statement.

```
SETMICR CYCLE- { Y }
               { N }
```

Note: The SETMICR command statement is used with the 1255 Magnetic Character Reader attachment and is described in *IBM System/32 1255 Magnetic Character Reader Reference and Logic Manual,* GC21-7692.

Figure 4 (Part 4 of 5). IBM SCP Command Statement Formats

```
SPECIFY  [ADDR-nn]  [ ,LINE- { C } ]  [ ,SWTYP- { AA } ]  [,ID-nnnnn]
                       { P }                   { MA }
                       { S }                   { MC }
                       { T }
```

Note: At least one parameter must be given in each SPECIFY command statement.

```
STATUS
SYSLIST [ PRINTER ]
        [ CRT     ]
        [ OFF     ]

TOLIBR filename, [ F1 ] , [ mmddyy ]  [,REPLACE]
                 [ I1 ]   [ ddmmyy ]
                          [ yymmdd ]

TRANSFER filename-1, [I1] , [ mmddyy ] ,ADD, [ filename-2 ]  [,date]
                            [ ddmmyy ]       [ filename-1 ]
                            [ yymmdd ]

  or

TRANSFER filename-1, [I1] , [ mmddyy ] , [NOADD] , [ value-1,value-2 ]  [ ,RECORDS,value-3 ]
                            [ ddmmyy ]             [                 ]  [ ,BLOCKS,value-4  ]
                            [ yymmdd ]

  or

TRANSFER filename-1,F1, [ mmddyy ]  ,vol-id  [ ,retention-days ]
                        [ ddmmyy ]           [ ,1              ]
                        [ yymmdd ]
```

Figure 4 (Part 5 of 5). IBM SCP Command Statement Formats

This page intentionally left blank.

This section describes all IBM SCP procedures supplied with IBM System/32 SCP to provide you with an easy method of using system functions. This section does not include the service procedures. The service procedures are described in Part 5 and in Appendix D.

The following information is given for each IBM SCP procedure:

- The function of the procedure
- The format of the command statement that evokes the procedure
- A description of the parameters of the command statement used to evoke the procedure

Examples are given for many of the command statements.

In the descriptions of command statement formats and parameters, capitalized words and letters, numbers, special characters, brackets, and braces have special meanings. Capitalized expressions must be entered as they appear in the descriptions. Sometimes numbers or nonalphabetic characters may appear in a capitalized expression—such numbers and characters must also be entered as they are shown. Words and expressions that are not capitalized must be replaced with a value that is appropriate to your job. The values you can use are listed in the parameter descriptions.

Brackets ([]) shown in command statement formats and parameters are not part of the parameters. Brackets can have two meanings: they can indicate that you can omit the parameter enclosed in brackets, and they can mean that if you use an expression enclosed in brackets, you must choose one of the values shown. For example,

[mmddyy
  ddmmyy
  yymmdd]

means that you need not give a date (the date parameter is optional), but if you choose to give a date, it must be in one of the three formats shown: mmddyy, ddmmyy, or yymmdd.

Underlining identifies default values. A default value is a value that is automatically substituted for an optional parameter that is omitted. For example, [I1 / F1] means that if neither I1 nor F1 is specified, F1 is used.

Braces ({ }) indicate that you must choose one of the values enclosed by the braces. For example, in the expression [PARM-{A / B}], the braces indicate that if you choose to enter the parameter, you must specify either A or B.

Note: In the preceding table (Figure 4) and in the descriptions that follow, the command statement formats often indicate that commas are required to separate parameters that are optional, whether the optional parameters are entered or not. Commas outside the brackets indicate positional parameters. The commas are shown in this manner to remind you that if a positional parameter is omitted, a comma must be entered in its place when another parameter is entered in a position that follows the position reserved for the omitted parameter.

ALTERBSC PROCEDURE

The ALTERBSC procedure alters the following BSC (binary synchronous communications) items:

Item	Parameter
Bits per second (bps) rate	BRATE
Modem clocking	CLOCK
Debug facility	DEBUG
Error retry count	ERC
Standby line	SLINE
Modem test	TEST
Non-USA	TONE

Additional BSC items that can be altered are included in the OVERRIDE procedure. (See index entry: *OVERRIDE procedure*.) To identify the current values in these parameters, use the STATUS procedure. (See index entry: *STATUS procedure.*)

The ALTERBSC procedure evokes the $SETCF utility (see index entry: *$SETCF utility program).*

Note: The ALTERBSC procedure is intended only for data communications programming that uses BSC. For background information on binary synchronous communications, see *General Information—Binary Synchronous Communications,* GA27-3004. For information on data communications programming, see *IBM System/32 Data Communications Programming Reference Manual,* GC21-7691.

ALTERBSC Command Statement Format

```
ALTERBSC [BRATE-{F}] [,CLOCK-{Y}] [,DEBUG-{Y}] [,ERC-{number}]
         {H}       {N}          {N}          {7 }
         [,SLINE-{Y}] [,TEST-{Y}] [,TONE-{Y}]
                  {N}         {N}         {N}
```

Note: Though each individual parameter is optional, at least one parameter must be specified. If a parameter is omitted and there is no default, the previous value is retained. If DEBUG-Y is specified, the system TRACE procedure (see index entry: *TRACE procedure*) is replaced by the BSC trace function. These options will remain in effect until changed by another ALTERBSC command statement, except the parameter DEBUG-Y, which is reset by IPL or by the TRACE procedure.

ALTERBSC Parameters

Parameter	Meaning
BRATE-F	Use the full rated speed of the modem.
H	Use only half the rated speed of the modem.
CLOCK-Y	The System/32 must provide the programmed clocking facility.
N	Modem has the clocking facility.

ALTERBSC Parameters (continued)

Parameter	Meaning
DEBUG-Y	Built-in debug facility is required, BSC trace requested.
N	Built-in debug facility is not required, BSC trace not requested.
ERC-number	Error retry count. The standard number of retries provided is seven (the default number); if more than seven are desired, enter
7	a number up to 255. Valid numbers are 7 through 255.
SLINE-Y	Switched standby line is used for a point-to-point line.
N	The nonswitched line is used.
TEST-Y	IBM modem is being used. Automatic wrap test includes modem testing when a permanent error occurs unless the user program specified a permanent error indicator for the BSC file.
N	Non-IBM modem is being used. Automatic wrap test does not include modem testing.
TONE-Y	Non-USA special tone is required for manual answering and automatic answering.
N	Non-USA special tone is not required for manual answering and automatic answering.

Notes:

1. If the SLINE-Y parameter is specified, then the line type (LINE) in the OVERRIDE procedure automatically defaults to a point-to-point switched line (LINE-S).
2. If the SLINE-N parameter is specified, then the line type (LINE) in the OVERRIDE procedure automatically defaults to the line type specified in the user program source statements (LINE-R).

ALTERSDL PROCEDURE

The ALTERSDL procedure alters the following SDLC (synchronous data link control) items in the system configuration record.

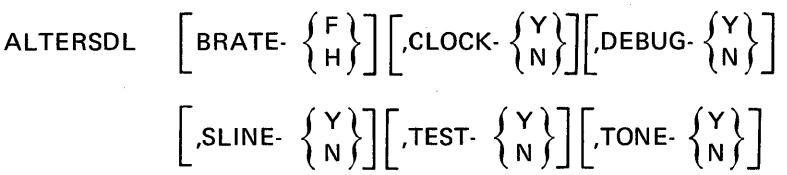
Item	Parameter
Bits per second (bps) rate	BRATE
Modem clocking	CLOCK
Debug facility	DEBUG
Standby line	SLINE
Modem test	TEST
Non-USA	TONE

Additional SDLC items that can be altered are included in the SPECIFY procedure. (See index entry: *SPECIFY procedure*). To identify the current values in these parameters, use the STATUS procedure. (See index entry: *STATUS procedure*).

The ALTERSDL procedure evokes the $SETCF utility (see index entry: *$SETCF utility program*).

Note: The ALTERSDL procedure is intended only for data communications programming that uses the SDLC. For background information on synchronous data link control, see *IBM Synchronous Data Link Control General Information,* GA27-3093. For information on data communication programming, see *IBM System/32 Data Communication Reference Manual,* GC21-7691.

ALTERSDL Command Statement Format



Note: Though each individual parameter is optional, at least one parameter must be specified. If a parameter is omitted, the previous value is retained. If DEBUG-Y is specified, the system TRACE procedure (see index entry: *TRACE procedure*) is replaced by the SDLC trace function. These options will remain in effect until changed by another ALTERSDL command statement, except the parameter DEBUG-Y, which is reset by IPL or by the TRACE procedure.

ALTERSDL Parameters

Parameter	Meaning
BRATE-F	Use the full rated speed of the modem.
H	Use only half the rated speed of the modem.
CLOCK-Y	The System/32 must provide the programmed clocking facility.
N	Modem has the clocking facility.
DEBUG-Y	Built-in debug facility is required, SDLC trace requested.
N	Built-in debug facility is not required, SDLC trace not requested.
SLINE-Y	Switched standby line is used for a point-to-point line.
N	The nonswitched line is used.
TEST-Y	IBM modem is being used. Automatic wrap test includes modem testing when a permanent error occurs.
N	Non-IBM modem is being used. Automatic wrap test does not include modem testing.
TONE-Y	Non-USA special tone is required for manual answering and automatic answering.
N	Non-USA special tone is not required for manual answering and automatic answering.

Notes:

1. If the SLINE-Y parameter is specified, then the line type (LINE) in the SPECIFY procedure automatically defaults to a point-to-point switched line (LINE-S).
2. If the SLINE-N parameter is specified, then the line type (LINE) in the SPECIFY procedure automatically defaults to a point-to-point nonswitched line (LINE-P).

APCHANGE PROCEDURE

The APCHANGE procedure reorganizes the library and data file area on disk in order to create usable space for adding library members from a diskette file. It also determines whether or not adequate data file space is available for use by the application program.

The functions of the APCHANGE procedure are:

- Reorganize the disk data file area by collecting unused space on the disk and making this space available at the end of the disk data file area.
- Determine whether the disk data file space is adequate for use by the application program.
- Delete non-SCP members and reorganize the library by collecting unused space between library members and making this space available at the end of the last active library member sector.
- Copy library members from a diskette file into the library.

The APCHANGE procedure evokes the $PACK utility to reorganize the disk and the $MAINT utility to delete library members, reorganize the library, and copy the library members from a diskette file into the library. (See index entries: *$PACK utility program* and *$MAINT utility program.*)

The APCHANGE procedure is especially useful when a System/32 is in a system sharing environment. See Appendix H, *System Sharing,* for a description of system sharing.

APCHANGE Command Statement Format

APCHANGE [blocks][,filename[,CLEAR]]

Note: At least one parameter must be specified on the APCHANGE command.

APCHANGE Parameters

blocks — Number of blocks of disk data file space that must be available for use by an application's data files. Data file space is reorganized first and then tested for availability.

Note: When this parameter is specified, files are moved from current locations to new locations. If the LOCATION parameter on a FILE statement is specified for a file that has been moved by the APCHANGE procedure, that location is no longer valid because the file location has changed. To determine the new location of the file, use the CATALOG procedure; this shows the file location in the VTOC entry. For information on the LOCATION parameter and the CATALOG procedure, see index entries: *FILE statement* and *CATALOG procedure.*

filename	Name of the diskette file that contains the library members to be copied into the reorganized library.
CLEAR	Specifies that all non-SCP library members are to be deleted from the library and the library is to be reorganized. When the CLEAR parameter is specified, the filename parameter must also be specified.

APCHANGE Examples

The following example reorganizes the disk, determines whether 200 disk blocks of data file space are available, reorganizes the library, and copies the library members that are contained in the diskette file, APPFILE, into the library:

```
APCHANGE 200,APPFILE
```

The following example reorganizes the disk, determines whether 125 disk blocks of data file space are available, deletes all non-SCP library members, reorganizes the library, and copies the library members that are contained in the diskette file, FORTAPP, into the library:

```
APCHANGE 125,FORTAPP,CLEAR
```

BACKUP PROCEDURE

BACKUP creates a diskette file that contains:

1. A stand-alone program that can change the directory and library size (for more information about changing directory and library size, see index entry: *RELOAD procedure*).

2. The reorganized library contents—unused space between members is collected at the end of the library.

To return the reorganized library to the disk, an IPL must be performed from the diskette(s) containing the backed up library, or the RELOAD procedure must be used (see index entry: *RELOAD procedure*). The vol-id of the first diskette containing the library becomes the vol-id of the disk file during the RELOAD operation.

The BACKUP procedure evokes the $BACK utility (see index entry: *$BACK utility program*).

Note: To determine the number of diskettes required to contain the library, see index entry: *calculating the number of backup diskettes required for the system.*

BACKUP Command Statement Format

BACKUP vol-id, [retention-days / 1] [,filename / ,#LIBRARY]

BACKUP Parameters

vol-id — Volume identification of the diskette(s). One to six *alphameric* (alphabetic or numeric) characters.

Note: When several diskettes are required for the BACKUP procedure and each diskette has a unique volume identification, the volume identification of the first diskette is the vol-id parameter you must specify for the BACKUP procedure. The vol-id parameter from the BACKUP procedure is used as the PACK parameter in the // FILE OCL statement for the backup library utility program ($BACK). When this program compares the PACK parameter with the volume identification of the second and succeeding diskettes, there is an error message (1493) if they are not equal. However, the system ignores the error and continues processing if you select option 0.

retention-days
1 — Length of the retention period (0 to 999 days) for the diskette file. The default is one day.

Note: A retention value of 999 makes a diskette file a permanent file.

filename — Specifies the name of the single file that is created on the diskette(s).

#LIBRARY — #LIBRARY is the name assigned to the created diskette file.

CATALOG PROCEDURE

The CATALOG procedure lists the disk or a diskette *VTOC (volume table of contents)* or a VTOC entry on the display screen or printer if either is assigned for listing from the system (see index entry: *SYSLIST procedure*). The disk VTOC contains an entry for each file on the disk, and a diskette VTOC contains an entry for each file on the diskette. Each VTOC entry identifies the related file by name, creation date, and location.

The CATALOG procedure evokes the $LABEL utility. A description of the VTOC display is provided in the description of $LABEL (see index entry: *$LABEL utility program*).

CATALOG Command Statement Format

CATALOG Parameters

ALL	Display all labels in the VTOC on the specified unit.
filename	Specifies the particular file whose VTOC information is to be displayed. If more than one file exists with the specified filename, the VTOC information for all those files will be displayed.
I1	Display the diskette VTOC.
F1	Display the disk VTOC.

COMPRESS PROCEDURE

The COMPRESS procedure causes all free space on the disk, except free space within the library file, to be put into a single area by copying each file as close to the library as possible. If the COMPRESS procedure does not go to normal end of job, it must be reissued immediately and go to normal end of job before any other jobs are run.

This procedure evokes the $PACK utility (see index entry: *$PACK utility program*).

Note: If LOCATION was specified in the FILE statement (see index entry: *FILE statement*) for a file moved by the COMPRESS procedure, the LOCATION specified is not valid after the COMPRESS procedure moves the file. Use the CATALOG procedure (see index entry: *CATALOG procedure*) to display the VTOC entries for files moved by COMPRESS to determine the new locations of the files.

COMPRESS Command Statement Format

COMPRESS

COMPRESS Parameters

None

CONDENSE PROCEDURE

The CONDENSE procedure collects all unused space between library members and makes this space available at the end of the last active library member (the library member area cannot be greater than 2867 blocks). To do this, the procedure moves all library members as close as possible to the beginning of the library member area.

When new library members are placed in the library, they are placed after the last active library member sector. Therefore, gaps or unused space are left in the library whenever either a library member is deleted or a library member is replaced by a member that requires a greater number of sectors. The CONDENSE procedure makes all unused space available for additional library members.

The CONDENSE procedure evokes the $MAINT utility program (see index entry: *$MAINT utility program*).

Note: If a permanent disk error occurs while the CONDENSE procedure is executing, there is no error recovery. The library must be reloaded from diskette (see index entry: *RELOAD procedure*).

CONDENSE Command Statement Format

CONDENSE

CONDENSE Parameters

None

CONVERT PROCEDURE

The CONVERT procedure converts the diskette header labels that were created prior to version 5 to a version 5 format.

Note: Unpredictable results may occur if diskette files with version 5 format header labels are processed by a preversion 5 SCP. A diskette file created by the $MAINT utility program (FROMLIBR procedure) in version 5 of the SCP, for example, *cannot* be used as input to the $MAINT utility program (TOLIBR procedure) in version 4 of the SCP.

The CONVERT procedure evokes the $CNVRT utility (see index entry: *$CNVRT utility program*).

CONVERT Command Statement Format

CONVERT

CONVERT Parameters

None

COPYI1 PROCEDURE

The COPYI1 procedure causes all files on a single diskette or an individual file on a single diskette to be copied to one or more output diskettes. A work space large enough to contain the file(s) to be copied must be available on the disk. Files from the copied diskette are placed contiguously on the receiving diskette(s). The receiving diskette(s) must be in the same format (512-bytes per sector extended format or 128-bytes per sector basic data exchange format) as the diskette being copied.

COPYI1 can be used to create a backup diskette file or to gather all unused space on an input diskette into a single free-space on the output diskette(s).

Important diskettes with permanent files are the diskettes normally copied. Because diskettes can develop surface irregularities as they undergo the wear of continued use, it is a good idea to copy your important files soon after they are created.

COPYI1 evokes the $DUPRD utility (see index entry: *$DUPRD utility program*).

COPYI1 Command Statement Format

Use	Format
Copy all diskette files to one or more output diskettes	COPYI1 [ALL] ,,vol-id, [DELETE], [PRESERVE] [,number of copies / ,1]
Copy specific diskette file to one or more output diskettes	COPYI1 filename, [mmddyy / ddmmyy / yymmdd] ,vol-id ,, [PRESERVE] [,number of copies / ,1]

COPYI1 Parameters

ALL Indicates that all files on the diskette are to be copied to one or more output diskettes.

filename Name of the single file to be copied to one or more output diskettes.

mmddyy or ddmmyy or yymmdd Creation date of the input file. This date must be in the same format as that of the input file. This date is used to verify that the correct file is on the input diskette. (The creation date of the output file will be the same as that of the input file.)

vol-id Volume label of output diskette(s); one to six alphameric characters.

DELETE Any expired file will not be copied. (DELETE can be specified only when ALL is specified.)

Note: If a multivolume file exists on the diskette, the DELETE parameter is ignored.

PRESERVE Indicates that for each file copied, the end of extent is preserved at the same relative displacement past the end of data on the output diskette(s) as it was on the input diskette.

number of copies 1 Specifies the number of output diskettes to be copied from one input diskette. The value specified can be 1 through 99. 1 is the default.

COPYI1 Example

In order to copy the file entitled PAYROLL (dated October 14, 1974) onto a diskette with a volume identifier of VOL001, you could enter:

```
COPYI1 PAYROLL,101474,VOL001
```

Note: In the preceding example, PAYROLL is not a multivolume file. If PAYROLL were a multivolume file, a separate COPYI1 command statement would be required for each diskette of the file.

CREATE PROCEDURE

The CREATE procedure creates a message load member from a message source member. A message load member contains messages that can be retrieved by user or IBM programs. (For information on how to create a message source member, see *Message Source Member* and *An Example of Creating a Message Source and Load Member* under index entry: *$MGBLD utility program*.) The CREATE procedure evokes the $MGBLD utility (see index entry: *$MGBLD utility program*).

CREATE Command Statement Format

CREATE sourcename [,REPLACE]

CREATE Parameters

sourcename	Name of the existing source member that contains a control statement and message text statement(s)
REPLACE	Message load member to be created to replace an existing message load member that has the same name

DATE PROCEDURE

The DATE procedure sets either the system date or the job (program) date. If the DATE command statement is given after an IPL and before a LOAD statement, the system date is set to the date specified. If the DATE command statement is given between the LOAD and RUN OCL statements in a job stream, the program date is set to the date specified and reset to the system date after the program ends.

The date established for the system or a program is printed on printed output and is used to determine file retention periods for diskette files (see the RETAIN parameter for diskette files under index entry: *// FILE statement*).

The function of the DATE procedure is identical to that of the // DATE statement (see index entry: *// DATE statement*).

DATE Command Statement Format

DATE $\left\{\begin{matrix}\text{mmddyy}\\\text{ddmmyy}\\\text{yymmdd}\end{matrix}\right\}$

DATE Parameters

mmddyy	Month-day-year
ddmmyy	Day-month-year
yymmdd	Year-month-day

Note: You must use the current system date format. Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems. The SET procedure can be used to change the system date format (see index entry: *SET procedure*). The STATUS procedure can be used to determine the current date format (see index entry: *STATUS procedure*).

DELETE PROCEDURE

The DELETE procedure causes the space occupied by the named diskette or disk file(s) to be made available. It also provides the option of erasing the contents of a data file. The system file #LIBRARY cannot be deleted with this procedure. This procedure evokes the $DELET utility (see index entry: *$DELET utility program*).

DELETE Command Statement Format

DELETE filename, [F1 / I1] , [SCRATCH / REMOVE / ERASE] [,mmddyy / ,ddmmyy / ,yymmdd]

DELETE Parameters

filename	Name of the file to be deleted from the disk or diskette(s). ALL cannot be used as a filename.
F1	The file to be deleted is on the disk.
I1	The file to be deleted is on one or more diskettes. If the file is a multivolume file, you are prompted to insert the required diskettes.
SCRATCH	If the file is on a diskette, the expiration date is changed to the current job date. If the file is on the disk, the VTOC entry for the file is removed.
REMOVE	The VTOC entry for the file is removed.

ERASE — Requests elimination of all data in the deleted file by replacing all bytes within the physical extents of the file with binary zeros. Also removes the VTOC entry for the file.

mmddyy
ddmmyy
yymmdd — Creation date of the file to be deleted. This date must be in the same format as the system date if the file is on the disk; it must be in the same format as the creation date of the diskette file if a diskette file is being deleted. You can use the STATUS command statement to display the system date and the CATALOG command statement to display creation dates of disk and diskette files (see index entries: *CATALOG procedure* and *STATUS procedure*).

Note: If no date is specified and more than one file with the given filename exists on the disk, the operator will have the option to either delete all of the files named by filename or to cancel the job.

DELETE Example

To delete the diskette file JOE (dated September 13, 1974) you could enter the following:

```
DELETE JOE,,REMOVE,091374
```

DISPLAY PROCEDURE

The DISPLAY procedure causes all or part of a disk file to be listed on the display screen or on the printer, depending on which is being used to display output (see index entry: *SYSLIST procedure*).

This procedure evokes the $COPY utility (see index entry: *$COPY utility program*).

Note: If you use DISPLAY to list a disk segment of an offline multivolume file (see index entry: *offline multivolume file*), the list will include variable system data.

DISPLAY Command Statement Format

Use	Format
Display a file	DISPLAY filename [,mmddyy / ,ddmmyy / ,yymmdd]
Display records by relative record number	DISPLAY filename, [mmddyy / ddmmyy / yymmdd] ,RECORD,value-1 [,value-2]

DISPLAY Parameters

filename	Name of the file to be displayed or printed.
mmddyy ddmmyy yymmdd	Creation date of file to be displayed or printed. If date is not specified, then the filename with the most recent date is displayed or printed.
RECORD	The records from the file are to be displayed or printed based on their relative record number.
value-1	Number of the first record to be displayed or printed. Valid for sequential, indexed, and direct files.
value-2	Number of the last record to be displayed or printed. Valid for sequential, indexed, and direct files. If value-2 is omitted, the listing continues until end of file is reached.

DISPLAY Example

To display or print the first one hundred records of the most recent file created with the name JOE, you would enter:

```
DISPLAY JOE,,RECORD,1,100
```

FROMLIBR PROCEDURE

The FROMLIBR procedure creates a file from members contained in the library, or adds library members to a file created from library members. Files created by the FROMLIBR procedure can be processed by the TOLIBR procedure (see index entry: *TOLIBR procedure*) to place members back in the library.

The FROMLIBR procedure evokes the $MAINT utility (see index entry: *$MAINT utility program*).

Note: If you use the FROMLIBR procedure to copy library members from the library to a file, you can copy the members from the file back to the library only by using the TOLIBR procedure or $MAINT.

FROMLIBR Command Statement Format

Use	Format
Copies a non-SCP library member or adds a non-SCP library member to a sequential file.	FROMLIBR library-name, [SOURCE / PROC / LOAD / SUBR / LIBRARY] , [filename-1 / library-name] , [ADD / retention-days / 1] [I1 / F1] , [P / T / S] [,blocks / ,8] [,ADD] ,vol-id

Use	Format
Copies or adds all non-SCP members, or copies or adds all non-SCP members having names beginning with name.	FROMLIBR {name,ALL / ALL} , [SOURCE / PROC / LOAD / SUBR / LIBRARY] , [filename-2 / name] , [ADD / retention-days / 1] [I1 / F1] , [P / T / S] [,blocks / ,8] [,ADD] ,vol-id

FROMLIBR Parameters

library-name	Name of the non-SCP library member being copied from the library.
name,ALL	All non-SCP members with names beginning with the indicated characters are to be copied. Up to seven characters can be used. Example: PAYR,ALL refers to non-SCP members having names that begin with PAYR.
ALL	All non-SCP members are copied from the library. *Note:* All non-SCP members include IBM-supplied non-SCP members (such as program product members) as well as members you have created.
SOURCE	Source members are copied.
PROC	Procedure members are copied.
LOAD	Load members are copied.
SUBR	Subroutine members are copied.
LIBRARY	All types of members (SOURCE,PROC,LOAD and SUBR) are copied.
filename-1	Name of the file being created. If the filename is not specified, the name specified for library-name is assumed.
filename-2	Name of the file being created. If not specified, name is assumed. If ALL is specified (all non-SCP members are copied from the library) and filename-2 is not specified, you are prompted for filename-2.
I1	Output file to be created on the diskette.
F1	Output file to be created on the disk.
retention-days 1	Length of the retention period (0 to 999 days) for the diskette file. If I1 is specified or assumed and retain is not given, default is one day. *Note:* A retention value of 999 makes a diskette file a permanent file. Retention cannot be specified if ADD if specified. ADD cannot be specified if retention is specified.
P	Permanent retention on disk.
T	Temporary retention on disk.
S	Scratch retention on disk.

ADD	Add library member(s) to an existing file that contains library members. *Note:* When adding a member to a disk file, the file must contain enough unused space to hold the member. When adding a member to a diskette file, the file must be the last active file on the diskette. Retention cannot be specified if ADD is specified. ADD cannot be specified if retention is specified.
vol-id	Diskette volume label. One to six alphameric characters.
blocks 8	Number of blocks to allocate for the output file. Ignored if ADD is specified (see preceding description of ADD). Default is eight blocks.

FROMLIBR Examples

To copy the payroll application source programs to diskette, all beginning with the characters PAY, you would specify:

```
FROMLIBR PAY,ALL,,,,,VOL001
```

A sequential system file on diskette named PAY containing the payroll application programs and procedures would be created.

To add all library members whose names begin with the characters PA to a diskette file named PAYSAVE you would specify:

```
FROMLIBR PA,ALL,LIBRARY,PAYSAVE,,ADD,PACKID
```

HISTORY PROCEDURE

The HISTORY procedure lists the contents of the history file on the display screen or on the printer, depending on which is listing output (see index entry: *SYSLIST procedure*). Recorded in the history file are the OCL statements, utility control statements, and procedures executed by the SCP; all messages issued; and all user responses to messages and prompts. The entire history file (ALL parameter) can be displayed, just the entries previously displayed to the operator (VIEWED parameter), or none of the entries (NOLIST parameter). Items previously displayed to the operator consist of items such as OCL statements and messages that were logged (displayed) as they were entered or issued (see index entry: *LOG procedure*). The file may be cleared after listing (RESET parameter).

Any messages issued when BSC is active will not be recorded in the history file.

This procedure evokes the $HIST utility (see index entry: *$HIST utility program*).

HISTORY Command Statement Format

```
HISTORY [ALL    ] [,RESET  ]
        [VIEWED ] [,NORESET]
        [NOLIST ]
```

HISTORY Parameters

ALL — The entire contents of the history file will be listed including OCL statements in procedures. If ALL is *not* specified, only items previously displayed to the operator are displayed.

VIEWED — Only the items previously displayed to the operator are listed.

NOLIST — The history file is not displayed. This parameter may be used with the RESET parameter to clear the history file.

RESET — The history file will be cleared after it is displayed.

NORESET — The history file will not be cleared.

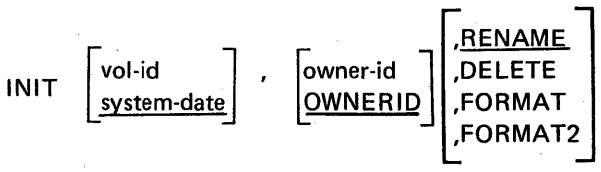
INIT PROCEDURE

The INIT procedure prepares (initializes) a diskette for use. It does this by performing some or all of the following functions:

- Writing sector addresses on the diskette
- Checking for defective tracks
- Assigning new track numbers when a track has a defective sector
- Writing a name on each diskette to identify the diskette
- Formatting track 0

This procedure evokes the $INIT utility (see index entry: *$INIT utility program*).

INIT Command Statement Format



INIT Parameters

vol-id system-date	If specified, it consists of one to six alphameric characters. The vol-id is left-adjusted and padded with blanks when placed in the volume label. When DELETE is specified, vol-id is checked for a match. If no value is specified, the system date is used as a default value.
owner-id OWNERID	Up to eight alphameric characters can be specified. These are left-justified and padded with blanks in the owner identification field of the volume label of the diskette. If owner-id is omitted and RENAME, FORMAT, or FORMAT2 is specified, owner-id is written as OWNERID.
RENAME	Allows the diskette to be renamed (vol-id and owner-id). Files and their labels are not affected.
DELETE	Deletes active files, thereby freeing up space on the diskette (initializes track 0 on the diskette).

This page intentionally left blank

INIT Parameters

vol-id system-date	If specified, it consists of one to six alphameric characters. The vol-id is left-adjusted and padded with blanks when placed in the volume label. When DELETE is specified, vol-id is checked for a match. If no value is specified, the system date is used as a default value.
owner-id OWNERID	Up to eight alphameric characters can be specified. These are left-justified and padded with blanks in the owner identification field of the volume label of the diskette. If owner-id is omitted and RENAME, FORMAT, or FORMAT2 is specified, owner-id is written as OWNERID.
RENAME	Allows the diskette to be renamed (vol-id and owner-id). Files and their labels are not affected.
DELETE	Deletes active files, thereby freeing up space on the diskette (initializes track 0 on the diskette).

FORMAT — Formats the entire surface of the diskette in the 128-bytes per sector basic data exchange format (see Appendix C). Tracks are renumbered for each track with a surface defect. If track 0 (index track) or more than 2 tracks have defects, the diskette is not initialized and no label of any kind is written (the diskette is not usable).

Note: If FORMAT is specified for one diskette in a multivolume file, it must be specified for all diskettes in the file.

FORMAT2 — Formats the surface of the diskette in the extended format. Extended format diskettes have eight 512-byte sectors per data track. The index track is formatted into twenty-six 128-byte sectors; the data tracks are 1 through 74. Position 76 of the volume label (VOL1) contains a 2 if a diskette is formatted in 512-byte data sectors. The physical record length field (position 34) of the data set labels also contains a 2 if the diskette is formatted in 512-byte data sectors. (The formats of the diskette volume labels and data set labels are given in the *IBM Diskette General Information Manual,* GA21-9182—also see Appendix C of this manual.) However, diskettes formatted in 512-byte data sectors cannot be used for basic data exchange files.

Tracks are renumbered for each track with a surface defect. If track 0 (index track) or more than 2 tracks have defects, the diskette is not initialized, and no label of any kind is written (the diskette is not usable).

Notes:

1. If FORMAT2 is specified for one diskette in a multivolume file, it must be specified for all diskettes in the file.
2. If a diskette read error occurs on a FORMAT2 diskette, you cannot correct the bad sector. You can either rerun the job using a different diskette or retry the same diskette.

INIT Examples

To place a volume identification of 934613 and an owner identification of JOESDISK on a diskette you would enter:

```
INIT 934613,JOESDISK
```

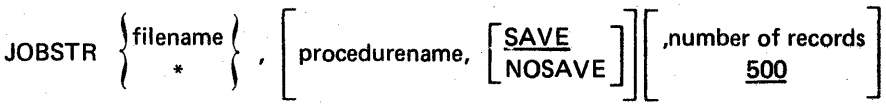
RENAME is the default and the diskette would be labeled (volume label) but not initialized. An example of initializing follows:

```
INIT 843163,,FORMAT
```

JOBSTR PROCEDURE

The JOBSTR procedure transfers, to the System/32 library, a job stream that contains procedure and source members created either on a diskette or on cards. Included in the JOBSTR procedure is an option you can specify to execute a procedure and then save or delete that procedure from the library.

JOBSTR Command Statement Format

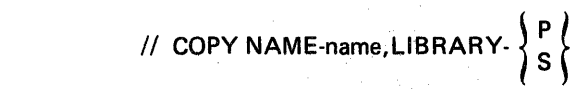


JOBSTR Parameters

filename	Name of the basic data exchange diskette file that contains the job stream.
*	Indicates that the job stream is on cards. *Note:* If neither the filename parameter nor the * parameter is specified, you are prompted for the parameter.
procedurename	Name of the procedure to execute.
SAVE	Saves the procedure named on the procedurename parameter in the library.
NOSAVE	Deletes the procedure named on the procedurename parameter from the library.
number of records 500	Specifies the number of records that the disk file is to contain when the job stream is transferred from diskette. The system uses a temporary disk file to transfer the job stream from diskette to the library. The number of records must be specified if the input file has more than the 500 record default.

The JOBSTR procedure evokes the queued job stream card-to-library utility program ($QJOB) for job stream input on cards. For job stream input on diskette, the JOBSTR procedure evokes the $BICR, $MAINT, and $DELET utility programs (see index entries: *$BICR, $MAINT, $DELET utility program*).

The job stream you create can consist of multiple procedure or source members. Each procedure or source member must begin with a COPY statement and end with a CEND statement. The format of the COPY statement is:



where name is the member name and P or S indicates procedure member or source member.

The format of the CEND statement is:

```
// CEND
```

The CEND statement is valid only as the last statement for a procedure or source member. It is not valid within a procedure or source member.

The /* statement must be the last statement in a job stream created on cards. This statement must immediately follow the last CEND statement.

A job stream created on diskette must be in 128-bytes-per-sector basic data exchange format and the record length must be between 40 and 120.

A diskette file could contain the following job stream:

```
// COPY NAME-P1,LIBRARY-P
      .
      .
      .
// CEND
// COPY NAME-P2,LIBRARY-P
      .
      .
      .
// CEND
// COPY NAME-S1,LIBRARY-S
      .
      .
      .
// CEND
```

The job stream is transferred to the System/32 library when the JOBSTR command statement is entered. For a diskette file, JBS, that contains the previous job stream you could enter:

```
JOBSTR JBS
```

and the procedure members (P1 and P2) and the source member (S1) would be placed in the system library.

Enter JOBSTR JBS,P2 and the procedure members (P1 and P2) and the source member (S1) are placed in the system library; procedure P2 is executed and then saved (SAVE is the default when omitted).

Enter JOBSTR JBS,P2,NOSAVE and the procedure members (P1 and P2) and the source member (S1) are placed in the system library; procedure P2 is executed and then deleted from the library.

JOBSTR Example

The following example shows a job stream created on either diskette or cards. The statements are numbered to correspond to the explanations following:

1. COPY statement indicating a procedure member named JOBSTRM.

   Steps 2 through 8 — Contents of procedure member JOBSTRM.

2. Command statement to compile a FORTRAN program from a source member.

3. Command statement to load and run the compiled FORTRAN program.

4. Data used as input to the program.

5. Indicates the end of data for the program.

6. Command statement to load and run the FORTRAN compiler with source statements in the job stream.

7. Indicates the end of the source statements.

8. Command statements in the job stream.

9. Indicates the end of the procedure member.

10. Source member to be moved to the source library.

    Note: A job stream on cards must contain a /* statement after the CEND statement.

LINES PROCEDURE

The LINES procedure provides a means of modifying the printer lines per page. This procedure contains a FORMS OCL statement (see index entry: *// FORMS statement*).

LINES Command Statement Format

LINES [number / 66]

LINES Parameters

number — Specifies the number of lines to be printed per page. The value specified can be 1 through 84.

Note: See index entry: *// FORMS statement* for the way the value specified determines the actual number of lines printed per page.

66 — The default value for number is 66.

LISTLIBR PROCEDURE

The LISTLIBR procedure allows you to list the contents of the system library. Either directory entries or contents of individual members can be listed.

This procedure evokes the $MAINT utility (see index entry: *$MAINT utility program*).

Note: If the display screen is used for listing the library, only the first 40 bytes of each LISTLIBR output line are displayed. To ensure that all the information in a library member or directory entry is listed, use the printer to list the output. You can use the STATUS procedure (see index entry: *STATUS procedure*) to determine where system output is currently listed (that is, what the current SYSLIST assignment is); and the SYSLIST procedure (see index entry: *SYSLIST procedure*) to change the current SYSLIST assignment.

LISTLIBR Command Statement Format

Use	Format
Directory entries are to be listed.	LISTLIBR DIR [,SOURCE / ,PROC / ,LOAD / ,SUBR / ,LIBRARY]
System information is to be listed from the directory.	LISTLIBR DIR,SYSTEM
Library members and their directory entries are to be listed.	LISTLIBR {library-name / name,ALL / ALL} [,SOURCE / ,PROC / ,LOAD / ,SUBR / ,LIBRARY]

LISTLIBR Parameters

DIR	Directory entry is to be listed.
library-name	Name of library member to be listed.
name,ALL	Specifies the characters that the library member names to be listed begin with. Up to seven characters can be used.
ALL	Specifies that all members of the specified type(s) be listed.
SYSTEM	System information in the library directory. Valid with DIR only.
SOURCE	Source directory entries, if DIR is specified; otherwise, indicates contents of source member(s).
PROC	Procedure directory entries, if DIR is specified; otherwise, indicates contents of procedure member(s).
LOAD	Load directory entries, if DIR is specified; otherwise, indicates contents of load member(s).
SUBR	Subroutine directory entries, if DIR is specified; otherwise, indicates contents of subroutine member(s).
LIBRARY	All types of directory entries (SYSTEM, SOURCE, PROC, LOAD, and SUBR), if DIR is specified; otherwise, indicates contents of all member types (SOURCE, PROC, LOAD, and SUBR).

LISTLIBR Examples

To list the procedure member JOE, you would enter:

```
LISTLIBR JOE,PROC
```

To list all procedure members which have names beginning with PA, you would enter:

```
LISTLIBR PA,ALL,PROC
```

To list the source, procedure, load, subroutine, and system directories, you would enter:

```
LISTLIBR DIR,LIBRARY
```

LOG PROCEDURE

The LOG procedure specifies where messages and OCL statements are to be displayed (on the display screen only or on the display screen and the printer), and specifies whether to skip to line 1 of the next page at end of job. The LOG procedure performs the same function as the LOG OCL statement (see index entry: *// LOG statement*).

LOG Command Statement Format

LOG Parameters

CRT	Display messages and statements on the display screen.
PRINTER	Print messages and statements and display them on the display screen.
	Note: When the BSCA is active, the messages are not printed.
EJECT	Skip to line 1 of next page at end of job.
NOEJECT	Do not skip to line 1 of next page at end of job.

ORGANIZE PROCEDURE

The ORGANIZE procedure performs one of the following functions:

- Copies a disk file to another area on the disk
- Copies a disk file to another area on the disk and deletes specified records
- Copies a disk file to a diskette
- Copies a disk file to a diskette and deletes specified records

If the input file is sequential, the output file is sequential. However, if reorganizing a sequential input file, records must be specified for deletion. If the input file is indexed, the output file is indexed, and the data records in the output file are in the same sequence as the keys in the index. A direct input file cannot be reorganized.

The ORGANIZE procedure evokes the $COPY utility (see index entry: *$COPY utility program*).

ORGANIZE Command Statement Format

Use	Format
Reorganize a disk file as another disk file.	ORGANIZE filename-1, [mmddyy / ddmmyy / yymmdd] ,F1,filename-2, [T / S / P] [,position,character]
Reorganize a disk file as a diskette file.	ORGANIZE filename-1, [mmddyy / ddmmyy / yymmdd] , [I1] ,vol-id, [retention-days / 1] [,position,character]

ORGANIZE Parameters

filename-1	Name of the disk file to be reorganized (and name of the diskette file created if reorganizing as a file on diskette).
mmddyy ddmmyy yymmdd	The creation date of the input file. If this parameter is omitted, the most recently created file with the name specified in filename-1 is the one that is reorganized.
F1	The disk will contain the reorganized copy.
I1	The diskette will contain the reorganized copy. Default is I1.
filename-2	Name of the disk file to contain the reorganized copy.
vol-id	Identifies the diskette by volume label. One to six alphameric characters. Valid only if I1 is specified.

T	Temporary retention on the disk.
S	Scratch retention on the disk.
P	Permanent retention on the disk.
retention-days 1	Number of days (0 to 999) in the retention period for the diskette file. Default is 1.
	Note: A retention value of 999 makes a diskette file a permanent file.
	Note: If the input file is sequential, then record deletion must be specified.
position	Requests deletion of records having a specified character (*character*) in the position specified. Position can be any position in the record (first position is 1, second 2, and so on) to a maximum of 999. These records will not be copied to the reorganized file.
character	Character can be any one of the standard characters, or the three characters Xdd, where X is constant and dd is the hexadecimal equivalent of the character. Records containing the specified character in the position specified by the position parameter are not copied to the reorganized file. See the *note* in the position parameter.

ORGANIZE Examples

To reorganize the indexed file, PAYROLL, into a permanent disk file called PAYR, you could enter:

```
ORGANIZE PAYROLL,,F1,PAYR,P
```

To reorganize the file called JOE and place the organized copy (except records containing a D in record position 13) on diskette volume 123456, you could enter:

```
ORGANIZE JOE,,,123456,999,13,D
```

In the preceding example neither F1 nor I1 is specified in the third parameter, so the default is I1. Also, the file is to be retained permanently, so retention-days 999 is specified.

Note: A date is not specified in either of the two preceding examples. Consequently, if more than one file named JOE or PAYROLL exists on the disk, the most recently created of the files named JOE or PAYROLL will be reorganized.

OVERRIDE PROCEDURE

The OVERRIDE procedure is used to override BSC parameters.

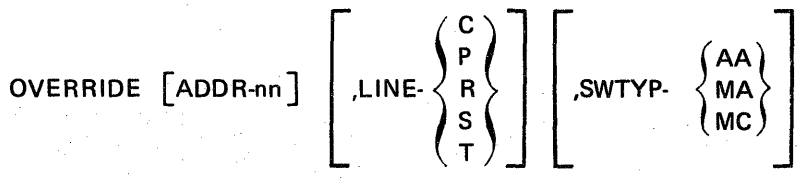
Item	Parameter
Tributary station address	ADDR
Line type	LINE
Switch type	SWTYP

Additional BSC items that can be altered are included in the ALTERBSC procedure (see index entry: *ALTERBSC procedure*). To identify the current values in these parameters, use the STATUS procedure. (See index entry: *STATUS procedure.*)

The OVERRIDE procedure evokes the $SETCF utility (see index entry: *$SETCF utility program*).

Note: The OVERRIDE procedure is intended only for data communications programming that uses BSC. For background information on binary synchronous communications, see *General Information—Binary Synchronous Communications,* GA27-3004. For information on data communications programming, see *IBM System/32 Data Communications Reference Manual,* GC21-7691.

OVERRIDE Command Statement Format



Notes:

1. Though each individual parameter is optional, at least one parameter must be specified.
2. To reset the ADDR parameter to the addressing characters specified by the user program specifications, reenter a valid OVERRIDE command omitting the ADDR parameter. The addressing characters then default to the user program specifications.

OVERRIDE Parameters

ADDR-nn	Hexadecimal equivalent of one of the pair of tributary station addressing characters. See Appendix G for the System/32 tributary station polling and addressing characters. Defaults to the user program specifications.
LINE-C	CDSTL (connect data set to line) switched line (World Trade only)
P	Point-to-point nonswitched line.
R	Line type specified in the user program source statements.
S	Point-to-point switched line.
T	Tributary station on multipoint line.

SWTYP-AA — If switched line (LINE-C or LINE-S) is specified and the modem is in auto-answer mode, then the System/32 automatically answers the call.

MA — If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually answers the call.

MC — If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually initiates the call.

Notes:

1. If LINE-C or LINE-S is specified, the SWTYP parameter must be specified.
2. If the SWTYP parameter is specified, then LINE-C, or LINE-S must be specified. However, if the line type was set previously to a switched line (LINE-C or LINE-S), then the line type does not have to be respecified.
3. If the line type is LINE-R, then both the line type and switch type are determined from the user program source statements and neither line type nor switch type is required.
4. If LINE-P or LINE-T is specified, then the switch type (SWTYP) automatically defaults to 0 (zero).
5. The line type defaults to the line type specified in the user program source statements (LINE-R) if the standby line (SLINE) is specified in the ALTERBSC procedure as SLINE-N.
6. The line type defaults to a point-to-point switched line (LINE-S) if the standby line (SLINE) is specified in the ALTERBSC procedure as SLINE-Y.

REBUILD PROCEDURE

The REBUILD procedure allows you to restore system information related to output files being processed at the time of a system failure, such as one caused by a power failure or inadvertent IPL. *The REBUILD procedure must be the first procedure run after a system failure, otherwise the information will not be restored.* The information restored by REBUILD is essential if you want to obtain data contained in output files being processed at the time of the system failure.

The REBUILD procedure evokes the $REBLD utility program. For a more complete description of the function of REBUILD, see index entry: *$REBLD utility program.*

REBUILD Command Statement Format

REBUILD

REBUILD Parameters

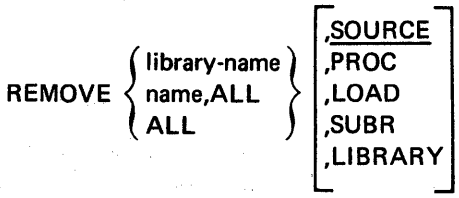
None

REMOVE PROCEDURE

The REMOVE procedure deletes the specified library member(s), unless they are SCP members. The space that was occupied by the deleted members can be used for new members, provided there are not active members physically located after the deleted ones in the library. If active members are located after deleted members you can use the CONDENSE procedure to relocate these active members and combine all unused space at the end of the library (see index entry: *CONDENSE procedure*).

This procedure evokes the $MAINT utility (see index entry: *$MAINT utility program*).

REMOVE Command Statement Format



REMOVE Parameters

library-name — Name of the non-SCP library member to be deleted.

name,ALL — Beginning characters of names of non-SCP members to be deleted. Up to seven characters can be used.

ALL — Remove all non-SCP members of the specified type or all types.

SOURCE — Remove non-SCP source member(s).

PROC — Remove non-SCP procedure member(s).

LOAD — Remove non-SCP load member(s).

SUBR — Remove non-SCP subroutine member(s).

LIBRARY — Remove non-SCP members of all member types (SOURCE, PROC, LOAD, and SUBR).

REMOVE Examples

To delete the non-SCP procedure member named JOE from the library, you would enter:

```
REMOVE JOE,PROC
```

To delete the non-SCP members in the library that are named SAM, you would enter:

```
REMOVE SAM,LIBRARY
```

To delete all non-SCP members in the library beginning with characters PAY, you would enter:

```
REMOVE PAY,ALL,LIBRARY
```

RENAME PROCEDURE

The RENAME procedure changes the name of an existing data file on disk. All of the other file attributes such as file location, creation date, file type, file length, and file retention remain unchanged.

This procedure evokes the $RENAM utility (see index entry: *$RENAM utility program*).

RENAME Command Statement Format

```
RENAME filename-1,filename-2 [,mmddyy]
                            [,ddmmyy]
                            [,yymmdd]
```

RENAME Parameters

filename-1	Current name of the file.
filename-2	New name of the file.
mmddyy ddmmyy yymmdd	Creation date of the disk file. If not specified, the last file created with the name given in filename-1 will be renamed.

RENAME Examples

To rename a data file from JOE to JOHN, enter:

```
RENAME JOE,JOHN
```

To rename a data file from JOE which was created on 2/10/78 to JOHN and there exists on the disk more than one file by the name of JOE, enter:

```
RENAME JOE,JOHN,021078
```

Note: The filename-2 parameter must not be the name of an existing file on disk at the time $RENAM is evoked.

RESTORE PROCEDURE

The RESTORE procedure restores on the disk a diskette file that was copied from the disk by one of the following:

- the ORGANIZE procedure (see index entry: *ORGANIZE procedure*)
- the SAVE procedure (see index entry: *SAVE procedure*)
- the $COPY utility (see index entry: *$COPY utility program*)

The RESTORE procedure can also be used to restore to the disk one or all of the entire group of files previously saved by a SAVE ALL request.

When only one file is to be restored, you can change the space allocation of the disk file by specifying the RECORDS or BLOCKS parameter in the RESTORE command statement. If the diskette file size was increased, beyond the original file capacity, the RECORDS or BLOCKS parameter must be used.

A RESTORE request reconstructs a file on the disk with the same attributes, except location (see index entry: *FILE statement* for a description of the LOCATION parameter), that the file had before it was copied to the diskette.

Messages to insert a diskette for multivolume files are automatically displayed as required, with appropriate label and volume-sequence-number checking.

This procedure evokes the $COPY utility (see index entry: *$COPY utility program*).

This page intentionally left blank

RESTORE Command Statement Format

Use	Format
Restore all previously saved files.	RESTORE [ALL] , [filename-1 / #SAVE] [,mmddyy / ,ddmmyy / ,yymmdd]
Restore a previously saved single file.	RESTORE filename-2, [mmddyy / ddmmyy / yymmdd] [,RECORDS, value-1 / ,BLOCKS, value-2]

RESTORE Parameters

ALL	All data files previously saved are to be restored to the disk.
filename-1 #SAVE	Name associated with the entire set of files previously saved on the diskette by the SAVE (SAVE ALL) procedure. #SAVE is the default.
filename-2	Name of the single diskette file that is to be restored to the disk.
mmddyy ddmmyy yymmdd	Creation date of the diskette file.
RECORDS	Requests that the disk file be made large enough to contain the number of records indicated by value-1.
value-1	Specifies the number of records that the disk file is to accommodate.
BLOCKS	Requests that the disk file be made large enough to contain the number of blocks indicated by value-2.
value-2	Specifies the number of blocks that the disk file is to accommodate.

Note: When restoring a file and there already exists a file on the fixed disk with the same name but with a different creation date and different number of blocks or records allocated, then the BLOCKS or RECORDS parameter should be used in the RESTORE procedure.

RESTORE Examples

To restore all files previously saved by a SAVE procedure, you would enter:

```
RESTORE
```

To restore a file named JOE that was saved or organized on a diskette, you would enter:

```
RESTORE JOE,,RECORDS,300
```

In the preceding example, RECORDS requests that the restored file be large enough to contain 300 records.

SAVE PROCEDURE

The SAVE procedure causes (1) a single disk file or all disk files to be copied to diskette(s) or (2) a single disk file to be added to a file saved previously on diskette(s). Sequential, indexed, and direct disk files can be copied to diskette(s) by SAVE, and can reside on diskette(s) as single volume or multivolume files. Sequential, indexed, and direct disk files can also be added to files saved previously and can reside as single volume or multivolume files. Messages to insert a diskette are given to the operator whenever a SAVE request causes a multivolume diskette file to be created or extended (added to).

This procedure evokes the $COPY utility (see index entry: *$COPY utility program*).

Note: If, after saving a file by copying it to diskette(s), you delete the original file from the disk, the file on the diskette(s) becomes the master copy of the file.

SAVE Command Statement Format

Use	Format
Save all disk files on diskette	SAVE [ALL], [retention-days / 1], [filename-1 / #SAVE] ,vol-id
Save one disk file on diskette, or add a disk file to a file saved previously	SAVE filename-2, [retention-days / 1 / ADD] , [mmddyy / ddmmyy / yymmdd] ,vol-id

SAVE Parameters

ALL	Requests that all data files on the disk be copied to diskette. The diskette should not contain any active files.
filename-2	Name of one file on the disk to be saved. The diskette file will have the same name.
retention-days 1	Number of days (0 to 999) the diskette file is to be retained. Default is 1. *Note:* A retention value of 999 makes a diskette file a permanent file.
ADD	Single disk file is to be added to a file previously saved on diskette.
filename-1 #SAVE	Name associated with the entire set of saved files. #SAVE is the default value.
mmddyy ddmmyy yymmdd	Creation date of the disk file. If not specified, the last file created with the name given in filename-2 is saved.
vol-id	Volume label of diskette. One to six alphameric characters.

RESTORE Command Statement Format

Use	Format
Restore all previously saved files.	RESTORE [ALL], [filename-1 / #SAVE] [,mmddyy / ,ddmmyy / ,yymmdd]
Restore a previously saved single file.	RESTORE filename-2, [mmddyy / ddmmyy / yymmdd] [,RECORDS, value-1 / ,BLOCKS, value-2]

RESTORE Parameters

ALL	All data files previously saved are to be restored to the disk.
filename-1 #SAVE	Name associated with the entire set of files previously saved on the diskette by the SAVE (SAVE ALL) procedure. #SAVE is the default.
filename-2	Name of the single diskette file that is to be restored to the disk.
mmddyy ddmmyy yymmdd	Creation date of the diskette file.
RECORDS	Requests that the disk file be made large enough to contain the number of records indicated by value-1.
value-1	Specifies the number of records that the disk file is to accommodate.
BLOCKS	Requests that the disk file be made large enough to contain the number of blocks indicated by value-2.
value-2	Specifies the number of blocks that the disk file is to accommodate.

Note: When restoring a file and there already exists a file on the fixed disk with the same name but with a different creation date and different number of blocks or records allocated, then the BLOCKS or RECORDS parameter should be used in the RESTORE procedure.

RESTORE Examples

To restore all files previously saved by a SAVE procedure, you would enter:

```
RESTORE
```

To restore a file named JOE that was saved or organized on a diskette, you would enter:

```
RESTORE JOE,,RECORDS,300
```

In the preceding example, RECORDS requests that the restored file be large enough to contain 300 records.

SAVE PROCEDURE

The SAVE procedure causes (1) a single disk file or all disk files to be copied to diskette(s) or (2) a single disk file to be added to a file saved previously on diskette(s). Sequential, indexed, and direct disk files can be copied to diskette(s) by SAVE, and can reside on diskette(s) as single volume or multivolume files. Sequential, indexed, and direct disk files can also be added to files saved previously and can reside as single volume or multivolume files. Messages to insert a diskette are given to the operator whenever a SAVE request causes a multivolume diskette file to be created or extended (added to).

This procedure evokes the $COPY utility (see index entry: *$COPY utility program*).

Note: If, after saving a file by copying it to diskette(s), you delete the original file from the disk, the file on the diskette(s) becomes the master copy of the file.

SAVE Command Statement Format

Use	Format
Save all disk files on diskette	SAVE [ALL], [retention-days / 1], [filename-1 / #SAVE],vol-id
Save one disk file on diskette, or add a disk file to a file saved previously	SAVE filename-2, [retention-days / 1 / ADD], [mmddyy / ddmmyy / yymmdd],vol-id

SAVE Parameters

ALL	Requests that all data files on the disk be copied to diskette. The diskette should not contain any active files.
filename-2	Name of one file on the disk to be saved. The diskette file will have the same name.
retention-days 1	Number of days (0 to 999) the diskette file is to be retained. Default is 1. *Note:* A retention value of 999 makes a diskette file a permanent file.
ADD	Single disk file is to be added to a file previously saved on diskette.
filename-1 #SAVE	Name associated with the entire set of saved files. #SAVE is the default value.
mmddyy ddmmyy yymmdd	Creation date of the disk file. If not specified, the last file created with the name given in filename-2 is saved.
vol-id	Volume label of diskette. One to six alphameric characters.

SAVE Examples

To save all files for a period of seven days on a diskette labeled 345678, you could enter:

```
SAVE ALL,7,#SAVE,345678
     or
SAVE ,7,,345678
```

To save a file named JOE (created on November 12, 1974) and to add this file to an existing diskette file named JOE (with a volume identification of 654321), you could enter:

```
SAVE JOE,ADD,741112,654321
```

SET PROCEDURE

The SET procedure establishes the following system environment items:

- Number of lines printed per page
- Print belt image
- System date format
- System date

The item(s) specified is placed in the library in the *system configuration record,* which defines system characteristics, and remains unchanged until a subsequent SET procedure is executed.

This procedure evokes the $SETCF utility (see index entry: *$SETCF utility program*).

SET Command Statement Format

SET [value] , [source-name] , [MDY / DMY / YMD] [,mmddyy / ,ddmmyy / ,yymmdd]

Note: Though each individual parameter is optional, at least one parameter must be specified.

SET Parameters

value	The number of lines that are to be printed per page. The maximum number of lines that can be specified is 84, minimum value is 1. *Note:* See index entry: *// FORMS statement* for the way the value specified determines the actual number of lines printed per page.
source-name	Name of the library source member containing the print belt image to be used by the system. The contents of the source member is described in the IMAGE statement (see index entry: *IMAGE statement*). *Note:* BELT48, BELT48HN (FORTRAN), BELT64, and BELT96 are library source members. The source-name parameter is either BELT48, BELT64, BELT96, or BELT48HN when specifying the print belt image to be used by the system.
MDY	Specifies system date format to be month-day-year.
DMY	Specifies system date format to be day-month-year.
YMD	Specifies system date format to be year-month-day.
mmddyy	Specifies the system date in month-day-year format.
ddmmyy	Specifies the system date in day-month-year format.
yymmdd	Specifies the system date in year-month-day format. *Note:* Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.

SPECIFY PROCEDURE

The SPECIFY procedure alters the following SDLC (synchronous data link control) items in the system configuration record.

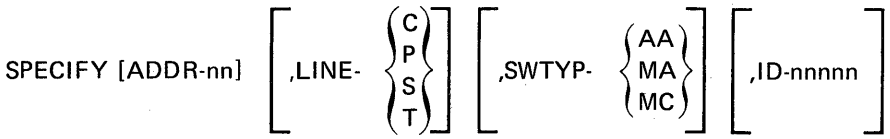
Item	Parameter
SDLC station address	ADDR
Line type	LINE
Switch type	SWTYP
Identification data	ID

Additional SDLC items that can be altered are included in the ALTERSDL procedure. (See index entry: *ALTERSDL procedure.*) To identify the current values in these parameters, use the STATUS procedure. (See index entry: *STATUS procedure.*)

The SPECIFY procedure evokes the $SETCF utility (see index entry: *$SETCF utility program*).

Note: The SPECIFY procedure is intended only for data communications programming that use SDLC. For background information on synchronous data link control, see *IBM Synchronous Data Link Control General Information,* GA27-3093. For information on data communications programming, see *IBM System/32 Data Communications Reference Manual,* GC21-7691.

SPECIFY Command Statement Format



Note: Though each individual parameter is optional, at least one must be specified.

SPECIFY Parameters

ADDR-nn	A two-character hexadecimal SDLC address.
LINE-C	CDSTL (connect data set to line) switched line (World Trade only)
P	Point-to-point nonswitched line.
S	Point-to-point switched line.
T	Tributary station on multipoint line.
SWTYP-AA	If switched line (LINE-C or LINE-S) is specified and the modem is in autoanswer mode, then the System/32 automatically answers the call.
MA	If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually answers the call.
MC	If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually initiates the call.
ID-nnnnn	A five-character hexadecimal number used as an exchange of identification between the host system and the System/32 SDLC station. Valid characters for this parameter must be from 0-9 and A-F. The characters specified are converted to hexadecimal characters by the system. If the ID parameter is not specified the default is 00000.

Notes:

1. If LINE-C or LINE-S is specified, the SWTYP parameter must be specified.
2. If the SWTYP parameter is specified, then LINE-C or LINE-S must be specified. However, if the line type was set previously to a switched line (LINE-C or LINE-S), then the line type does not have to be respecified.
3. If the SWTYP parameter (MA or MC) is specified on a switched line, a message is displayed that indicates a manual answer or a manual call is required. If the SWTYP parameter (AA) is specified on a switched line, no message is displayed.
4. If LINE-P or LINE-T is specified, then the switch type (SWTYP) automatically defaults to 0 (zero).
5. The line type defaults to a point-to-point nonswitched line (LINE-P) if the standby line (SLINE) is specified in the ALTERSDL procedure as SLINE-N.
6. The line type defaults to a point-to-point switched line (LINE-S) if the standby line (SLINE) is specified in the ALTERSDL procedure as SLINE-Y.

SYSLIST Command Statement Format

```
SYSLIST [PRINTER]
        [CRT    ]
        [OFF    ]
```

SYSLIST Parameters

PRINTER	Selects the printer for system list output
CRT	Selects the display screen for system list output
	Note: If CRT is specified, the ROLL↑ without the SHIFT key (roll up) must be pressed after each system list output record is displayed to advance to the next record.
OFF	Suppresses system list output

TOLIBR PROCEDURE

The TOLIBR procedure copies into the library either a disk or diskette file containing one or more library members. Any number of library members can be contained in a data file to be copied into the library by TOLIBR.

All sector mode files to be copied by TOLIBR must have been created either by the $MAINT utility or by the FROMLIBR procedure (see index entries: *$MAINT utility program* and *FROMLIBR procedure*).

Each library member in a record mode file that is to be copied by TOLIBR must begin with a COPY statement and end with a CEND statement. The format of the COPY statement, where name is the member and P or S indicates procedure member or source member, is:

```
// COPY NAME-name,LIBRARY-{P}
                       {S}
```

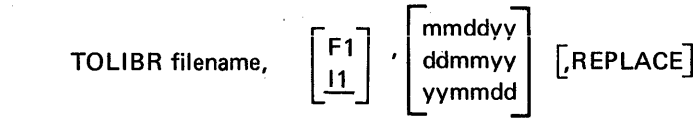
The format of the CEND record is: // CEND. COPY and CEND statements are automatically inserted in members created by $MAINT. You must insert them in members that were not created by $MAINT.

If a file to be copied by TOLIBR is a record mode diskette file in 128-bytes per sector basic data exchange format (see Appendix C), the TRANSFER procedure (see index entry: *TRANSFER procedure*) must be used to copy the file to disk before TOLIBR can copy the file to the library.

Note: In record mode TOLIBR can copy only records from 40 through 120 bytes in length.

The TOLIBR procedure evokes the $MAINT utility.

TOLIBR Command Statement Format



TOLIBR Parameters

filename	Name of the file containing the member(s) to be copied in the library.
F1	The file is on the disk.
I1	The file is on a diskette.
mmddyy ddmmyy yymmdd	Specifies the creation date of the file containing the member(s) to be copied. If date is not specified, the filename with the most recent date is copied to the library.
REPLACE	Replace the library member specified, if one exists. If REPLACE is not specified, members are placed in the library until a duplicate is found, at which time the system displays a message telling the operator that a duplicate exists. In response to the message, the operator can either cancel the job or continue processing. If the job is continued, the new member replaces the existing member in the library. If other duplicates are found during the job, then existing members are automatically replaced and no messages are displayed regarding the duplicate members. If REPLACE is specified, new members replace existing duplicate members in the library, and no messages regarding them are displayed.

TRANSFER PROCEDURE

The TRANSFER procedure moves files between the disk and diskettes that have data in the 128-bytes per sector basic data exchange format. (See Appendix C for information on the 128-bytes per sector basic data exchange format.) TRANSFER can:

- Add a diskette file that is in the 128-bytes per sector basic data exchange format to an existing sequential disk file
- Convert a basic data exchange diskette file to a disk sequential or indexed file
- Convert a disk file to a basic data exchange diskette file (Basic data exchange files are sequential files.)

Note: Because TRANSFER only moves files between the disk and basic data exchange formatted diskettes, TRANSFER cannot be used to move files between the disk and diskettes that have data recorded in 512-byte sectors (extended format). If the diskette format is not known, you can use the CATALOG procedure to list the diskette VTOC. This listing shows whether the format is 128- or 512-byte sectors.

When a basic data exchange diskette file is added to an existing disk sequential file, the record length of the disk file is used for all records added to the file. When a basic data exchange diskette file is converted to a disk sequential or indexed file, records are placed in the disk file sequentially, using the record length of the diskette file.

A disk file to be converted by TRANSFER to a basic data exchange diskette—always sequential—can be a sequential, indexed, or direct file. If the record length of the disk file is greater than 128 bytes all records are truncated to 128.

For an example of converting a source member or a procedure member to a diskette file in 128-bytes per sector basic data exchange format, see index entry: *source member to basic data exchange diskette file* and *procedure member to basic data exchange diskette file.*

The TRANSFER procedure evokes the $BICR utility (see index entry: *$BICR utility program*).

TRANSFER Command Statement Format

Use	Format
Transfer file from diskette to an existing disk file	TRANSFER filename-1, [I1] , [mmddyy / ddmmyy / yymmdd] ,ADD, [filename-2 / filename-1] [,date]
Transfer a file from diskette to a new disk file	TRANSFER filename-1, [I1] , [mmddyy / ddmmyy / yymmdd] , [NOADD], [value-1,value-2] [,RECORDS,value-3 / ,BLOCKS,value-4]
Transfer a file from disk to diskette	TRANSFER filename-1,F1, [mmddyy / ddmmyy / yymmdd] ,vol-id [,retention-days / ,1]

TRANSFER Parameters

filename-1	Name of the file being transferred. If a new file is being created, it assumes the name specified by filename-1.
I1	A basic data exchange diskette file is being transferred to a disk sequential or indexed file.
F1	A disk file is being transferred to basic data exchange diskette file.
mmddyy ddmmyy yymmdd	Creation date of the file being transferred. If the file being transferred resides on disk and no date is specified, then the filename with the most recent date is transferred to diskette. *Note:* Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.
ADD	Records in a diskette file are added to the records in an existing disk sequential file. The first record from the diskette file is placed after the last record existing in the disk file.
filename-2	Name of the existing disk file to which a basic data exchange diskette file is to be added. Filename-2 is valid only if ADD is specified. If filename-2 is omitted, defaults to filename-1.
date	Creation date of an existing disk file. Date is valid only if ADD is specified. The date must be given in one of the following formats: mmddyy, ddmmyy, or yymmdd.
NOADD	The basic data exchange diskette file being transferred will become a new disk file with filename-1 as the filename. NOADD is assumed whenever a file is transferred from diskette to disk.
value-1	Key length for a disk indexed file that is being created. Value-1 can be 1 through 29. It must be specified with value-2, and the sum of value-1 and value-2 must not exceed the record length + 1.
value-2	The start position of the record keys for an indexed disk file that is being created. Value-2 can be 1 through 128. It must be specified with value-1, and the sum of value-2 and value-1 must not exceed the record length + 1.
RECORDS, value-3	Specifies that the disk file being created be large enough to contain the number of records specified by value-3. *Note:* Either RECORDS, value-3 or BLOCKS,value-4 (see following) is required if (1) a multivolume file is being transferred, or (2) the created disk file is to be larger than the file being transferred.

TRANSFER PROCEDURE

The TRANSFER procedure moves files between the disk and diskettes that have data in the 128-bytes per sector basic data exchange format. (See Appendix C for information on the 128-bytes per sector basic data exchange format.) TRANSFER can:

- Add a diskette file that is in the 128-bytes per sector basic data exchange format to an existing sequential disk file
- Convert a basic data exchange diskette file to a disk sequential or indexed file
- Convert a disk file to a basic data exchange diskette file (Basic data exchange files are sequential files.)

Note: Because TRANSFER only moves files between the disk and basic data exchange formatted diskettes, TRANSFER cannot be used to move files between the disk and diskettes that have data recorded in 512-byte sectors (extended format). If the diskette format is not known, you can use the CATALOG procedure to list the diskette VTOC. This listing shows whether the format is 128- or 512-byte sectors.

When a basic data exchange diskette file is added to an existing disk sequential file, the record length of the disk file is used for all records added to the file. When a basic data exchange diskette file is converted to a disk sequential or indexed file, records are placed in the disk file sequentially, using the record length of the diskette file.

A disk file to be converted by TRANSFER to a basic data exchange diskette—always sequential—can be a sequential, indexed, or direct file. If the record length of the disk file is greater than 128 bytes all records are truncated to 128.

For an example of converting a source member or a procedure member to a diskette file in 128-bytes per sector basic data exchange format, see index entry: *source member to basic data exchange diskette file* and *procedure member to basic data exchange diskette file.*

The TRANSFER procedure evokes the $BICR utility (see index entry: *$BICR utility program*).

TRANSFER Command Statement Format

Use	Format
Transfer file from diskette to an existing disk file	TRANSFER filename-1, [I1] , [mmddyy / ddmmyy / yymmdd] ,ADD, [filename-2 / filename-1] [,date]
Transfer a file from diskette to a new disk file	TRANSFER filename-1, [I1] , [mmddyy / ddmmyy / yymmdd] ,[NOADD], [value-1,value-2 / ,] [,RECORDS,value-3 / ,BLOCKS,value-4]
Transfer a file from disk to diskette	TRANSFER filename-1,F1, [mmddyy / ddmmyy / yymmdd] ,vol-id [,retention-days / ,1]

TRANSFER Parameters

filename-1	Name of the file being transferred. If a new file is being created, it assumes the name specified by filename-1.
I1	A basic data exchange diskette file is being transferred to a disk sequential or indexed file.
F1	A disk file is being transferred to basic data exchange diskette file.
mmddyy ddmmyy yymmdd	Creation date of the file being transferred. If the file being transferred resides on disk and no date is specified, then the filename with the most recent date is transferred to diskette. *Note:* Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.
ADD	Records in a diskette file are added to the records in an existing disk sequential file. The first record from the diskette file is placed after the last record existing in the disk file.
filename-2	Name of the existing disk file to which a basic data exchange diskette file is to be added. Filename-2 is valid only if ADD is specified. If filename-2 is omitted, defaults to filename-1.
date	Creation date of an existing disk file. Date is valid only if ADD is specified. The date must be given in one of the following formats: mmddyy, ddmmyy, or yymmdd.
NOADD	The basic data exchange diskette file being transferred will become a new disk file with filename-1 as the filename. NOADD is assumed whenever a file is transferred from diskette to disk.
value-1	Key length for a disk indexed file that is being created. Value-1 can be 1 through 29. It must be specified with value-2, and the sum of value-1 and value-2 must not exceed the record length + 1.
value-2	The start position of the record keys for an indexed disk file that is being created. Value-2 can be 1 through 128. It must be specified with value-1, and the sum of value-2 and value-1 must not exceed the record length + 1.
RECORDS, value-3	Specifies that the disk file being created be large enough to contain the number of records specified by value-3. *Note:* Either RECORDS, value-3 or BLOCKS,value-4 (see following) is required if (1) a multivolume file is being transferred, or (2) the created disk file is to be larger than the file being transferred.

BLOCKS,
value-4

Specifies that the disk file being created be large enough to contain the number of blocks specified by value-4.

Note: Either BLOCKS,value-4 or RECORDS,value-3 (see preceding) is required if (1) a multivolume file is being transferred, or (2) the created disk file is to be larger than the file being transferred.

vol-id

Volume identification for the created basic data exchange diskette file. One to six alphameric characters.

retention-days
1

Number of days (0 to 999) the created basic data exchange diskette file is to be retained. Default is 1.

Note: A retention value of 999 makes a diskette file a permanent file.

TRANSFER Examples

In order to add a diskette basic data exchange file named JOE to an existing disk file named JOE, you could enter:

```
TRANSFER JOE,,,ADD
```

In order to create a disk sequential file named JIM from diskette basic data exchange file named JIM, you could enter:

```
TRANSFER JIM
```

In order to create a diskette basic data exchange file named JON on a diskette with vol-id of 888777 from a disk file named JON, you could enter:

```
TRANSFER JON,F1,,888777,30
```

Part 3

Using OCL Statements and Procedures

DISK FILE

Creating a disk file requires that:

- Disk space be available to hold the file
- The file be described to the SCP

Obtaining Space for a File

The CATALOG procedure (see index entry: *CATALOG procedure*) can be used to determine how much space is available on the disk and where available space is. Space to be allocated to a file must be contained in a single continuous area on the disk. If enough space is available for a file but is not contained in a single continuous area (for example, part of the available space is at one location on the disk and the rest of the space is at another location), you can use the COMPRESS procedure (see index entry: *COMPRESS procedure*) to collect all available space into one area at the high end of the disk. The $FREE (disk reorganization utility) program can also be used to move all data files to the high end of the disk, thus, collecting all available space into one area at the low end of the disk (between the library and the data files).

If the space required by a file is not available on the disk, you can do one of the following:

- Use the CATALOG procedure to see which files are currently on the disk and use the DELETE procedure (see index entry: *DELETE procedure*) to delete any unneeded files, thereby making disk space available for new files.
- Use the SAVE procedure (see index entry: *SAVE procedure*) to copy from the disk to diskette(s) one or more files that are not needed for the next job. Then, to make space available for new files, use the DELETE procedure to delete the original files. When they are needed, you can return the copied files from diskette(s) to the disk by using the RESTORE procedure (see index entry: *RESTORE procedure*).

Note: After you delete the original files from the disk, the diskette(s) contain the master copies. You can use the COPYI1 procedure (see index entry: *COPYI1 procedure*) to create a backup copy of the files you moved to diskette(s).

Describing a File

Use a FILE statement to describe a file to the SCP (see index entry: *// FILE statement*). The NAME parameter of the FILE statement must identify the file to the program creating the file. The LABEL parameter assigns a name for user identification of the file on the disk, regardless of the name a program uses to refer to the file. If the LABEL parameter is omitted from a FILE statement, the name specified by the NAME parameter identifies the file on the disk. Assign names that are related to file contents or to application programs using the files, to make the files easy to identify by programmers and operators.

Either the RECORDS or BLOCKS parameter must be used to define the size of a file, but both parameters cannot be specified for one file. If RECORDS is specified the system calculates the number of blocks required to contain the file (see Appendix A). If BLOCKS is used, the system reserves the number of blocks specified. For an indexed file the number of blocks specified is apportioned between index areas and data areas.

Note: If RECORDS is specified, the number of records actually allocated may be larger than the number requested. The system allocates disk space in blocks and always rounds up to the next whole block if part of a block is required.

The LOCATION parameter specifies the block location where the file will begin. If LOCATION is not used, the system places the file as close to the library as possible.

The RETAIN parameter classifies a file according to its retention status. Permanent files (RETAIN-P) remain on the disk until you delete them by using the DELETE procedure (or $DELET utility program—see index entry: *$DELET utility program*). A classification of RETAIN-P protects a file from being deleted accidentally. Temporary files (RETAIN-T) are usually used more than once. You can free the space used by a temporary file at any time by changing its classification to RETAIN-S, which identifies the file as a scratch file. Scratch files do not exist after the job in which they are created ends.

Three disk work files called $WORK, $WORK2, and $SOURCE are created automatically by the System/32 SCP for programs that require this work space. These files are scratch files (with a file size of 24 blocks each), used by source programs to generate an object program. If you need to change the file size, or if you want to create the files yourself, you can enter three FILE statements, one for $WORK, one for $WORK2, and one for $SOURCE, with the RECORDS or BLOCKS parameter to define the file size. You can determine if your program needs space allocated for the disk work files by looking at the library directory entry. A field (ATTRIBUTES) contains two bytes of attributes; the first byte has bit 4 on (set to 1) if a program requires that $WORK and $SOURCE be allocated; the second byte has bit 3 on (set to 1) if a program requires that $WORK2 be allocated. (See index entry: *library directory entry* for a description of the information contained in library directory entries.)

The disk VTOC can contain up to 200 permanent or temporary files at any one time (199 user files plus the system file #LIBRARY). You can use the CATALOG procedure to determine the number of permanent and temporary files currently on the disk. (For more information, see index entry: *CATALOG procedure.)*

DISKETTE FILE

You must use a FILE statement to describe each diskette file you want created. The FILE statement for diskette files is described in detail under index entry: *// FILE statement.* Diskette files are created by IBM system utility programs, described in Part 4, or by offline multivolume file processing. Diskette files created by system utility programs cannot be processed as offline multivolume files, and offline multivolume files cannot be processed by the system utility programs except by $DUPRD. The following paragraphs concern using the utility programs to create and process diskette files. For a discussion of offline multivolume file processing, see *Offline Multivolume File* which follows.

DISK FILE

Creating a disk file requires that:

- Disk space be available to hold the file
- The file be described to the SCP

Obtaining Space for a File

The CATALOG procedure (see index entry: *CATALOG procedure*) can be used to determine how much space is available on the disk and where available space is. Space to be allocated to a file must be contained in a single continuous area on the disk. If enough space is available for a file but is not contained in a single continuous area (for example, part of the available space is at one location on the disk and the rest of the space is at another location), you can use the COMPRESS procedure (see index entry: *COMPRESS procedure*) to collect all available space into one area at the high end of the disk. The $FREE (disk reorganization utility) program can also be used to move all data files to the high end of the disk, thus, collecting all available space into one area at the low end of the disk (between the library and the data files).

If the space required by a file is not available on the disk, you can do one of the following:

- Use the CATALOG procedure to see which files are currently on the disk and use the DELETE procedure (see index entry: *DELETE procedure*) to delete any unneeded files, thereby making disk space available for new files.
- Use the SAVE procedure (see index entry: *SAVE procedure*) to copy from the disk to diskette(s) one or more files that are not needed for the next job. Then, to make space available for new files, use the DELETE procedure to delete the original files. When they are needed, you can return the copied files from diskette(s) to the disk by using the RESTORE procedure (see index entry: *RESTORE procedure*).

Note: After you delete the original files from the disk, the diskette(s) contain the master copies. You can use the COPYI1 procedure (see index entry: *COPYI1 procedure*) to create a backup copy of the files you moved to diskette(s).

Describing a File

Use a FILE statement to describe a file to the SCP (see index entry: *// FILE statement*). The NAME parameter of the FILE statement must identify the file to the program creating the file. The LABEL parameter assigns a name for user identification of the file on the disk, regardless of the name a program uses to refer to the file. If the LABEL parameter is omitted from a FILE statement, the name specified by the NAME parameter identifies the file on the disk. Assign names that are related to file contents or to application programs using the files, to make the files easy to identify by programmers and operators.

Either the RECORDS or BLOCKS parameter must be used to define the size of a file, but both parameters cannot be specified for one file. If RECORDS is specified the system calculates the number of blocks required to contain the file (see Appendix A). If BLOCKS is used, the system reserves the number of blocks specified. For an indexed file the number of blocks specified is apportioned between index areas and data areas.

Note: If RECORDS is specified, the number of records actually allocated may be larger than the number requested. The system allocates disk space in blocks and always rounds up to the next whole block if part of a block is required.

The LOCATION parameter specifies the block location where the file will begin. If LOCATION is not used, the system places the file as close to the library as possible.

The RETAIN parameter classifies a file according to its retention status. Permanent files (RETAIN-P) remain on the disk until you delete them by using the DELETE procedure (or $DELET utility program—see index entry: *$DELET utility program*). A classification of RETAIN-P protects a file from being deleted accidentally. Temporary files (RETAIN-T) are usually used more than once. You can free the space used by a temporary file at any time by changing its classification to RETAIN-S, which identifies the file as a scratch file. Scratch files do not exist after the job in which they are created ends.

Three disk work files called $WORK, $WORK2, and $SOURCE are created automatically by the System/32 SCP for programs that require this work space. These files are scratch files (with a file size of 24 blocks each), used by source programs to generate an object program. If you need to change the file size, or if you want to create the files yourself, you can enter three FILE statements, one for $WORK, one for $WORK2, and one for $SOURCE, with the RECORDS or BLOCKS parameter to define the file size. You can determine if your program needs space allocated for the disk work files by looking at the library directory entry. A field (ATTRIBUTES) contains two bytes of attributes; the first byte has bit 4 on (set to 1) if a program requires that $WORK and $SOURCE be allocated; the second byte has bit 3 on (set to 1) if a program requires that $WORK2 be allocated. (See index entry: *library directory entry* for a description of the information contained in library directory entries.)

The disk VTOC can contain up to 200 permanent or temporary files at any one time (199 user files plus the system file #LIBRARY). You can use the CATALOG procedure to determine the number of permanent and temporary files currently on the disk. (For more information, see index entry: *CATALOG procedure.)*

DISKETTE FILE

You must use a FILE statement to describe each diskette file you want created. The FILE statement for diskette files is described in detail under index entry: *// FILE statement.* Diskette files are created by IBM system utiiity programs, described in Part 4, or by offline multivolume file processing. Diskette files created by system utility programs cannot be processed as offline multivolume files, and offline multivolume files cannot be processed by the system utility programs except by $DUPRD. The following paragraphs concern using the utility programs to create and process diskette files. For a discussion of offline multivolume file processing, see *Offline Multivolume File* which follows.

Before a diskette can contain any files, it must be *initialized.* That is, it must be examined for bad tracks, and formatted control information required by the system must be recorded on the diskette. You can use the INIT procedure (see index entry: *INIT procedure*) to initialize diskettes.

Note: If a job will require a number of diskettes, initialize all required diskettes that have not been initialized before you begin the job. If all diskettes are initialized in advance, you will not have to interrupt or cancel the job in order to initialize a diskette when another diskette is required.

If the file you want to create is to be placed on a diskette that already contains files (but does not contain part of an offline multivolume file), use the CATALOG procedure (see index entry: *CATALOG procedure)* to determine how much space is available on the diskette. The available space is unused space following the last active file currently on the diskette. (Files added to a diskette always follow active files already on the diskette.)

If a diskette lacks space for a new file, you can do either of the following:

1. Allow the file to become a multivolume file; use the diskette to start the file. When diskette space expires, the system requests another diskette to continue the file. A description of multivolume files follows.

2. Use the COPY11 procedure to rearrange the active files and to delete the expired files, leaving space for a new file at the end of the diskette. (For more information, see index entry: *COPY11 procedure.)*

If multiple files are to be created on a single diskette, each file LABEL must be unique. Duplicate file labels on the same diskette are not permitted.

For the operator's convenience, write in the space provided on the diskette envelope the name of each file contained on the diskette. You may also want to store with the diskettes the listings created by the CATALOG procedure to help identify which files are on which diskettes. The diskette VTOC can contain up to 19 active files.

OFFLINE MULTIVOLUME FILE

Each diskette is a volume of storage. A *multivolume file* is a diskette file residing on more than one diskette, or expanded from one diskette to more than one diskette. Multivolume files can be created by the system utility programs or by the offline multivolume function of the SCP. These two kinds of files cannot be processed interchangeably. Files created and processed by the offline multivolume function are called *offline multivolume files.*

Purpose of Offline Multivolume Files

Many jobs process files that exist entirely on the disk. However, you may have a job requiring more file space than the disk currently has available. The last file to be allocated, for example, may need 200 blocks of disk space when only 95 are available. If you reduced the BLOCKS parameter specification on the FILE statement to 95, problems would occur in the job after the 95 blocks were filled. A solution would be to use the DELETE and COMPRESS procedures to free up disk space. (See index entries: *COMPRESS procedure* and *DELETE procedure.*)

Using an offline multivolume file would be another solution. It allows you to allocate the last file, even though the disk does not have enough space for the entire file.

Offline multivolume file processing uses all available disk space (up to the maximum allowed—see *Offline Multivolume Restrictions and Considerations*) as an intermediate work area for processing a file a portion at a time. Offline multivolume processing moves a file, a portion at a time, from the allocated disk extent to an output diskette, or from diskettes to the allocated disk extent, for processing.

The portion of an offline multivolume file, moving in this manner from and to the disk, is called a file segment. File segments are stored on diskettes, one segment per diskette.

Creating an Offline Multivolume File

You can evoke offline multivolume file processing by entering a FILE statement specifying the same NAME given in a FILE statement for a disk file, and I1 for UNIT. Suppose, for example, you want to allocate the file described by the following FILE statement, but 200 blocks of available space do not exist on the disk:

```
// FILE NAME-PAYMSTR,UNIT-F1,BLOCKS-200,RETAIN-T
```

If 95 blocks of disk space are available, enter the following two FILE statements to allocate and process PAYMSTR as an offline multivolume file:

```
// FILE NAME-PAYMSTR,UNIT-F1,BLOCKS-95,RETAIN-S
// FILE NAME-PAYMSTR,UNIT-I1,RETAIN-20,PACK-666666
```

As PAYMSTR is processed, records are placed in the 95-block extent on the disk. When all 95 blocks are full, the system issues a message requesting the operator to insert a diskette for output. After the diskette is inserted, the system copies the records from the disk extent to the diskette. The disk extent is then reused, with the next record being written at the beginning of the extent. When the extent is again full, the system requests another diskette. This process, writing PAYMSTR in file segments of 95 blocks, continues until the job ends. The system writes the remaining records (whether or not it fills the 95-block extent) on a diskette at the end of the job.

Note: The offline multivolume function saves the file during job processing. This is different from the SAVE procedure which, being issued after job processing, copies the file from the disk onto a diskette, thus creating a backup file.

After the job ends, PAYMSTR resides only on diskettes. The 95-block disk extent contains a copy only of those records in the last file segment. If you want a backup copy of an offline multivolume file, you can use the COPYI1 procedure (see index entry: *COPYI1 procedure*) to copy, one at a time, each of the diskettes composing the file.

Reading an Offline Multivolume File

In the example below, the offline multivolume file PAYMSTR will be read, a segment at a time, from diskettes into a disk extent named PAYMSTR:

```
// FILE NAME-PAYMSTR,UNIT-F1,BLOCKS-95,RETAIN-S
// FILE NAME-PAYMSTR,UNIT-I1,PACK-666666
```

PAYMSTR file segments were defined previously as being 95 blocks long because PAYMSTR was created with 95-block segments (see the FILE statement example under *Creating an Offline Multivolume File*).

Offline Multivolume File Restrictions and Considerations

Restrictions

- Use the same NAME on both the disk and the diskette FILE statement when you are creating an offline multivolume file. The LABEL parameters can be different. For example:

  ```
  // FILE NAME-PAYMSTR,UNIT-F1,LABEL-TEMP,BLOCKS-95
  // FILE NAME-PAYMSTR,UNIT-I1,LABEL-PAY01,PACK-666666
  ```

  The resulting offline multivolume file will be named PAY01.

- Use BLOCKS, not RECORDS, to specify segment size on the disk FILE statement for an offline multivolume file. Any block size, from one block to the maximum 95 blocks or 118 blocks allowed, can be used if space is available.

- BLOCKS - 95 blocks (basic data exchange format diskette) or 118 blocks (extended format diskette) are the maximum allocations for offline multivolume disk file segments. For offline processing, the block value for a given format equals the data area of one diskette. To use diskettes efficiently, the number of blocks allocated should be as close to 95 or 118 as possible, but can never exceed the format maximum.

  Note: Though diskettes can be initialized in either of the basic data exchange format or the extended formats, you cannot create an offline multivolume file using these formats interchangeably. Either format can be used to create an offline multivolume file, but all diskettes for a file must have the same format.

- To process an offline multivolume file after it is created, you must allocate a disk extent at least equal in size to the extent defined when the file was created. The disk extent size however, must not exceed the size of the maximum allocations for offline multivolume disk file segments (95 or 118 blocks). If you do not remember or do not have a record of the number of blocks allocated originally, you can run the CATALOG ALL, I1 procedure (see index entry: *CATALOG procedure*) using the offline multivolume diskette. The disk extent is indicated in the column titled NUMBER OF BLOCKS IN OFFLINE MV FILE found on the CATALOG procedure printout.

- A multivolume file created by a system utility cannot be processed as an offline multivolume file. Utilities that create diskette files cannot process offline multivolume files.

Restrictions (continued)

- To maintain offline multivolume file support, the INQUIRY/OFFLINE option must be selected whenever using the RELOAD procedure (see index entry: *RELOAD procedure*).

- Offline multivolume files cannot be used in the same program with the following:
  - Shared I/O data management
  - BSC (binary synchronous communications support)
  - SDLC (synchronous data link control) support
  - Data recorder attachment support
  - Word processing support
  - 1255 Magnetic Character Reader attachment support

- The same file cannot be processed twice during one job as an offline multivolume file, but more than one file can be processed as an offline multivolume file during one job.

- Offline multivolume files cannot be processed while running an inquiry program. (For more information on inquiry programs see index entry: *$LOAD utility program*).

- Offline multivolume files must be sequential files. They can be processed by consecutive output, input, update, and add access methods. They cannot be processed by indexed or direct access methods.

- Offline multivolume files must be written to diskettes containing no active files. Therefore, be sure the diskettes you use (for output or add offline mutlivolume files) have been initialized before you begin the job. You can use the INIT procedure (see index entry: *INIT procedure*) to initialize the diskettes.

- Active files cannot be written to a diskette containing part of an offline multivolume file.

- When adding file records to an offline multivolume file, you must add the new file records to the end of the file. Suppose, for example, you have an offline multivolume file: diskettes A, B, and C. Diskette C is the end of the file.

  For an add operation, the system displays the message: CONTINUE WHEN PROPER DISKETTE INSERTED. After diskette C is inserted, the system transfers the records to the disk extent; processes the file records, and adds new file records to the file extent until it is full. The system displays the same message again: CONTINUE WHEN PROPER DISKETTE INSERTED for the output operation. After you insert the diskette, the system writes the disk file extent back onto a diskette.

MESSAGES

Message text can be retrieved from a message load member in the library and displayed on the display screen or printed. There are two levels of messages: level 1 and level 2. Level 1 messages are a maximum of 40 characters long and level 2 messages are a maximum of 200 characters long. A level 2 message is an extension of a level 1 message that further describes the error. A level 2 message can be displayed only after the level 1 message of the same MIC (message identification code) is issued.

User messages are created and used by doing the following:

1. Creating a message source member.
2. Creating a message load member.
3. Specifying the message load member.
4. Retrieving the messages.

Creating a Message Source Member

The first entry in the source member must be the message control statement, which specifies the name of the message load member to be created and whether it is a first or second level message load member. The message text statement consists of the MIC and the text (actual message). For a detailed description of the message control statement and the message text statement, see index entry: *message source member.* Once the message source member statements have been defined, the message source member is put into the library by either using the $MAINT utility program or the Source Entry Utility Program Product.

The following is an example of a message source member called USERM1:

```
USERMSG,1                                          Message Control Statement
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.  }   MIC and Message
1235 THE INPUT IS IN A DISKETTE FILE.          }   Text Statements
1236 INSERT DISKETTE NUMBER 123456.            }
```

Using SEU

To put a message source member called USERM1 into the library using SEU, key SEU USERM1,S and the message source member statements you have defined. The entries for the message source member USERM1 would be:

```
SEU USERM1,S                                          Source Member Name
USERMSG,1
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.   }     Message Source Member
1235 THE INPUT IS IN A DISKETTE FILE.           }
1236 INSERT DISKETTE NUMBER 123456.             }
```

For further information on using SEU, see *IBM System/32 Utilities Program Product Reference Manual Source Entry Utility,* SC21-7605.

Using the $MAINT Utility

The following OCL is needed to put the message source member, USERM1, into the library using $MAINT:

```
// LOAD $MAINT
// RUN                                  Source Member Name
// COPY FROM-READER,LIBRARY-S,NAME-USERM1,TO-F1,RETAIN-P,RECL-45
USERMSG,1
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.   }     Message Source Member
1235 THE INPUT IS IN A DISKETTE FILE.           }
1236 INSERT DISKETTE NUMBER 123456.             }
// CEND
// END
```

CREATING A MESSAGE LOAD MEMBER

To create a message load member named USERMSG from the above source member (USERM1), use the CREATE procedure by entering:

```
CREATE USERM1
```

Once this is done, your messages 1234, 1235, and 1236 are ready to be used by the message OCL statement or your program. For more information on the CREATE procedure, see index entry: *CREATE procedure.*

MESSAGES

Message text can be retrieved from a message load member in the library and displayed on the display screen or printed. There are two levels of messages: level 1 and level 2. Level 1 messages are a maximum of 40 characters long and level 2 messages are a maximum of 200 characters long. A level 2 message is an extension of a level 1 message that further describes the error. A level 2 message can be displayed only after the level 1 message of the same MIC (message identification code) is issued.

User messages are created and used by doing the following:

1. Creating a message source member.
2. Creating a message load member.
3. Specifying the message load member.
4. Retrieving the messages.

Creating a Message Source Member

The first entry in the source member must be the message control statement, which specifies the name of the message load member to be created and whether it is a first or second level message load member. The message text statement consists of the MIC and the text (actual message). For a detailed description of the message control statement and the message text statement, see index entry: *message source member.* Once the message source member statements have been defined, the message source member is put into the library by either using the $MAINT utility program or the Source Entry Utility Program Product.

The following is an example of a message source member called USERM1:

```
USERMSG,1                                          Message Control Statement
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.  }   MIC and Message
1235 THE INPUT IS IN A DISKETTE FILE.          }   Text Statements
1236 INSERT DISKETTE NUMBER 123456.            }
```

Using SEU

To put a message source member called USERM1 into the library using SEU, key SEU USERM1,S and the message source member statements you have defined. The entries for the message source member USERM1 would be:

```
SEU USERM1,S                                          Source Member Name
USERMSG,1
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.   }     Message Source Member
1235 THE INPUT IS IN A DISKETTE FILE.           }
1236 INSERT DISKETTE NUMBER 123456.             }
```

For further information on using SEU, see *IBM System/32 Utilities Program Product Reference Manual Source Entry Utility,* SC21-7605.

Using the $MAINT Utility

The following OCL is needed to put the message source member, USERM1, into the library using $MAINT:

```
// LOAD $MAINT
// RUN                                    Source Member Name
// COPY FROM-READER,LIBRARY-S,NAME-USERM1,TO-F1,RETAIN-P,RECL-45
USERMSG,1
1234 THIS PROCEDURE RUNS THE PAYROLL PROGRAM.   }     Message Source Member
1235 THE INPUT IS IN A DISKETTE FILE.           }
1236 INSERT DISKETTE NUMBER 123456.             }
// CEND
// END
```

CREATING A MESSAGE LOAD MEMBER

To create a message load member named USERMSG from the above source member (USERM1), use the CREATE procedure by entering:

```
CREATE USERM1
```

Once this is done, your messages 1234, 1235, and 1236 are ready to be used by the message OCL statement or your program. For more information on the CREATE procedure, see index entry: *CREATE procedure.*

SPECIFYING THE MESSAGE LOAD MEMBER

Message load members PROGRAM1 and PROGRAM2 are used by IBM program products to assign names to associated message load members. In order to retrieve the messages you have created, you must specify which message load member they are in with the MEMBER OCL statement. In our example, the message load member was a first level message load member named USERMSG. To specify this message load member, the MEMBER statement would be:

```
// MEMBER USER1-USERMSG
```

For more information on the MEMBER statement, see index entry: *// MEMBER statement.*

RETRIEVING THE MESSAGES

After the messages are placed in a message load member and the load member is specified by the MEMBER OCL statement, messages can be retrieved by using either the message OCL statement (//*) or your program.

Retrieving Messages by Using the Message OCL Statement

To retrieve the first message from the message load member USERMSG as shown in the previous example, the following message OCL statement would be used:

```
// * 1234
```

This would cause the first message (THIS PROCEDURE RUNS THE PAYROLL PROGRAM.) to appear on the display screen.

The following is an example of a procedure (named PAYROLL) that would use the messages in the previous message source member example (USERMSG):

```
// MEMBER USER1-USERMSG
// * 1234
// * 1235
// * 1236
// PAUSE
// LOAD PAYROLL1
// RUN
```

When this procedure is run, the following messages would appear on the display screen:

```
THIS PROCEDURE RUNS THE PAYROLL PROGRAM.
THE INPUT IS IN A DISKETTE FILE.
 INSERT DISKETTE NUMBER 123456.
ACTION SCP 1162 CRPS OPTIONS (0    )?_
PAUSE--WHEN READY, ENTER 0 TO CONTINUE
```

A PAUSE statement normally follows the message statement if an operator response is required. The PAUSE statement causes the SCP to suspend processing, allowing the operator time to perform the action required in the message. For more information on the PAUSE statement, see the index entry: *// PAUSE statement.*

Retrieving Messages by Using Your Program

You can retrieve some messages through your program. For information on how to do this and what messages cannot be retrieved, see the *IBM System/32 RPG II Reference Manual,* SC21-7595.

RESTRICTIONS ON RETRIEVING MESSAGES

A level 2 message can only be displayed immediately after the level 1 message if the same MIC has been issued. Since processing is not stopped when you retrieve a message using OCL, level 2 messages cannot be used. This is true even when a PAUSE statement is used. This restriction is not always true when messages are retrieved by your program. For more information, see the *IBM System/32 RPG II Reference Manual,* SC21-7595.

IBM PROGRAMS

Many IBM programs require only one command statement or two OCL statements (LOAD and RUN OCL statements).

The following two examples show the statements needed to load and run two IBM programs, one requiring a command statement and the other requiring two OCL statements.

- The CREATE command statement (see index entry: *CREATE procedure*) evokes the $MGBLD utility program:

```
CREATE MSG1234
```

- The following two OCL statements load and run the $STATS utility program (see index entry: *$STATS utility program*):

```
// LOAD $STATS
// RUN
```

OBJECT PROGRAMS USING ONE DISK FILE

To load and run an object program that uses one disk file, a FILE OCL statement is required in addition to the LOAD and RUN statements. The NAME parameter is always required in the FILE statement, and the RECORDS or BLOCKS parameter is required for a disk output file. (See index entry: *// FILE statement* for a complete description of FILE statements.)

For example, to load and run the object program PROG1, which uses the disk file NAMEADD, the following OCL statements are required:

```
// LOAD PROG1
// FILE NAME-NAMEADD
// RUN
```

OBJECT PROGRAMS USING MORE THAN ONE DISK FILE

One FILE statement is required for each file used by a program (see index entry: *// FILE statement* for a complete description of FILE statements).

Two disk files are named in the following sequence of OCL statements, an input file (INPUTF) and an output file (OUTPUTF):

```
// LOAD PROG1
// FILE NAME-INPUTF
// FILE NAME-OUTPUTF,BLOCKS-10,RETAIN-P
// RUN
```

The first FILE statement contains information needed to refer to the data in the disk file INPUTF. The second FILE statement contains information needed to create the disk output file OUTPUTF.

OBJECT PROGRAMS USING ONE DISK FILE AND EXTERNAL INDICATORS

The SWITCH OCL statement (see index entry: // *SWITCH statement*) is used to set external indicators (U1–U8 on RPG II specification sheets) on or off. External indicators are used to regulate processing.

In the following example, a program (PROG2) is being run using one existing disk file (INVMSTR), an inventory master file.

```
// LOAD PROG2
// FILE NAME-INVMSTR
// FILE NAME-NEWMSTR,BLOCKS-50
// SWITCH 1XXXXXXX
// RUN
```

In the example, the SWITCH statement specifies that the first external indicator (U1) must be turned on before the program (PROG2) creates the file (NEWMSTR). Only one external indicator is used: U1.

This section illustrates some of the uses of OCL and command statements through an example of a series of jobs.

The main program is INVUPD (inventory update). INVUPD reads the file named INVTRANS (inventory transactions), updates the file named INVMSTR (inventory master), and prints a report. If INVTRANS is not on the disk, the COPYTRAN procedure is evoked to copy the transactions from a diskette to the disk. After the INVUPD program is run, SWITCH1 is checked by an IF expression to determine whether or not the user wants the COPYINV procedure run. The COPYINV procedure copies the updated INVMSTR to diskette.

The OCL and command statements for these jobs are shown in Figure 5. The sets of statements are numbered to correspond to the explanations following.

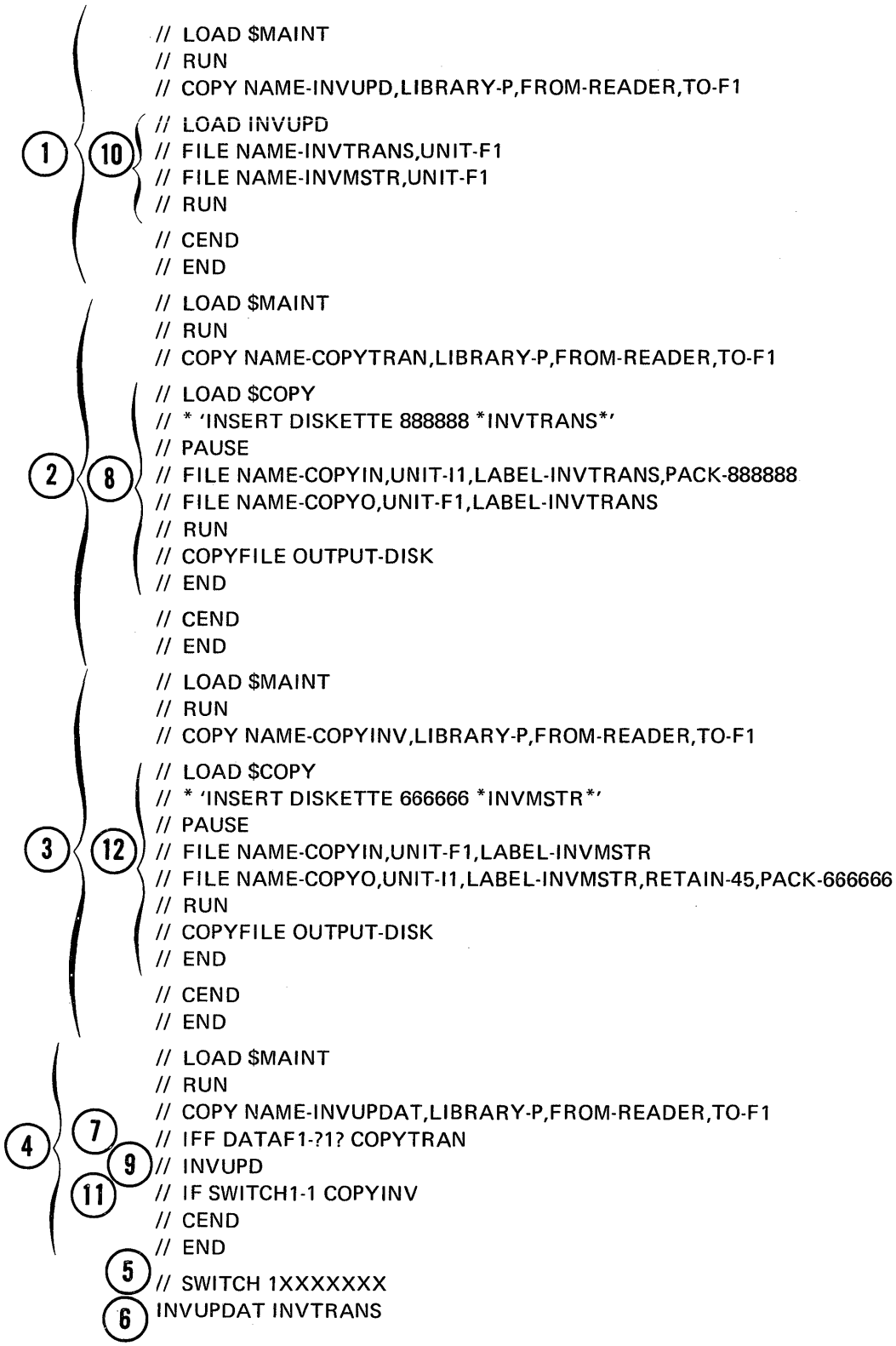


Figure 5. OCL and Command Statement Example

1. The procedure INVUPD (10) is cataloged in the library as a procedure member.

   Note: The sets of statements, 1-4, show // CEND and // END utility control statements. The // CEND utility control statement identifies the end of a source or a procedure member being put into the library. A source or a procedure member statement is preceded by a // COPY utility control statement and followed by a // CEND utility control statement. The // END utility control statement indicates the end of utility control statements for a utility program. The // END statement must be the last utility control statement entered for that utility program.

2. The procedure COPYTRAN (8) is cataloged in the library as a procedure member.

3. The procedure COPYINV (12) is cataloged in the library as a procedure member.

4. The procedure INVUPDAT (7, 9, 11) is cataloged in the library as a procedure member.

5. // SWITCH 1XXXXXXX is entered on the keyboard. This sets U1 of SWITCH to a 1 (refer to explanation 11 following) without changing any of the other 7 switches.

6. INVUPDAT INVTRANS is entered on the keyboard. The procedure INVUPDAT is evoked.

7. The first statement of the INVUPDAT procedure is the IFF (if false) statement. This statement checks to see if the file identified by the first parameter (INVTRANS) in the command statement entered on the keyboard exists on the disk. In this example, assume that there is no existing INVTRANS disk file. Therefore, the COPYTRAN procedure is evoked in order to copy the INVTRANS diskette file to disk. (If INVTRANS was already on the disk, the statement would not have been false and COPYTRAN would not have been evoked.)

8. The COPYTRAN procedure evokes the $COPY utility program. It also tells the operator to insert the diskette: 'INSERT DISKETTE 888888 *INVTRANS*'. After the operator has inserted diskette 888888 and replied to the PAUSE, the $COPY utility copies the INVTRANS file to the disk.

9. The INVUPD procedure is evoked.

10. The INVUPD procedure loads and runs the inventory update program (INVUPD).

11. After the INVUPD program has been run, SWITCH1 is checked by an IF statement in order to determine if the procedure COPYINV should be evoked. In this example, SWITCH1 was set to 1. Therefore, the IF statement is satisfied and the COPYINV procedure is evoked. (If SWITCH1 had not been 1, COPYINV would not have been evoked.)

12. The COPYINV procedure evokes the $COPY utility program. It also tells the operator to insert the diskette: 'INSERT DISKETTE 666666 *INVMSTR*'. After the operator has inserted diskette 666666 and replied to the PAUSE, the $COPY utility copies the INVMSTR to diskette.

After the last procedure (COPYINV) is run, the system returns to a ready status (awaits keyboard entry).

Once the procedures are cataloged (steps 1 through 4 in the example), the entire job can be evoked anytime by two statements (steps 5 and 6).

Part 4

System Utility Programs

IBM System/32 system control programming includes a group of utility programs that reside on the disk. These programs do a variety of jobs, from preparing the disk and diskettes for use to maintaining the system library.

WRITING UTILITY CONTROL STATEMENTS

Most of the utility programs require utility control statements. You must provide them. Utility control statements give the utilities information about the output you want and the way in which you want a utility to perform its function. The utilities read these statements from procedures and from the keyboard. Utility control statements must be the first input read by a utility if the utility requires control statements. A // END utility control statement must be the last control statement entered for a utility if control statements are used.

Every control statement is made up of an identifier and parameters. The identifier is a word that identifies the control statement. It is always the first word of the statement. Parameters are information you are supplying to the utility. Parameters are either positional or keyword.

A positional parameter, whenever it appears in a statement, must appear in the same position in relation to other parameters. For example:

```
// INCLUDE PROCEDUR FILEA,YES,NO
```

FILEA is the first parameter, YES is the second parameter, and NO is the third parameter. If you omit the second parameter (a valid positional parameter), a comma must indicate the position reserved for the omitted parameter. For example:

```
// INCLUDE PROCEDUR FILEA,,NO
```

A keyword parameter contains a keyword that distinguishes the parameter from other parameters. For example:

```
// FILE NAME-COPYIN,UNIT-F1,LABEL-PAYROLL
```

NAME-COPYIN, UNIT-F1, and LABEL-PAYROLL are keyword parameters in the preceding statement. COPYIN, F1, and PAYROLL are the values supplied by the parameters to the utility.

RULES FOR CODING UTILITY CONTROL STATEMENTS

The rules for coding utility control statements are:

1. *Statement identifier.* // in positions 1 and 2, followed by a blank, must precede the statement identifier. Do not use blanks within the identifier.

2. *Blanks.* Use one or more blanks between the identifier and the first parameter.

3. *Statement parameters.* Keyword parameters can be in any order; but positional parameters must be in the same order. Use a comma to separate one parameter from another. Use a hyphen (-) within each keyword parameter to separate the keyword from the information you supply. Do not use blanks between parameters; do not use blanks within a parameter unless the parameter contains a value enclosed by single quotation marks (for example, 'CONSTANT VALUE').

The following is an example of a utility control statement:

```
// COPYI1 NAME-JOE,PACK-123456
```

The statement identifier is COPYI1. The parameter keywords are NAME and PACK. The information supplied by the parameters is JOE and 123456.

4. *// END control statement.* This utility control statement indicates the end of utility control statements for a utility. An end control statement must be the last control statement entered for a utility if utility control statements are used. A // END control statement cannot contain other statement information such as a comment or a sequence number. Only // END is valid.

5. *Continuation.* Some utility control statements can be expressed in two or more records. A record can consist of a maximum of 120 characters, including blanks and commas, when expressing a utility control statement. A utility control statement can be continued if statement parameters are entered.

Rules for using continuation are:

- Place a comma after the last parameter in every record except the last. The comma, followed by a blank, tells the system that the statement is continued in the next record.
- Begin each new record with // in positions 1 and 2.
- Leave one or more blanks between the // and the first parameter in the record.

The following is an example of a continued utility control statement:

```
// TRANSFER ADD-NO,
// KEYLEN-5,
// KEYLOC-3
```

Utility Program Descriptions

This section describes each utility program provided with IBM System/32. The following information is given for each utility:

- The function of the utility
- The format of the related utility control statement(s)
- A description of the parameters in the related utility control statement(s)
- The sequence of the OCL and utility control statements required to evoke the utility

Examples are given for many of the utilities.

CAUTION
When a program that allows an inquiry request is interrupted, the execution of that program is suspended, permitting the execution of other programs. However, if these other programs alter the status of the system or the status of files, the effect may be abnormal termination of the interrupted program or erroneous results when the interrupted program regains control. If you are using inquiry, *do not* change any files that were being used by the interrupted (rolled-out) program. System/32 system control programming does not always check for duplicate file labels in the inquiry and interrupted programs. For example, program X is interrupted while it is processing file A. Records in file A are then deleted using inquiry. A return to program X will cause unpredictable results.

The system and disk oriented functions listed below have the potential for such abnormal termination and erroneous results when executed in an inquiry mode:

- An inquiry request cannot be used to execute the following utilities:

Utility	Function(s)
$BACK	Back up library
$LOAD	Reload library
$PACK	Compress file space
$REBLD	Rebuild VTOC
$SETCF	Reconfigure system
$BUILD	Rebuild alternate sector
$FREE	Compress file space

- An inquiry request cannot be used to run the following utilities to perform the listed functions:

Utility	Function(s)
$COPY	Restore all/save all files
$DELET	Delete all files

- An inquiry request cannot be used to run the following utilities to process active files:

Utility	Function(s)
$BICR	Transfer active file
$COPY	Save/organize active file
$DELET	Delete active file
$RENAM	Rename data files

- An inquiry request can be used to run the following utilities to perform the following functions, but a warning message will be issued when the function is requested:

Utility	Function(s)
$COPY	Display active file
$LABEL	Catalog all/active file

$BACK—BACKUP LIBRARY UTILITY PROGRAM

The $BACK utility allows the user to copy and reorganize the entire system library to a diskette file.

When the library is copied to the diskette(s), library members are shifted to remove gaps between them—unused space between members is collected at the end of the library. The output diskette(s) must not contain active files.

More than one diskette may be required to contain the system library. When this situation arises, the operator is automatically instructed to insert another diskette if it is required, after which processing resumes.

To determine the number of backup diskettes required to contain the library, see index entry: *calculating the number of backup diskettes required for the system.* To reconstruct a library on the disk that was backed up on (copied to) diskettes, you can use the RELOAD procedure (see index entry: *RELOAD procedure*) or perform an IPL from the diskette(s) containing the copy of the library. (See *IBM System/32 Operator's Guide,* GC21-7591, for a step-by-step description of how to reload the library.) The vol-id of the first (or only) diskette containing the library becomes the vol-id of the disk file during the reload operation.

$BACK is evoked by the BACKUP procedure (see index entry: *BACKUP procedure*).

$BACK Utility Control Statement Format

Utility control statements are not used.

$BACK OCL Sequence

```
// LOAD $BACK
// FILE NAME-#LIBRARY,UNIT-I1,...
// RUN
```

$BICR–BASIC DATA EXCHANGE UTILITY PROGRAM

This utility provides a means of converting a disk file to a basic data exchange file on a diskette, of converting a diskette basic data exchange file to a sequential or indexed disk file, and of adding a basic data exchange file to a sequential disk file. All diskette files that are input for $BICR must be in the 128-bytes-per-sector basic data exchange format (see Appendix C); all diskette files created by $BICR are in the basic data exchange format.

In adding a basic data exchange diskette file to an existing disk file, the records in the diskette file are truncated or padded with hex zeros (hex 00) to conform to the record length of the disk file. In creating a new disk file from a basic data exchange diskette file, the record length of the disk file is set to that of the diskette file. In creating a new basic data exchange diskette file from a disk file, the record length of the diskette file is set to that of the disk file or to 128, whichever is smaller.

$BICR processes records sequentially during file conversion. If input for $BICR is an indexed disk file, records are read sequentially by key. $BICR is evoked by the TRANSFER procedure and JOBSTR procedure (see index entries: *TRANSFER procedure* and *JOBSTR procedure*).

$BICR Utility Control Statement Formats

Use	Control Statements
To create a diskette basic data exchange file from a disk file or convert a diskette basic data exchange file to a disk sequential file	[// TRANSFER] // END
To add the data in a basic data exchange diskette file to a disk sequential file	// TRANSFER ADD-YES // END
To create an indexed file on the disk from a diskette basic data exchange file	// TRANSFER ADD-NO,KEYLEN-value,KEYLOC-value // END

$BICR Parameters

ADD-YES — Specifies that when converting a basic data exchange diskette file to a disk file, the records in the diskette file are to be added to an existing sequential disk file.

The first record in the diskette file will be placed after the last record in the disk file. If a multivolume file is being converted, records will be added to the disk file until the end of either the diskette file or the disk file is reached. However, for both multivolume diskette files and single volume diskette files, the add operation is not started unless the diskette file or file segment on the diskette currently in the diskette drive will fit into the space is available in the disk file.

If ADD-YES is not specified, ADD-NO is assumed.

ADD-NO — Indicates that when copying a diskette basic data exchange file to the disk, a new disk file is to be created.

KEYLEN-value — Defines the length of the record keys when an indexed file is to be created on the disk. Value can be from 1 through 29.

Note: KEYLEN must be specified with KEYLOC, and the sum of their values must not exceed record length plus 1.

KEYLOC-value — Specifies the start position of the record key in the records. Value can be from 1 through 128.

Note: KEYLOC must be specified with KEYLEN, and the sum of their values must not exceed record length plus 1.

$BICR OCL and Utility Control Statement Sequence

```
// LOAD $BICR
// FILE NAME-COPYIN,UNIT- {I1|F1} ,LABEL-from-filename,...
[// FILE NAME-COPYO,UNIT- {F1|I1} ,LABEL-to-filename,...]
// RUN
[// TRANSFER ...]
// END
```

Notes:

1. If a new disk file is to be created from a multivolume diskette file, then the COPYO FILE statement must be given, and the required RECORDS or BLOCKS parameter must be large enough to contain the entire diskette file.
2. If a new disk file (with space requirements of a nonmultivolume diskette file) is to be created, do not specify the COPYO FILE statement.
3. If a new disk file larger than the diskette file is to be created, then the COPYO FILE statement must be specified with the required RECORDS or BLOCKS parameter.
4. If a file is being created on diskette, the COPYO FILE statement with a PACK parameter is required.

$BICR Example

In order to create a basic data exchange diskette file (JOEB1) from a disk file (JOE), you could enter:

```
// LOAD $BICR
// FILE NAME-COPYIN,UNIT-F1,LABEL-JOE
// FILE NAME-COPYO,UNIT-I1,LABEL-JOEB1,PACK-9
// RUN
// TRANSFER
// END
```

$BUILD—ALTERNATE SECTOR REBUILD UTILITY PROGRAM

This utility program allows you to display and correct data on the disk after a disk error occurs.

When a disk read or write error occurs, the data is written to an *alternate sector.* Disk alternate sectors are sectors reserved for use in place of defective disk sectors. The $BUILD utility program searches the alternate sectors of the disk for data that was unreadable because of a read/write error. Each sector containing unreadable data is printed, along with the sector logically preceding and the sector logically following it in the file.

The data is displayed on the display screen and by the printer in both character and hexadecimal format, as shown in Figure 6. The data is displayed in character format on the first line. If the character cannot be displayed, it is replaced by a blank. The data is also displayed in hexadecimal form on the second and third lines. The left hex digit of each byte is on the second line and the right digit is below it on the third line.

PRINTER OUTPUT:

```
                                   ALTERNATE SECTOR REBUILD UTILITY                               DATE  XX/XX/XX

  SS-00656   FILENAME-#LIBRARY
SECTOR BEFORE BAD DATA
1.......10........20........30........40........50........60........70........80........90........100........110........120.....128
    •Z 2             L                   E  "  2    "  2     2    "  2    "  2    " *2    •2              J        2

80303B7E0F0000383C003B3DC800C831303B00393C003731F81003732F8000393DF8B003732F8B003732F8A003735F8A3034F89003B3B003B3D303320350F863

1A515DD94216F15D3DE15D3307D0072848 37C15935D14F4E222D14F44249C1593F27DD14F4A224D14F4C21BD14F4C212D06B215C15155E151315251C067027A5

131......140........150........160........170........180........190........200........210........220........230........230........250...256
      2 ! P  2       T          ¢     ¢ 0           2        >        2       V       ¢  E  2    + K    " P    "2

03B20350F053035F0300393E303B2035034340034343F340034343035F81023436003537C03BF8000393E303B203403C35F80C8343D35C8373D35C837F800039

251D0631 21AD76721DC159335251C0682C06AC3696AC065C16465D06821060657EF06870012C27FC159355251C76A9C557274070ED267012FD767012F276C159

  SS-00657   FILENAME-#LIBRARY
SECTOR WITH BAD DATA
1.......10........20........30........40........50........60........70........80........90........100........110........120.....128
2  SYLC                                           F           2          0  8  3                                          M    2

F80EEDC00300A300B300BC8000B02C80A3001300300030B340F0C0F0F300BBC0F15200003F00C00F3000300390003 00B9000B00B008000B8000BC800DB00F008

27628336541 2E4222482607040D08012851 01413 8E13828C04FC64E4F5222900205C0340B03421236134523BC0005122C123C16C05C142AC182607044D05210C




131......140........150........160........170........180........190........200........210........220........230........230........250...256
          M          2  S                      `   2     2    •  2                          B             `  2  :      3

000B880C800D3003200B0F84E00300390090008000B720F90B00F80B00700F81100D0000D0B000D0D700C003900031013001742F917423100F103000C800300

122CC1007044423BC126827521052 3BC052C061C142A900206A86273886D0221CC01C2F01C2DC11B1C5142248C0001307510190F202A0F10363605155070C510

  SS-00658   FILENAME-#LIBRARY
SECTOR AFTER BAD DATA
1.......10........20........30........40........50........60........70........80........90........100........110........120.....128
122CC1007044423BC12682752105238C052C061C142A900206A86273B86D0221CC01C2F01C2DC11B1C5142248C0001307510190F202A0F10363605155070C510

FFFCCFFFFFFFFFFCCFFFFFFFFFFFFFFFCCFFFCFFFCFFFCFFFFFFCFFFFFCFFCFFFFCCFFCFCFFCFCCFFCFCFFFFFFFCFFFFFFFFFFFFFFCFFFCFCFFFFFFFFFFFFFFFFCFFF

6663366666666663366666666666666633666366636663666666366666366366663366636366363366636366666663666666666666636663636666666666666663666

131......140........150........160........170........180........190........200........210........220........230........230........250...256
SYM             SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

EED4400000000000EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE

2840000000007311222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
```

DISPLAY SCREEN OUTPUT:

```
ABCDEFGHI         JKLMNOPQR          STUVWXY
CCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDEEEEEEEEE
123456789ABCDEF0123456789ABCDEF012345678
COL=00001    SS-03741     FILENAME-HEXFILE
```

Figure 6. Example Output of a Disk Sector with Character and Hexadecimal Printout

After the unreadable data is displayed, you have two options:

- Bypass the data
- Correct the data

$BUILD is evoked by the BUILD procedure (see index entry: *BUILD procedure*).

Bypass Unreadable Data

If you wish to bypass the data, press the ENTER key on the keyboard. The $BUILD utility then searches for the next alternate sector with unreadable data. The next time $BUILD is evoked, the bypassed sector is displayed again.

Correct Unreadable Data

In order to correct the data, use the keyboard function keys to display the portion of the bad sector that you wish to correct. After the display is shifted to the desired position, place the cursor on either the character data line or the hexadecimal data line. Type the desired data over the unreadable data. The display screen provides the following information to help you correct the data:

- The displacement into the record (in decimal) of the character pointed to by the cursor: COL=00001 on the display screen in Figure 6
- The sector number: SS-03741 on the display screen in Figure 6
- The filename: FILENAME-HEXFILE on the display screen in Figure 6

After you have keyed all your corrections, if any, for a bad sector, press the REC ADV (record advance) key. The corrected sector will be rewritten to the disk, and $BUILD will search for other bad sectors. The next time $BUILD is evoked, the corrected sector will not be displayed.

Note: If you press the ENTER key after keying corrections, the corrected sector is not rewritten to the disk. If you cannot correct the data and wish to copy the data from a backup copy, advance the cursor in any position in the bad sector and press REC ADV, which removes the indication of bad data and permits you to copy the file from the diskette.

$BUILD Utility Control Statement Format

Utility control statements are not used.

$BUILD OCL Sequence

The following entries are needed to load and run the program:

```
// LOAD $BUILD
// RUN
```

$CNVRT–CONVERT DISKETTE HEADER LABEL UTILITY

The $CNVRT utility program converts the diskette header labels that were created prior to version 5 to a version 5 format.

Note: Unpredictable results may occur if diskette files with version 5 format header labels are processed by a preversion 5 SCP. A diskette file created by the $MAINT utility program (FROMLIBR procedure) in version 5 of the SCP, for example, *cannot* be used as input to the $MAINT utility program (TOLIBR procedure) in version 4 of the SCP.

$CNVRT is evoked by the CONVERT procedure (see index entry: *CONVERT procedure*).

$CNVRT Utility Control Statement Format

Utility control statements are not used.

$CNVRT OCL Sequence

```
// LOAD $CNVRT
// RUN
```

$COPY–DISK COPY/DISPLAY UTILITY PROGRAM

The disk copy/display utility has several uses:

- Copy an entire file from the disk to diskette(s), from diskette(s) to the disk, or from the disk to another location on the disk to:

  1. Provide a duplicate of a file

     Note: If, after copying a file to a diskette you delete the original file from the disk, the file on the diskette becomes the master copy of the file.

  2. Move a file to a larger disk area

- Delete records from a file (selected records are omitted from the copy; the original remains unchanged).

- Copy a portion of a file; you have the option of deleting selected records from the copy.

- Copy all data files (except #LIBRARY) on the disk to diskette(s) to create a backup copy of the files or to obtain more space on the disk; or, restore previously copied files from diskette(s) to the disk.

- Copy an indexed file putting the records in key order (reorganize the file) to improve the performance, in some cases, of programs that use the file. Selected records can be deleted from the copy.

Bypass Unreadable Data

If you wish to bypass the data, press the ENTER key on the keyboard. The $BUILD utility then searches for the next alternate sector with unreadable data. The next time $BUILD is evoked, the bypassed sector is displayed again.

Correct Unreadable Data

In order to correct the data, use the keyboard function keys to display the portion of the bad sector that you wish to correct. After the display is shifted to the desired position, place the cursor on either the character data line or the hexadecimal data line. Type the desired data over the unreadable data. The display screen provides the following information to help you correct the data:

- The displacement into the record (in decimal) of the character pointed to by the cursor: COL=00001 on the display screen in Figure 6
- The sector number: SS-03741 on the display screen in Figure 6
- The filename: FILENAME-HEXFILE on the display screen in Figure 6

After you have keyed all your corrections, if any, for a bad sector, press the REC ADV (record advance) key. The corrected sector will be rewritten to the disk, and $BUILD will search for other bad sectors. The next time $BUILD is evoked, the corrected sector will not be displayed.

Note: If you press the ENTER key after keying corrections, the corrected sector is not rewritten to the disk. If you cannot correct the data and wish to copy the data from a backup copy, advance the cursor in any position in the bad sector and press REC ADV, which removes the indication of bad data and permits you to copy the file from the diskette.

$BUILD Utility Control Statement Format

Utility control statements are not used.

$BUILD OCL Sequence

The following entries are needed to load and run the program:

```
// LOAD $BUILD
// RUN
```

$CNVRT–CONVERT DISKETTE HEADER LABEL UTILITY

The $CNVRT utility program converts the diskette header labels that were created prior to version 5 to a version 5 format.

Note: Unpredictable results may occur if diskette files with version 5 format header labels are processed by a preversion 5 SCP. A diskette file created by the $MAINT utility program (FROMLIBR procedure) in version 5 of the SCP, for example, *cannot* be used as input to the $MAINT utility program (TOLIBR procedure) in version 4 of the SCP.

$CNVRT is evoked by the CONVERT procedure (see index entry: *CONVERT procedure*).

$CNVRT Utility Control Statement Format

Utility control statements are not used.

$CNVRT OCL Sequence

```
// LOAD $CNVRT
// RUN
```

$COPY–DISK COPY/DISPLAY UTILITY PROGRAM

The disk copy/display utility has several uses:

- Copy an entire file from the disk to diskette(s), from diskette(s) to the disk, or from the disk to another location on the disk to:

  1. Provide a duplicate of a file

     Note: If, after copying a file to a diskette you delete the original file from the disk, the file on the diskette becomes the master copy of the file.

  2. Move a file to a larger disk area

- Delete records from a file (selected records are omitted from the copy; the original remains unchanged).

- Copy a portion of a file; you have the option of deleting selected records from the copy.

- Copy all data files (except #LIBRARY) on the disk to diskette(s) to create a backup copy of the files or to obtain more space on the disk; or, restore previously copied files from diskette(s) to the disk.

- Copy an indexed file putting the records in key order (reorganize the file) to improve the performance, in some cases, of programs that use the file. Selected records can be deleted from the copy.

- Add a disk file to an existing diskette file.

- Display all or part of a file (either on the display screen or printer, depending on the current SYSLIST assignment—see index entries: *STATUS procedure* and *SYSLIST procedure*) to check records for errors.

$COPY is evoked by the DISPLAY, ORGANIZE, RESTORE, and SAVE procedures (see index entries: *DISPLAY procedure, ORGANIZE procedure, RESTORE procedure,* and *SAVE procedure*).

Notes:
1. If you use $COPY to list a disk segment of an offline multivolume file (see index entry: *offline multivolume file*), the listing will include variable system data.
2. $COPY can copy a diskette file only if the file was copied to the diskette(s) by $COPY.

$COPY Utility Control Statement Formats

The different uses of $COPY require different utility control statements.

Use	Control Statements
Copy an entire file	// COPYFILE OUTPUT-DISK [,DELETE-'position,character'] [,REORG-{NO / YES}] // END

Copy a portion of a file	// COPYFILE OUTPUT-DISK [,DELETE-'position,character'] [,REORG-{NO / YES}]
	[// SELECT KEY,FROM-'key'
	// SELECT KEY,FROM-'key',TO-'key'
	// SELECT RECORD,FROM-number
	// SELECT RECORD,FROM-number,TO-number
	// SELECT PKY,FROM-'key'
	// SELECT PKY,FROM-'key'
	// SELECT PKY,FROM-'key',TO-'key']
	// END

Copy all data files on the disk to diskette, or restore previously copied files from diskette to the disk	// COPYALL TO-{F1 / I1} // END

Copy a sequential or direct file to an indexed file	// COPYFILE OUTPUT-DISK [,DELETE-'position,character']
	// KEY LENGTH-value-1,POSITION-value-2
	// END

Add a disk file to an existing diskette file	// COPYADD // END
Display an entire file	// COPYFILE [OUTPUT-PRINT / OUTPTX-PRINT] // END

Display part of a file	// COPYFILE [OUTPUT-PRINT / OUTPTX-PRINT] [,DELETE-'position,character']
	[// SELECT KEY,FROM-'key'
	// SELECT KEY,FROM-'key',TO-'key'
	// SELECT RECORD,FROM-number
	// SELECT RECORD,FROM-number,TO-number
	// SELECT PKY,FROM-'key'
	// SELECT PKY,FROM-'key',TO-'key']
	// END

$COPY Parameters

COPYFILE Statement

The COPYFILE statement specifies copy, display, and reorganize.

OUTPUT-DISK	The file or a portion of the file is copied from disk to diskette, from diskette to disk, or from one area to another on the disk.
OUTPUT-PRINT	The entire file or only part of the file is displayed in EBCDIC character format. *Note:* If the display is on the display screen, all lines are truncated to forty (40) characters.
OUTPTX-PRINT	The entire file or only part of the file is displayed in EBCDIC character format and in hexadecimal format. *Note:* If the display is on the display screen, all lines are truncated to forty (40) characters.
DELETE-'position, character'	This parameter is optional except when REORG-YES is specified for a sequential file. It means delete all records with the specified character in the specified record position. Character can either be one of the standard characters (see Appendix F. *IBM System/32 Characters* for the standard character and its hexadecimal equivalent) or the three characters Xdd, where X is constant and dd is the hexadecimal equivalent of any character. Position can be any position in the record (the first position is 1, second is 2, and so on) to a maximum of 999.
REORG-NO	Records are copied the way they are organized in the original file. REORG-NO is assumed if the REORG parameter is not specified.
REORG-YES	REORG-YES can be specified: • When copying an indexed file from the disk, in which case the records are to be copied in the same order as their keys appear in the index. • When copying a sequential file to a sequential file. The DELETE parameter—see the description preceding—is required when REORG-YES is specified for a sequential file.

SELECT Statement

The SELECT statement specifies which part of a file is to be copied or displayed. The SELECT statement is not valid for a COPYALL request.

KEY
or,FROM-'key'
PKY

For indexed files only. Copy or display only part of a file—from the record identified by the specified key to the end of the file (including the record with the specified key).

Note: You can specify the SELECT KEY or SELECT PKY parameters to select records from a diskette file only if the records in the diskette file are in ascending key order. An organized diskette file consists of records in ascending key order and is created in either of the following ways:

- Specify the REORG parameter on the COPYFILE statement as REORG-YES. The file copied to diskette is then an organized diskette file.
- Use the ORGANIZE procedure to create an organized diskette file.

In addition, the organized diskette file must not have records added to it.

If you select records from an indexed file to copy or display, and those records are not in ascending key order, the results are unpredictable.

This note also applies to the following description of the KEY or PKY (FROM and TO keys) parameters.

KEY
or,FROM-'key',TO-'key'
PKY

For indexed files only. Copy or display only part of a file—from the record identified by the specified FROM key to the record identified by the specified TO key (including the two records with the specified keys).

Note: To copy or display only one record, make the FROM and TO keys the same. If the specified record key does not exist, no records are copied or displayed.

RECORD,FROM-number

Copy or display only part of a file—from the record identified by the specified record number to the end of the file (including the record identified by the specified number).

RECORD,FROM-number,TO-number

Copy or display only part of a file—from the record identified by the FROM record number to the record identified by the TO record number (including the two records identified by the FROM and TO record numbers).

Note: To copy or display only one record, make the FROM and TO numbers the same. If the specified record number does not exist, no records are copied or displayed.

KEY Statement

The KEY statement specifies the length and position of record keys for a file. The statement is used to create an indexed file from a sequential or direct file. When the KEY statement is used, both the LENGTH and POSITION parameter must be specified and the sum of their values must not exceed record length plus 1.

LENGTH-value-1

The LENGTH parameter specifies the length of the key in bytes. Value-1 can be any number from 1 through 29.

POSITION-value-2

The position specifies the position of the key in the records. This position is the leftmost byte of the key. Value-2 can be any number from 1 through 999.

COPYALL Statement

The COPYALL statement specifies that all data files on the disk (but not #LIBRARY) be copied to diskette(s), or specifies that files previously copied be restored from diskette(s) to the disk.

TO-{F1 / I1}

Specifies that the disk (F1) or a set of diskettes (I1) is to contain the copy.

COPYADD Statement

The COPYADD statement requests addition of a disk file to an existing diskette file. The disk file is added to the diskette file so that restoring the extended file creates a single disk file. The user must specify on the COPYIN file statement the name of the file to be added and on the COPYO file statement the name of the file to be extended.

$COPY Parameter Summary

OUTPUT and OUTPTX Parameters (COPYFILE)

These parameters specify whether you want to copy or display data files.

Copying a File: The parameter OUTPUT-DISK means the file is to be copied. $COPY can copy a file from the disk to diskette(s), from diskette(s) to the disk, or from one area on the disk to another area on the disk. Data files copied to and from diskette(s) are system files (see Appendix C).

In copying a disk file to diskette(s), the disk file is, in effect, dumped onto diskette(s), so that when it is copied back to the disk, its original format (filename, file size, retention) is retained, unless the original format is overridden by the appropriate parameter(s) (LABEL, BLOCKS or RECORDS, RETAIN) on the COPYO file statement. For example, the RECORDS or BLOCKS parameter might have to be specified for the disk file if records have been added to the diskette file.

The OCL load sequence for the $COPY program indicates (1) the name and unit of the file being copied, and (2) the name and unit of the copy being created. If the file is to be created on the disk, then the size of the file can be specified.

Displaying Files: OUTPUT-PRINT means the file is displayed in EBCDIC character format, and OUTPTX-PRINT means the file is displayed in hexadecimal format.

The $COPY program uses as many lines as it needs to display the contents of a record (100 characters per line are printed; if the display screen is used, only the first 24 characters of each record are displayed). After displaying the last record, the program prints a message stating the number of records displayed. Characters that have no graphic display symbol (unprintable characters) are displayed as two-digit hexadecimal numbers in over-and-under format.

The following examples show the hexadecimal numbers in over-and-under format of an unprintable character (B6) for both parameters:

```
ABCDEF J12345 }
      B       } OUTPUT-PRINT
      6       }

A B C D E F   J 1 2 3 4 5 }
C C C C C C B D F F F F F } OUTPTX-PRINT
1 2 3 4 5 6 6 1 1 2 3 4 5 }
```

Records from indexed files are displayed in the order of the records, unless you specify SELECT KEY and/or SELECT PKY and/or REORG-YES. For each record, the program displays the record key followed by the contents of the record.

Records from sequential, indexed, and direct files on diskette are displayed in the order they appear in the file. For each record, the program displays the relative record number for sequential and direct files, or the record key for indexed files followed by the contents of the record.

DELETE Parameter (COPYFILE)

The $COPY program can omit records of one type while copying or displaying a single file.

The form of the parameter for omitting records is DELETE-'position,character'. Character is the character or hexadecimal equivalent (Xdd) that identifies the records. Position is the position of the character in the records. For example, the parameters DELETE-'100,XE2' and DELETE-'100,S' would yield the same results. (See Appendix F, *IBM System/32 Characters,* for the character and its hexadecimal equivalent.)

REORG Parameter (COPYFILE)

In copying or displaying an indexed file, the program can reorganize the file so that the records in the data portion are in the same order as their keys in the file index. The REORG parameter tells the program whether or not to reorganize the file. The file can be reorganized while it is being copied from F1 to either I1 or F1.

SELECT KEY and SELECT PKY Parameters

The SELECT KEY and SELECT PKY parameters are used to select records from an indexed file to copy or display part of that indexed file. The SELECT PKY parameter applies to an indexed file that contains packed keys.

When you specify either of these parameters to select records from a diskette file, the records in the diskette file must be in ascending key order. To put the records in ascending key order, you can either specify the REORG parameter on the COPYFILE statement of $COPY as REORG-YES (to organize the diskette file while it is being copied from disk), or you can create an organized diskette file using the ORGANIZE procedure. Either way, the organized diskette file should not have records added to it.

If you select records from an indexed file to copy or display, and those records are not in ascending key order, the results are unpredictable.

Related parameters of SELECT KEY and SELECT PKY are the FROM and TO parameters. If none of the keys in the file index begin with the characters indicated in the FROM or TO parameters, the program uses the key beginning with the next higher characters than in the FROM parameter and the key beginning with the next lower characters than in the TO parameter.

The TO parameter can be omitted. When this is done, the program uses the last key in the index as the TO key.

There may be fewer characters in the FROM or TO parameter than are contained in the actual keys.

For example, assume that the following are consecutive record keys in an index: A0999, A1000, A1010, A1040, A1500, A1510, A1690, and A1955. The parameters FROM-'A10' and TO-'A15' refer to record keys A1000, A1010, A1040, A1500, and A1510.

If you want to copy or display only one record, make the FROM and TO keys the same.

SELECT RECORD Parameter

This parameter is used to copy or display a portion of a file. This parameter uses relative record numbers to identify the records to be copied or displayed.

Relative record numbers identify a record's location with respect to other records in the file. The relative record number of the first record is 1, the number of the second record is 2, and so on.

The related parameters are FROM and TO. The FROM parameter (FROM-number) gives the relative record number of the first record to be copied or displayed. The TO parameter (TO-number) gives the number of the last record to be copied or displayed. Records between those two records in the file are also copied or displayed.

For example, the parameters FROM-1 and TO-30 mean that the first thirty records (1–30) in the file will be copied or displayed.

You can omit the TO parameter. If you do, the program uses the number of the last record in the file as the TO number. If you want to copy or display only one record, use the same number in the FROM and TO parameters.

TO Parameter (COPYALL)

This parameter specifies whether diskette or disk will contain the copy. I1 and F1 are the only values allowed. When I1 is specified, all data files on the disk are copied to the same number of files on one or more diskettes. When F1 is specified, all files previously copied to diskette(s) are restored to the disk from the diskette(s).

Copying All Disk Files: The output of $COPY when copying all disk data files to diskette is: Files on one or more diskettes which had no active files on them. Each diskette file contains information about the file as it appeared on the disk. The set of files is associated with a name of #SAVE unless a different name was specified via the LABEL parameter in the COPYO file statement.

Restoring Disk Files: When restoring all previously saved files to the disk, you can specify the name associated with the diskette files (if the name #SAVE was not used) via the LABEL parameter on the COPYIN file statement.

To restore only one file from diskette(s) containing all files previously copied from the disk, you must specify the name of the file to be restored on the COPYIN file statement, and you can specify a name for the new disk file on the COPYO file statement.

$COPY OCL and Utility Control Statement Sequence

When copying, reorganizing, or displaying files, the user must (1) describe the disk files being copied or displayed and (2) describe the file being created. To do this, the following OCL statements are needed:

```
// LOAD $COPY
// FILE NAME-COPYIN   [,UNIT- {F1 / I1}],LABEL-filename
{ // FILE NAME-COPYO,UNIT-I1,LABEL-filename [,RETAIN- {retention-days / 1}]
    ,PACK-vol-id
      or
  // FILE NAME-COPYO [,UNIT-F1] ,LABEL-filename [, {RECORDS-number / BLOCKS-number}]
  [,RETAIN- {T / P / S}] }
// RUN
// COPYALL...
      or
// COPYADD
      or
// COPYFILE...
[// SELECT...]
[// KEY...   ]
// END
```

Statement Entry	Meaning
// LOAD	
$COPY	Name of disk copy/display program.
// FILE	
NAME-COPYIN	Name $COPY uses to refer to the file to be copied, reorganized, or displayed.
UNIT-{F1 / I1}	Identifies either the disk (F1) or a diskette (I1) as containing the file to be copied.
LABEL-filename	Name by which the file to be copied is identified. This parameter must be used to specify the name associated with the entire set of files copied when the COPYALL statement is used to copy from diskette.
// FILE	
NAME-COPYO	Name $COPY uses to refer to output file being created. (This OCL statement is not needed for displaying a file.)
UNIT-{F1 / I1}	Specifies location of output file: disk (F1) or diskette (I1).
LABEL-filename	Name by which output file is to be identified. This parameter must be used to specify the name associated with the entire set of files being copied when the COPYALL statement is used to copy to diskette.
[{RECORDS-number / BLOCKS-number}]	Size of output file expressed either as number of records (RECORDS) or number of disk blocks (BLOCKS). Used only when copying individual files to the disk.
[RETAIN-{T / P / S}]	Retention designation of the disk output file: T is temporary, P is permanent, S is scratch.
or	
[retention-days 1]	Retention designation of diskette output file expressed in number of days. Default is one day.
PACK-vol-id	The diskette volume label. Meaningful only if the unit designation is I1.

$COPY File Retention Summary

The effect that the RETAIN parameter retention code (P, T, or S) has on the retention of a disk file for the $COPY program depends on whether:

- The file is an input file or an output file
- The file is on disk or diskette

Each file that exists on disk has a record in the VTOC of system information describing the file, such as filename, file date, file organization, and retention code. This record is called a VTOC format 1. A disk VTOC format 1 has a retention code of either P (permanent) or T (temporary).

A file being processed by a program must also have a format 1 in the SWA (scheduler work area). The SWA format 1 is created by the FILE statement. A SWA format 1 has a retention code of P, T, or S.

For a file existing on disk (input file), the SWA format 1 is the same as the existing VTOC format 1; therefore, the retention code is the same unless it is modified by the RETAIN parameter in the FILE statement for the input file (COPYIN). For a file being created on disk (output file), the retention code is specified in the RETAIN parameter (or defaults to T if that parameter is not used) for the output file (COPYO).

The SWA format 1, with a retention code of P or T, becomes the VTOC format 1 at the end of the job. The SWA format 1, with a retention code of S, is deleted at the end of the job. Therefore, a file with an SWA format 1, with a retention code of S, exists only during job execution. If an SWA format 1, with a retention code of S, identifies a file in the VTOC with a retention code of T, the VTOC format 1 is also deleted.

Note: If both the SWA format 1 retention code for the disk input file and the SWA format 1 retention code for the disk output file have a retention code of S, neither file will exist at the end of the job.

In the following examples, which summarize the results of the RETAIN parameter on the VTOC format 1 retention code, the // FILE statement named COPYIN identifies the input file (file being copied) and COPYO identifies the output file (file being created).

Example 1: Existing Disk File

The RETAIN parameter is optional in the FILE statement that describes the input file, COPYIN. However, if you are accessing a temporary file and want to delete that file at the end of the job, include the RETAIN parameter with a retention code of S. The SWA format 1 modifies the VTOC format 1 for that file and the VTOC format 1 is deleted at the end of the job. Otherwise, the input file retention code is not altered in the VTOC format 1. The following summary shows the effect of the RETAIN parameter retention code on the SWA format 1 for a disk input file:

VTOC Format 1 Retention Code for Disk Input File	RETAIN Parameter on COPYIN FILE Statement	SWA Format 1 Retention Code for Disk Input File
P	P	P
P	T	P
P	S	P
T	P	T
T	T	T
T	S	S

Example 2: Disk to Disk

If the input file, identified by the FILE statement named COPYIN, resides on disk, and the output file, identified by the FILE statement named COPYO, will reside on disk, the SWA format 1 for the output file becomes whatever is specified in the RETAIN parameter as shown in the following summary:

SWA Format 1 Retention Code for Disk Input File	RETAIN Parameter on COPYO FILE Statement	SWA Format 1 Retention Code for Disk Output File
P	P	P
P	T	T
P	S	S
T	P	P
T	T	T
T	S	S
S	P	P
S	T	T
S	S	S

Note: The retention code of the input file does not change unless you also code the RETAIN parameter for COPYIN as shown in example 1.

Example 3: Diskette to Disk

If the input file, identified by the FILE statement named COPYIN, resides on diskette, the SWA format 1 retention code for this file is the SWA format 1 retention code of the disk file from which it was created. The output file, identified by the FILE statement named COPYO, will be recreated on disk. The following summary shows the effect of the RETAIN parameter on the SWA format 1 retention code for the output file:

SWA Format 1 Retention Code for Diskette File	RETAIN Parameter on COPYO FILE Statement	SWA Format 1 Retention Code for Disk Output File
P	P	P
P	T	P
P	S	P
T	P	P
T	T	T
T	S	T
S	P	P
S	T	T
S	S	T

$COPY Examples

Copy all disk files to diskette(s):

```
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-F1
// FILE NAME-COPYO,UNIT-I1,LABEL-#SAVE,PACK-vol-id
// RUN
// COPYALL TO-I1
// END
```

Copy a diskette file (JOE) to a disk file (JOEF):

```
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-I1,LABEL-JOE
// FILE NAME-COPYO,UNIT-F1,LABEL-JOEF,BLOCKS-100,RETAIN-P
// RUN
// COPYFILE OUTPUT-DISK
// END
```

Print from the diskette file JON all records with keys from ADAMS to BAKER:

```
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-I1,LABEL-JON
// RUN
// COPYFILE OUTPUT-PRINT
// SELECT KEY,FROM-'ADAMS',TO-'BAKER'
// END
```

Copy back to the disk the entire set of files previously copied from the disk to diskette(s):

```
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-I1,LABEL-#SAVE
// FILE NAME-COPYO
// RUN
// COPYALL TO-F1
// END
```

$DELET—FILE DELETE UTILITY PROGRAM

The $DELET program frees the space occupied by existing files for use by new files.

The space is freed in the following ways:

SCRATCH — Changes the diskette file(s) expiration date to the current job date. For disk file(s), SCRATCH removes the VTOC entry.

REMOVE — Removes the VTOC entry with the option of erasing the contents of the named file(s) on the disk or diskette by overwriting with binary zeros.

If you want to delete more than one file, additional control statements must be used. The end statement (// END) must follow the last SCRATCH or REMOVE statement.

You can delete permanent disk data files only by using the $DELET program. The system file #LIBRARY cannot be deleted.

$DELET is evoked by the DELETE procedure and JOBSTR procedure (see index entries: *DELETE procedure* and *JOBSTR procedure*).

$DELET Utility Control Statement Formats

Use	Control Statements
Scratch the VTOC entry for the named file	// SCRATCH UNIT-{F1 / I1},LABEL-filename [,PACK-vol-id] // END
Scratch the VTOC entry for the named file identified by the specified creation date	// SCRATCH UNIT-{F1 / I1},LABEL-filename,DATE-{mmddyy / ddmmyy / yymmdd} [,PACK-vol-id] // END

$DELET OCL and Utility Control Statement Sequence

To initiate the $DELET program through OCL, the following is required:

```
// LOAD $DELET
// RUN
// SCRATCH UNIT-{F1|I1} ,LABEL-{filename|ALL} [,DATE-{mmddyy|ddmmyy|yymmdd}] [,PACK-vol-id]
```

and/or

```
// REMOVE UNIT-{F1|I1} ,LABEL-{filename|ALL} [,DATE-{mmddyy|ddmmyy|yymmdd}] [,DATA-{NO|YES}] [,PACK-vol-id]

// END
```

$DELET Examples

In order to remove the VTOC entry JOE (created October 14, 1974) on the disk, you could enter:

```
// LOAD $DELET
// RUN
// SCRATCH UNIT-F1,LABEL-JOE,DATE-101474
// END
```

In order to remove and erase all files named JON on the disk, you would enter the following:

```
// LOAD $DELET
// RUN
// REMOVE UNIT-F1,LABEL-JON,DATA-YES
// END
```

$DUPRD—DISKETTE COPY UTILITY PROGRAM

The diskette copy program copies a single file on a diskette or all files on a diskette to one or more output diskettes to provide a duplicate of the file(s). When an entire diskette is copied, unused space on the input diskette can be gathered together into a single free space on the output diskette(s). The output diskette(s) must be in the same format (512-bytes per sector extended format or 128-bytes per sector basic data exchange format) as the diskette being copied.

Diskettes with permanent files are the diskettes normally copied. Because diskettes can develop surface irregularities as they undergo the wear of continued use, it is a good idea to copy your important files soon after they are created.

$DUPRD is evoked by the COPYI1 procedure (see index entry: *COPYI1 procedure*).

$DUPRD Utility Control Statement Formats

Use	Control Statement
Copy all files on a diskette to one or more output diskettes	// COPYI1 NAME-ALL,PACK-vol-id [,DELETE-{YES / NO}] [,PRESERVE-{YES / NO}] [,COPIES-{number of copies / 1}] // END
Copy one file on a diskette to one or more output diskettes	// COPYI1 NAME-filename,PACK-vol-id [,PRESERVE-{YES / NO}] [,COPIES-{number of copies / 1}] // END

$DUPRD Parameters

NAME-ALL	Requests that all files on a diskette be copied to one or more output diskettes.
NAME-filename	Specifies the name of the single file on a diskette that is to be copied to one or more output diskettes.
PACK-vol-id	Identifies the output diskette(s).
DELETE-YES	Indicates that no expired files on the input diskette are to be copied. The DELETE parameter is valid only with NAME-ALL. *Note:* If a multivolume file exists on the input diskette, the DELETE-YES parameter is ignored.
DELETE-NO	Indicates that expired files on the input diskette are to be copied to the new diskette(s). The DELETE parameter is valid only with NAME-ALL. If the DELETE parameter is not specified, DELETE-NO is the default.
PRESERVE-YES	Indicates that the end of extent for each file copied is to be preserved at the same relative displacement past the end of data on the output diskette(s) as it was on the input diskette.
PRESERVE-NO	Indicates that the end of extent for each file is not to be preserved. If the PRESERVE parameter is not specified, PRESERVE-NO is the default.
COPIES-{number of copies / 1}	Specifies the number of output diskettes to be copied from one input diskette. If the COPIES parameter is not specified, 1 is the default. The maximum number of copies allowed is 99.

$DUPRD Parameter Summary

NAME Parameter

There are two types of NAME parameters:

NAME-ALL and NAME-filename.

The NAME-ALL parameter indicates that all files on the inserted diskette are to be copied to one or more output diskettes. The NAME-filename parameter specifies the name of the single file that is to be copied from one input diskette to one or more output diskettes.

When all files or a single file on a diskette are copied, the input and output diskettes may differ in the volume identification and alternate track information. If NAME-ALL is specified, the DELETE parameter can be used.

The diskettes that are being copied can contain basic data exchange files or system files (see Appendix C). The diskette(s) to contain the copy must not contain active files if all files on a diskette are being copied, or if the file to be copied is part of a multivolume file. For either NAME-ALL or NAME-filename, if a diskette to be copied is a portion of a multivolume file, only that one portion of the multivolume file will be copied.

To perform the copy, $DUPRD requires enough space on the disk to contain the data being copied. $DUPRD copies the file or diskette to the disk, then displays a message telling the operator to insert the diskette that is to contain the copy. For each copy that is specified in the COPIES parameter, a message tells the operator to insert another diskette. After transferring the copy from the disk to each output diskette that is inserted, $DUPRD execution is complete.

PACK Parameter

The PACK parameter supplies the volume identification (vol-id) of the output diskette. The PACK parameter is always required.

DELETE Parameter

The DELETE parameter can be used if NAME-ALL is specified. DELETE-YFS specifies that expired files on the input diskette are to be deleted. (Space between files is eliminated; the files are physically contiguous on the new diskette.) However, if a multivolume file exists on the input diskette, the DELETE-YES parameter is ignored. DELETE-NO specifies that expired files on the input diskette are to be copied. DELETE-NO is the default.

PRESERVE Parameter

PRESERVE-YES indicates that for each file copied, the end of extent is preserved at the same relative displacement past the end of data on the output diskette(s) as it was on the input diskette. PRESERVE-NO indicates that the end of extent for each file is not to be preserved. PRESERVE-NO is the default.

COPIES Parameter

The COPIES parameter specifies the number of output diskettes to be copied from one input diskette. The maximum number of copies is 99. COPIES-1 is the default.

$DUPRD OCL and Utility Control Statement Sequence

To initiate the diskette copy program, the following OCL is required:

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1,...
// RUN
// COPYI1...
// END
```

$DUPRD Examples

Copy all files on a diskette to the diskette with a vol-id of 123456.

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1
// RUN
// COPYI1 NAME-ALL,PACK-123456
// END
```

Copy a file on a diskette (with filename of JIM and creation date of 01-02-75) to another diskette (with vol-id of 345678).

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1,DATE-010275
// RUN
// COPYI1 NAME-JIM,PACK-345678
// END
```

$FREE--DISK REORGANIZATION UTILITY PROGRAM

The $FREE utility program causes all free space on the disk, except free space within files and the system library, to be accumulated into a single area. The location of the area of free space depends upon the parameters specified in the $FREE utility control statement.

$FREE cannot be run while in inquiry mode.

If a system failure occurs during the running of $FREE, $FREE must be run again to ensure the integrity of data on the disk. If the disk VTOC is to be displayed, run the $LABEL utility program. If $FREE must be run, the following message appears as part of the information displayed by $LABEL:

```
$FREE MUST BE RUN BEFORE INFORMATION CAN BE OBTAINED FROM
THIS FILE.
```

$FREE must then be the next program run. No other program except $LABEL should be run until $FREE completes.

Note: Because files are physically moved by $FREE, the locations specified by LOCATION parameters in FILE statements for the moved files (see index entry: *// FILE statement*) will not be valid. To determine new file locations after using $FREE, use the $LABEL utility or CATALOG procedure to display the disk VTOC.

$FREE Utility Control Statement Format

```
// COMPRESS [FREE- {LOW }]
                  {HIGH}
```

$FREE Parameters

FREE- {LOW } The FREE parameter specifies the direction in which the free space
      {HIGH} is to be accumulated.

- FREE-LOW specifies that free space is accumulated at the lowest available block numbers on the disk; that is, the free space immediately following the system library.
- FREE-HIGH specifies that free space is accumulated at the highest available block numbers on the disk. FREE-HIGH is the default.

$FREE OCL and Utility Control Statement Sequence

To initiate the $FREE program through OCL, the following is required:

```
// LOAD $FREE
// RUN
[// COMPRESS [FREE- {LOW }] ]
                  {HIGH}
// END
```

$FREE Examples

To accumulate the free space on disk at the high block location addresses, use any of the following examples. They all accomplish the same logical results.

```
// LOAD $FREE
// RUN
// END
```

or

```
// LOAD $FREE
// RUN
// (one of the following:)
   COMPRESS
   COMPRESS FREE-HIGH
// END
```

Note: The same results can also be obtained with the following example:

```
// LOAD $PACK
// RUN
   or
// COMPRESS command statement
```

To accumulate the free space on disk at the low block location addresses (between #LIBRARY and data files), use the following example:

```
// LOAD $FREE
// RUN
// COMPRESS FREE-LOW
// END
```

$HIST—HISTORY FILE DISPLAY UTILITY PROGRAM

The $HIST utility program lists, according to the current SYSLIST assignment (see index entry: *SYSLIST procedure*), the contents of the HISTORY file. The HISTORY file is an area on the disk reserved for collecting information such as OCL statements entered, utility control statements entered, error messages displayed, and the operator's response to each error message. Thus, the contents of the HISTORY file allows you to trace the sequence of events leading to current system status.

Because the HISTORY file is limited in size to thirty-nine 256-byte sectors, the number of events reflected in the HISTORY file at a particular time varies with the length of entries in the file. Once the file is filled, each new entry causes the oldest entry to be dropped from the file. When the file is listed, the oldest entry is displayed or printed first, and the most recent entry is displayed or printed last.

$HIST is evoked by the HISTORY procedure (see index entry: *HISTORY procedure*).

$HIST Utility Control Statement Formats

Use	Control Statement
Display only previously displayed HISTORY file data	[// DISPLAY] // END
List complete contents of HISTORY file (including items not previously displayed)	// DISPLAY ALL // END

$DUPRD Parameter Summary

NAME Parameter

There are two types of NAME parameters:

NAME-ALL and NAME-filename.

The NAME-ALL parameter indicates that all files on the inserted diskette are to be copied to one or more output diskettes. The NAME-filename parameter specifies the name of the single file that is to be copied from one input diskette to one or more output diskettes.

When all files or a single file on a diskette are copied, the input and output diskettes may differ in the volume identification and alternate track information. If NAME-ALL is specified, the DELETE parameter can be used.

The diskettes that are being copied can contain basic data exchange files or system files (see Appendix C). The diskette(s) to contain the copy must not contain active files if all files on a diskette are being copied, or if the file to be copied is part of a multivolume file. For either NAME-ALL or NAME-filename, if a diskette to be copied is a portion of a multivolume file, only that one portion of the multivolume file will be copied.

To perform the copy, $DUPRD requires enough space on the disk to contain the data being copied. $DUPRD copies the file or diskette to the disk, then displays a message telling the operator to insert the diskette that is to contain the copy. For each copy that is specified in the COPIES parameter, a message tells the operator to insert another diskette. After transferring the copy from the disk to each output diskette that is inserted, $DUPRD execution is complete.

PACK Parameter

The PACK parameter supplies the volume identification (vol-id) of the output diskette. The PACK parameter is always required.

DELETE Parameter

The DELETE parameter can be used if NAME-ALL is specified. DELETE-YES specifies that expired files on the input diskette are to be deleted. (Space between files is eliminated; the files are physically contiguous on the new diskette.) However, if a multivolume file exists on the input diskette, the DELETE-YES parameter is ignored. DELETE-NO specifies that expired files on the input diskette are to be copied. DELETE-NO is the default.

PRESERVE Parameter

PRESERVE-YES indicates that for each file copied, the end of extent is preserved at the same relative displacement past the end of data on the output diskette(s) as it was on the input diskette. PRESERVE-NO indicates that the end of extent for each file is not to be preserved. PRESERVE-NO is the default.

COPIES Parameter

The COPIES parameter specifies the number of output diskettes to be copied from one input diskette. The maximum number of copies is 99. COPIES-1 is the default.

$DUPRD OCL and Utility Control Statement Sequence

To initiate the diskette copy program, the following OCL is required:

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1,...
// RUN
// COPYI1...
// END
```

$DUPRD Examples

Copy all files on a diskette to the diskette with a vol-id of 123456.

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1
// RUN
// COPYI1 NAME-ALL,PACK-123456
// END
```

Copy a file on a diskette (with filename of JIM and creation date of 01-02-75) to another diskette (with vol-id of 345678).

```
// LOAD $DUPRD
// FILE NAME-COPYI1,UNIT-I1,DATE-010275
// RUN
// COPYI1 NAME-JIM,PACK-345678
// END
```

Delete (DELETE)

If the DELETE option of $INIT is requested, the operator is notified via the display screen when any active files exist on the inserted diskette. If active files do exist, the job can be canceled or the files can be deleted. If the DELETE option is taken, the VTOC for the diskette is set to indicate that one file, DATA, occupies tracks 1-73, and DATA is empty. The vol-id specified with the DELETE option is compared with the vol-id in the diskette volume label on track 0. They must be identical for deletion to occur. The owner-id information specified with the DELETE option is not compared to information in the volume label.

Rename (RENAME)

If the RENAME option is chosen instead of FORMAT, FORMAT2, or DELETE, only the volume label (track 0) is changed. The vol-id and owner-id fields are replaced by the contents of the PACK and ID parameters, respectively. These parameters are specified with the RENAME option. If a new vol-id is not specified, the system date is used. If owner-id is not specified, OWNERID is used.

Diskette Defects Encountered During Processing

If the system encounters diskettes with physical defects during output operation, the following information will apply.

If a defect is discovered while a job is being processed, the system will make one or more attempts (called retries) to read or write the bad sector. If the retries are not successful and the program is creating output to diskette, the file is closed at the beginning of the operation during which the error occurred, and normally at the start of a track. The operator is notified that the diskette contains a defect and is given the option of inserting another diskette and continuing the operation (which will result in a multivolume file) or terminating the job and restarting with an error-free diskette.

To restore to full use, the diskette should be initialized; however, if the initialization process results in discovery of more than two defective tracks, the diskette is unusable.

$INIT Utility Control Statement Formats

The utility control statement for $INIT functions must appear in the order shown:

Use	Control Statement

Initialize a diskette	// UIN OPTION-{FORMAT / FORMAT2} [,RECL-{number / 080}]
	[// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID]]
	// END

Delete files on a diskette	// UIN OPTION-DELETE [,RECL-{number / 080}]
	[// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID]]
	// END

Rename a diskette	[// UIN OPTION-RENAME [,RECL-{number / 080}]] [// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID]] // END

$INIT Parameters

UIN Statement

The UIN statement specifies which $INIT option is selected and the record length the header labels contain.

OPTION-{FORMAT / FORMAT2}	Initializes a diskette as a basic data exchange format diskette (FORMAT) with 128-byte data sectors or as an extended format diskette with 512-byte data sectors (FORMAT2). For more details on FORMAT and FORMAT2, see index entry: *INIT command statement.*
OPTION-DELETE	Deletes files on a diskette.
OPTION-RENAME	Renames a diskette. RENAME is the option selected if no option is specified.
RECL-{number / 080}	Specifies the record length to be inserted into the header labels (HDR1 and DDR1). 080 is the default.

VOL Statement

The VOL statement provides information to be written in the volume label.

PACK-vol-id system date	The PACK parameter specifies the vol-id. If the PACK parameter is not used, the system date is the default.
ID-owner-id OWNERID	The ID parameter specifies information for the owner-id field of the volume label. If the ID parameter is not used, OWNERID is the default.

$INIT Parameter Summary

OPTION- {FORMAT / FORMAT2}

If FORMAT or FORMAT2 is specified, the initialization function of $INIT is selected. The initialization function of $INIT formats and tests a diskette. Track 0 is built or rebuilt and tested for defects. If any defects are found on track 0, the diskette is unusable.

Tracks are tested through attempts to write IDs and records consisting of 128 bytes (FORMAT) or 512 bytes (FORMAT2) of hex E5 on the tracks. If a defect is found within a track, the entire track is marked defective (track IDs all hex FF) and an alternate is assigned. If more than two bad tracks are found, the job is terminated and the diskette is not usable.

Tracks are initialized by writing sectors of blanks on all data tracks (1-74). During initialization the VTOC is set to indicate that one file, DATA, occupies tracks 1–73, and DATA is empty. The vol-id and owner-id fields of the volume label (track 0) are replaced by the vol-id and owner-id given in the PACK and ID parameters, respectively (see following). If neither parameter is used, the system date and OWNERID are written in the vol-id and owner-id fields, respectively.

For more details on FORMAT and FORMAT2, see index entry: *INIT command statement.* A description of diskette formats is in Appendix C.

OPTION-DELETE

The DELETE function deletes files from a diskette by setting the VTOC to indicate that one file, DATA, occupies the entire diskette, and DATA is empty.

OPTION-RENAME

The RENAME function places the vol-id specified in the PACK parameter (see following), or the system date, if PACK is not used, in the vol-id field of the volume label (track 0). The vol-id is left-adjusted and padded with blanks. If the ID parameter is used (see following), as many as 14 characters of owner identification information, left-adjusted and padded with blanks, are placed in the owner-id field of the volume label. If the ID parameter is not used, OWNERID is placed in the owner identification field.

RECL- {number / 080}

Specifies the record length contained in the HDR1 and DDR1 header labels. 080 is the default. The maximum number of 128-byte sectors (FORMAT) is 128. The maximum number for 512-byte sectors (FORMAT2) is 512. This is also true when using OPTION-DELETE. When using OPTION-RENAME, RECL is ignored. If RECL-080 is entered or RECL is allowed to default, the record length is inserted into the header labels as 080. Otherwise the record length is inserted as 00nnn, where nnn is the number entered.

PACK-vol-id
system date

During initialization (FORMAT or FORMAT2) $INIT writes the vol-id specified by the PACK parameter in the volume label of the diskette being initialized. The vol-id can be as many as six alphameric characters. The system date is written if the PACK parameter is not specified.

In the DELETE function, the vol-id specified or system date must be equal to the vol-id existing on the inserted diskette, or the DELETE function is not performed.

In the RENAME function, the vol-id specified or the system date is written in the volume label of the inserted diskette.

ID-owner-id
OWNERID

The ID parameter specifies owner information to be written in the volume label to further identify a diskette. As many as 14 characters can be specified. Any combination of characters except single quotation marks ('), commas, and leading or embedded blanks can be specified. If the ID parameter is not used, OWNERID is written in the owner-id field of the volume label.

Owner identification information is strictly for the user. It is not used by the system to verify that the appropriate diskette is being used for a job.

$INIT OCL and Utility Control Statement Sequence

To initiate the $INIT program through OCL, the following is required:

```
// LOAD $INIT
// RUN
[ // UIN OPTION- { FORMAT  } [ ,RECL- { number } ] ]
               { FORMAT2 }           { 080    }
               { DELETE  }
               { RENAME  }
[ // VOL [ PACK-vol-id ][ ,ID-owner-id ] ]
         [ system date ][ ,OWNERID    ]
// END
```

$INIT Examples

In order to name a diskette JIM, you could enter:

```
// LOAD $INIT
// RUN
// VOL PACK-JIM
// END
```

To initialize to 128-byte data sectors, test for bad tracks, name the diskette APRO, and insert an owner identification of FRANT, you would enter:

```
// LOAD $INIT
// RUN
// UIN OPTION-FORMAT
// VOL PACK-APRO,ID-FRANT
// END
```

This page intentionally left blank

$HIST OCL and Utility Control Statement Sequence

To initiate the $HIST utility program through OCL, the following is required:

```
// LOAD $HIST
// RUN
[// DISPLAY...]
// END
```

$HIST Examples

Display only previously displayed HISTORY file data:

```
// LOAD $HIST
// RUN
// END
```

Display the entire HISTORY file:

```
// LOAD $HIST
// RUN
// DISPLAY ALL
// END
```

Display the entire HISTORY file and remove all entries after the file is shown:

```
// LOAD $HIST
// RUN
// DISPLAY ALL,RESET
// END
```

$INIT–DISKETTE LABELING AND INITIALIZATION UTILITY PROGRAM

$INIT, which is evoked by the INIT procedure (see index entry: *INIT procedure*), performs three functions:

- Initializes (formats) diskettes
- Deletes files from diskettes
- Renames diskettes by changing volume label information

Initialize (FORMAT and FORMAT2)

To initialize a diskette, $INIT formats the diskette as a basic data exchange diskette with twenty-six 128-byte sectors per data track or as an extended format diskette with eight 512-byte sectors per data track (see the description of FORMAT2 under index entry: *INIT command statement*).

During the initialization process, the diskette is checked for active files. If one or more active files exist on the diskette, the operator is notified via the display screen to cancel the job or continue. If the operator continues the job, active files are deleted and the remainder of the diskette is checked for defective tracks. If no active files are found on the diskette, $INIT checks for defective tracks, marking (flagging) any defective tracks found.

If track 0 or more than two other tracks are found to be defective, the operator is notified via the display screen and initialization is terminated by $INIT. Otherwise, if one or two bad tracks are found, their addresses are preserved in the ERMAP field on track 0 (see *The IBM Diskette General Information Manual,* GA21-9182).

To complete initialization, all data sectors (tracks 1-74) on the diskette are written with 128-byte or 512-byte sectors consisting of blanks. The vol-id and owner-id fields in the volume label on track 0 are replaced by the PACK and ID parameter values, respectively (the parameters are described with the other $INIT parameters). If the PACK parameter is not specified, the system date is used. If the ID parameter is not specified, OWNERID is used.

The VTOC is initialized to indicate that one file, DATA, occupies tracks 1-73, and DATA is empty.

Notes:

1. All diskettes for a multivolume file must be initialized in the same format; all diskettes in the file must be in the 128-bytes-per-sector basic data exchange format or the 512-bytes per sector extended format.
2. If a diskette read error occurs on a 512-bytes-per-sector diskette, you cannot correct the bad sector. You can either rerun the job using a different diskette or retry the same diskette.

Delete (DELETE)

If the DELETE option of $INIT is requested, the operator is notified via the display screen when any active files exist on the inserted diskette. If active files do exist, the job can be canceled or the files can be deleted. If the DELETE option is taken, the VTOC for the diskette is set to indicate that one file, DATA, occupies tracks 1-73, and DATA is empty. The vol-id specified with the DELETE option is compared with the vol-id in the diskette volume label on track 0. They must be identical for deletion to occur. The owner-id information specified with the DELETE option is not compared to information in the volume label.

Rename (RENAME)

If the RENAME option is chosen instead of FORMAT, FORMAT2, or DELETE, only the volume label (track 0) is changed. The vol-id and owner-id fields are replaced by the contents of the PACK and ID parameters, respectively. These parameters are specified with the RENAME option. If a new vol-id is not specified, the system date is used. If owner-id is not specified, OWNERID is used.

Diskette Defects Encountered During Processing

If the system encounters diskettes with physical defects during output operation, the following information will apply.

If a defect is discovered while a job is being processed, the system will make one or more attempts (called retries) to read or write the bad sector. If the retries are not successful and the program is creating output to diskette, the file is closed at the beginning of the operation during which the error occurred, and normally at the start of a track. The operator is notified that the diskette contains a defect and is given the option of inserting another diskette and continuing the operation (which will result in a multivolume file) or terminating the job and restarting with an error-free diskette.

To restore to full use, the diskette should be initialized; however, if the initialization process results in discovery of more than two defective tracks, the diskette is unusable.

$INIT Utility Control Statement Formats

The utility control statement for $INIT functions must appear in the order shown:

Use	Control Statement

Initialize a diskette	// UIN OPTION-{FORMAT / FORMAT2} [,RECL-{number / 080}]
	[// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID]
	// END]

Delete files on a diskette	// UIN OPTION-DELETE [,RECL-{number / 080}]
	[// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID]
	// END]

Rename a diskette	[// UIN OPTION-RENAME [,RECL-{number / 080}]] [// VOL [PACK-vol-id / system date] [,ID-owner-id / ,OWNERID] // END]

$INIT Parameters

UIN Statement

The UIN statement specifies which $INIT option is selected and the record length the header labels contain.

OPTION-{FORMAT / FORMAT2}	Initializes a diskette as a basic data exchange format diskette (FORMAT) with 128-byte data sectors or as an extended format diskette with 512-byte data sectors (FORMAT2). For more details on FORMAT and FORMAT2, see index entry: *INIT command statement.*
OPTION-DELETE	Deletes files on a diskette.
OPTION-RENAME	Renames a diskette. RENAME is the option selected if no option is specified.
RECL-{number / 080}	Specifies the record length to be inserted into the header labels (HDR1 and DDR1). 080 is the default.

VOL Statement

The VOL statement provides information to be written in the volume label.

PACK-vol-id system date	The PACK parameter specifies the vol-id. If the PACK parameter is not used, the system date is the default.
ID-owner-id OWNERID	The ID parameter specifies information for the owner-id field of the volume label. If the ID parameter is not used, OWNERID is the default.

$INIT Parameter Summary

OPTION- {FORMAT / FORMAT2}

If FORMAT or FORMAT2 is specified, the initialization function of $INIT is selected. The initialization function of $INIT formats and tests a diskette. Track 0 is built or rebuilt and tested for defects. If any defects are found on track 0, the diskette is unusable.

Tracks are tested through attempts to write IDs and records consisting of 128 bytes (FORMAT) or 512 bytes (FORMAT2) of hex E5 on the tracks. If a defect is found within a track, the entire track is marked defective (track IDs all hex FF) and an alternate is assigned. If more than two bad tracks are found, the job is terminated and the diskette is not usable.

Tracks are initialized by writing sectors of blanks on all data tracks (1-74). During initialization the VTOC is set to indicate that one file, DATA, occupies tracks 1–73, and DATA is empty. The vol-id and owner-id fields of the volume label (track 0) are replaced by the vol-id and owner-id given in the PACK and ID parameters, respectively (see following). If neither parameter is used, the system date and OWNERID are written in the vol-id and owner-id fields, respectively.

For more details on FORMAT and FORMAT2, see index entry: *INIT command statement.* A description of diskette formats is in Appendix C.

OPTION-DELETE

The DELETE function deletes files from a diskette by setting the VTOC to indicate that one file, DATA, occupies the entire diskette, and DATA is empty.

OPTION-RENAME

The RENAME function places the vol-id specified in the PACK parameter (see following), or the system date, if PACK is not used, in the vol-id field of the volume label (track 0). The vol-id is left-adjusted and padded with blanks. If the ID parameter is used (see following), as many as 14 characters of owner identification information, left-adjusted and padded with blanks, are placed in the owner-id field of the volume label. If the ID parameter is not used, OWNERID is placed in the owner identification field.

RECL- {number / 080}

Specifies the record length contained in the HDR1 and DDR1 header labels. 080 is the default. The maximum number of 128-byte sectors (FORMAT) is 128. The maximum number for 512-byte sectors (FORMAT2) is 512. This is also true when using OPTION-DELETE. When using OPTION-RENAME, RECL is ignored. If RECL-080 is entered or RECL is allowed to default, the record length is inserted into the header labels as 00080. Otherwise the record length is inserted as 00nnn, where nnn is the number entered.

PACK-vol-id
system date

During initialization (FORMAT or FORMAT2) $INIT writes the vol-id specified by the PACK parameter in the volume label of the diskette being initialized. The vol-id can be as many as six alphameric characters. The system date is written if the PACK parameter is not specified.

In the DELETE function, the vol-id specified or system date must be equal to the vol-id existing on the inserted diskette, or the DELETE function is not performed.

In the RENAME function, the vol-id specified or the system date is written in the volume label of the inserted diskette.

ID-owner-id
OWNERID

The ID parameter specifies owner information to be written in the volume label to further identify a diskette. As many as 14 characters can be specified. Any combination of characters except single quotation marks ('), commas, and leading or embedded blanks can be specified. If the ID parameter is not used, OWNERID is written in the owner-id field of the volume label.

Owner identification information is strictly for the user. It is not used by the system to verify that the appropriate diskette is being used for a job.

CAUTION

When a program that allows an inquiry request is interrupted, the execution of that program is suspended, permitting the execution of other programs. However, if these other programs alter the status of the system or the status of files, the effect may be abnormal termination of the interrupted program or erroneous results when the interrupted program regains control. If you are using inquiry, *do not* change any files that were being used by the interrupted (rolled-out) program. System/32 system control programming does not always check for duplicate file labels in the inquiry and interrupted programs. For example, program X is interrupted while it is processing file A. Records in file A are then deleted using inquiry. A return to program X will cause unpredictable results.

The system and disk oriented functions listed below have the potential for such abnormal termination and erroneous results when executed in an inquiry mode:

- An inquiry request cannot be used to execute the following utilities:

Utility	Function(s)
$BACK	Back up library
$LOAD	Reload library
$MAINT	All except LISTLIBR and FROMLIBR
$PACK	Compress file space
$REBLD	Rebuild VTOC
$SETCF	Reconfigure system
$BUILD	Rebuild alternate sector

- An inquiry request cannot be used to run the following utilities to perform the listed functions:

Utility	Function(s)
$COPY	Restore all/save all files
$DELET	Delete all files

- An inquiry request cannot be used to run the following utilities to process active files:

Utility	Function(s)
$BICR	Transfer active file
$COPY	Save/organize active file
$DELET	Delete active file

- An inquiry request can be used to run the following utilities to perform the following functions, but a warning message will be issued when the function is requested:

Utility	Function(s)
$COPY	Display active file
$LABEL	Catalog all/active file

Offline Option

For a description of how offline multivolume files are processed, see index entry: *offline multivolume file.*

$LOAD Utility Control Statement Format

Control statements are not used.

$LOAD OCL Sequence

```
// LOAD $LOAD
// FILE NAME-#LIBRARY,UNIT-I1
// RUN
```

$MAINT—LIBRARY MAINTENANCE UTILITY PROGRAM

$MAINT is evoked by the APCHANGE, CONDENSE, FROMLIBR, JOBSTR, LISTLIBR, REMOVE, and TOLIBR procedures (see index entries: *APCHANGE procedure, CONDENSE procedure, FROMLIBR procedure, JOBSTR procedure, LISTLIBR procedure, REMOVE procedure,* and *TOLIBR procedure*). $MAINT has four major functions:

- *Allocate.* Specifies or changes the size of the library file (#LIBRARY).
- *Copy.* Copy
  1. Places library members in the library
  2. Duplicates library members within the library
  3. Copies library members to a file on the disk or on a diskette
  4. Displays or prints the contents of the library or directory
- *Delete.* Removes library members by deleting them.
- *Compress.* Removes unusable space within the library.

All functions of $MAINT and the related utility control statements are described in detail after the following general description of the library.

UNIT	F1 for disk VTOC displays.
DATE	The current system date in the current format.
FILE NAME	The name of the file described by the VTOC entry.
FILE DATE	The creation date of the file described.
RECORD COUNT	The number, in decimal, of records currently contained by a file. A VTOC entry for the library file, #LIBRARY, is shown in the preceding sample display. Because #LIBRARY is not a data file, no record count is given.
RECORDS AVAILABLE	The number, in decimal, of records for which there is still room in a file. A VTOC entry for the library file, #LIBRARY, is shown in the preceding sample display. Because #LIBRARY is not a data file, no record count is given.
RETAIN TYPE	The retention classification of a file: P for permanent, T for temporary.
FILE ORG	File organization: S for sequential, I for indexed sequential, D for direct, P for pseudo tape.
RECORD LENGTH	The length, in decimal number of bytes, of the records in a file. A VTOC entry for the library file, #LIBRARY, is shown in the preceding sample display. Because #LIBRARY is not a data file, the record length given in the sample display is 00.
FILE LOCATION	The decimal block number of the beginning of data in a file.
KEY LENGTH	Decimal length of the keys in an indexed file.
KEY LOCATION	The position, in decimal, of the rightmost byte of the key in the records in an indexed file.
CREATION FORMAT	The format in which a file was created, either BLOCKS or RECORDS.
UNITS ALLOCATED	The number, in decimal, of blocks or records allocated for a file. *Note:* If the RECORDS parameter was used to allocate space, the number shown might be greater than the number requested, because the system allocates space in blocks and rounds up to the next higher block whenever part of a block is required.

Diskette VTOC

```
DISKETTE DISPLAY DATE - (DATE)
PACK - INVTRY      ID - JONES

SPACE AVAILABLE ON THIS PACK IS 1861 BLOCKS - EACH 128 BYTES

  FILE     FILE      FILE     FILE     RECORD      FILE     EXPIRATION     MVF     SEQUENCE    NUMBER OF BLOCKS
  NAME     DATE     LENGTH    TYPE     LENGTH    LOCATION     DATE        FILE     NUMBER    IN OFFLINE MV FILE

KRCINV   01/17/75     30     SYSTEM      37          27       PROTECT
TEMPRM   01/24/75     33     SYSTEM      20          57       PROTECT
*****END OF VTOC DISPLAY*****
```

DATE — The current system date in the current format.

PACK — The volume identification given in the diskette volume label. The volume label for 128-bytes per sector basic data exchange format diskettes is described in the *IBM Diskette General Information Manual,* GA21-9182. See also Appendix C of this manual.

ID — The owner identification given in the diskette volume label. The volume label for 128-bytes per sector basic data exchange format diskettes is described in the *IBM Diskette General Information Manual,* GA21-9182. See also Appendix C.

SPACE AVAILABLE — The number, in decimal, of 128-byte or 512-byte sectors available on the diskette. The number represents space following the last active file on the diskette. If all files were removed, using the DELETE procedure or the $DELET utility program, the heading THERE ARE NO FILES PRESENT ON THIS PACK appears and the VTOC display ends.

FILE NAME — The name specified when the file described was created.

FILE DATE — The creation date of the file described.

FILE LENGTH — The number, in decimal, of 128-byte or 512-byte sectors contained in a file.

Note: If the file was created by the $COPY utility (see index entry: *$COPY utility program*) and the record length of the file is 128 bytes or less, a sector of control information is inserted at the beginning of the file. This sector of control information increases the FILE LENGTH by one. When the file is returned to the disk, the control information is dropped and record count returns to the original number.

Allocate Function

Allocate Uses

- Specify library size
- Increase library size
- Decrease library size

Allocate Control Statement Formats

Use	Control Statement
Specify library size	// ALLOCATE LIBRSIZE-number
Increase library size	// ALLOCATE INCREASE-number
Decrease library size	// ALLOCATE DECREASE-number

Allocate Parameters

LIBRSIZE-number	Specifies the size of the library in number of blocks (1 block = ten 256-byte sectors)
INCREASE-number	Increases the library size by the number of blocks indicated
DECREASE-number	Decreases the library size by the number of blocks indicated

Allocate OCL and Utility Control Statement Sequence

```
// LOAD $MAINT
// RUN
// ALLOCATE...
// END
```

Note: Within any one run of the $MAINT utility program (that is, for any one // LOAD $MAINT and // RUN sequence), you cannot increase, then decrease, then increase the library size, whether you use the INCREASE and DECREASE keywords or the LIBRSIZE keyword to change the library size.

Allocate Examples

In order to set the library size at 1000 blocks you would enter:

```
// LOAD $MAINT
// RUN
// ALLOCATE LIBRSIZE-1000
// END
```

In order to increase the library size by 10 blocks you would enter:

```
// LOAD $MAINT
// RUN
// ALLOCATE INCREASE-10
// END
```

In order to decrease the library size by 3 blocks you would enter:

```
// LOAD $MAINT
// RUN
// ALLOCATE DECREASE-3
// END
```

Copy Function

Copy Uses

Reader-to-Library	• Add a procedure or source member to the library, or replace a procedure or source member in the library. When $MAINT OCL statements are entered into the system from the keyboard, reader refers to the keyboard. If a procedure calls $MAINT copy function, reader refers to the procedure itself. The member to be copied to the library is then all statements following the COPY statement until a CEND statement is encountered.
Library-to-Library	• Copy a member from the library to the library, changing the name of the member. • Copy a member having a certain name or all members having the name. • Copy members, either of a specified type or of all types, that have names beginning with certain characters. • Copy members, either of a specified type or of all types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members (SCP members are library members of any type, provided with and used by the SCP).
Library-to-File (record mode or sector mode)	• Copy a member from the library to a file. • Copy a member having a certain name or all members having the name.

Note: The $LOAD utility provides the only method whereby you can change the size of the library directory (alter the space allocated to it) and the size of the HISTORY file. When using the $LOAD utility program (or the RELOAD procedure) however, be aware that library members that exist on the disk but do not exist in the backed up library will be lost when the backed up library is returned to the disk. If you have added library members to the disk since the library was backed up and you want to save those members, use the FROMLIBR procedure to copy them before executing RELOAD or $LOAD, then use TOLIBR to place them in the backed up library after it is reloaded. (You might have to increase the size of the backed up library in order to have room for the additional members.) For information on FROMLIBR and TOLIBR, see index entries: *FROMLIBR procedure* and *TOLIBR procedure.*

Inquiry Option

Certain programs can be interrupted while they are being processed. A request for interruption is called an *inquiry request* (made by pressing the INQ key on the keyboard and choosing the 1 option). Programs are usually interrupted to permit another program to run. Control is then returned to the first program.

The inquiry interrupt involves three steps:

1. When a program that can be interrupted recognizes an inquiry request (the INQ key was pressed and the 1 option chosen), a rollout routine moves the interrupted program from main storage to the disk.

   Note: If an inquiry request is made during execution of certain system programs, the inquiry display is delayed until the system programs are completed. When the system programs are completed, the inquiry display appears.

2. The program for which the interrupt was requested must be loaded normally using the keyboard. The interrupting program can be any type. This interrupting program cannot be interrupted, but can be canceled.

   Note: The printer does not skip to line 1 of the next page at the end of an interrupting program.

3. After the interrupting program is executed, a roll in routine moves the interrupted program back into main storage. The interrupted program begins execution at the point of interruption and terminates in a normal manner.

If the inquiry option is selected, the SCP allocates a rollout area on the disk to contain programs that can be rolled out from main storage. If the inquiry option is not selected, the inquiry interrupt itself is allowed, but attempts to perform the rollout routine are bypassed.

Notes:
1. If the operator presses the INQ key after pressing the STOP key, and the IPL diskette switch is on, the system displays the contents of main storage on the display screen, beginning with the main storage address specified by the data switches on the CE control panel. To terminate the display and continue processing, the operator can press the START key.
2. The inquiry option is inactive if data communications or the data recorder is being used.

CAUTION
When a program that allows an inquiry request is interrupted, the execution of that program is suspended, permitting the execution of other programs. However, if these other programs alter the status of the system or the status of files, the effect may be abnormal termination of the interrupted program or erroneous results when the interrupted program regains control. If you are using inquiry, *do not* change any files that were being used by the interrupted (rolled-out) program. System/32 system control programming does not always check for duplicate file labels in the inquiry and interrupted programs. For example, program X is interrupted while it is processing file A. Records in file A are then deleted using inquiry. A return to program X will cause unpredictable results.

The system and disk oriented functions listed below have the potential for such abnormal termination and erroneous results when executed in an inquiry mode:

- An inquiry request cannot be used to execute the following utilities:

Utility	Function(s)
$BACK	Back up library
$LOAD	Reload library
$MAINT	All except LISTLIBR and FROMLIBR
$PACK	Compress file space
$REBLD	Rebuild VTOC
$SETCF	Reconfigure system
$BUILD	Rebuild alternate sector

- An inquiry request cannot be used to run the following utilities to perform the listed functions:

Utility	Function(s)
$COPY	Restore all/save all files
$DELET	Delete all files

- An inquiry request cannot be used to run the following utilities to process active files:

Utility	Function(s)
$BICR	Transfer active file
$COPY	Save/organize active file
$DELET	Delete active file

- An inquiry request can be used to run the following utilities to perform the following functions, but a warning message will be issued when the function is requested:

Utility	Function(s)
$COPY	Display active file
$LABEL	Catalog all/active file

Offline Option

For a description of how offline multivolume files are processed, see index entry: *offline multivolume file.*

$LOAD Utility Control Statement Format

Control statements are not used.

$LOAD OCL Sequence

```
// LOAD $LOAD
// FILE NAME-#LIBRARY,UNIT-I1
// RUN
```

$MAINT—LIBRARY MAINTENANCE UTILITY PROGRAM

$MAINT is evoked by the APCHANGE, CONDENSE, FROMLIBR, JOBSTR, LISTLIBR, REMOVE, and TOLIBR procedures (see index entries: *APCHANGE procedure, CONDENSE procedure, FROMLIBR procedure, JOBSTR procedure, LISTLIBR procedure, REMOVE procedure,* and *TOLIBR procedure*). $MAINT has four major functions:

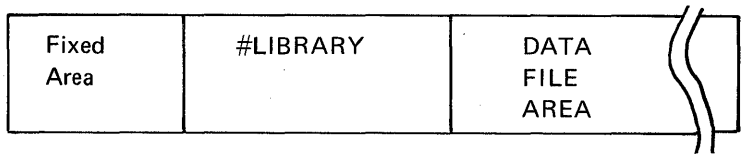
- *Allocate.* Specifies or changes the size of the library file (#LIBRARY).
- *Copy.* Copy
  1. Places library members in the library
  2. Duplicates library members within the library
  3. Copies library members to a file on the disk or on a diskette
  4. Displays or prints the contents of the library or directory
- *Delete.* Removes library members by deleting them.
- *Compress.* Removes unusable space within the library.

All functions of $MAINT and the related utility control statements are described in detail after the following general description of the library.

System Library File (#LIBRARY)

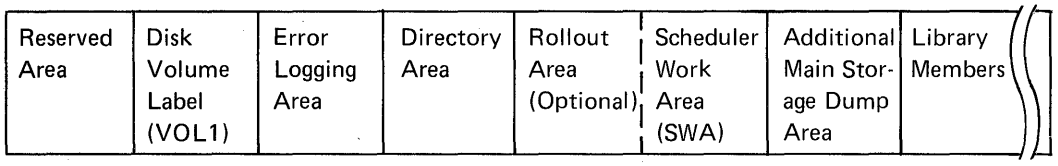
Location

The library is the first file on the disk immediately following the reserved fixed area of the disk.



The boundaries of the library at any one time are fixed, though they can be changed by a BACKUP and RELOAD sequence (see index entries: *BACKUP procedure* and *RELOAD procedure*), and by ALLOCATE, a function of $MAINT (see index entry: *ALLOCATE*).

Contents



Disk volume label (VOL1): The volume label is 256 bytes long and contains owner identification information and system control programming information regarding the disk.

Error Logging Area: The error logging area is a variable number of sectors used for recording hardware and hardware-related system errors. The error logging area is assigned by the $LOAD utility.

Directory area: The directory area contains system information, recorded and maintained by $MAINT, and the library directory. The library directory contains an entry for each member in the library. Each entry describes the corresponding library member and identifies its location. $MAINT places an entry in the directory each time it places a member in the library, and deletes an entry each time it deletes a member. The size of the library directory can be changed by the $LOAD utility (see index entry: *$LOAD utility program*). However, the size of the directory is restricted to a maximum of 256 sectors.

Rollout area: A rollout area is allocated only if inquiry support and offline multivolume file support is selected (INCLUDE INQUIRY/OFFLINE? = YES on the RELOAD display—see index entry: *RELOAD display*). For a description of the inquiry option and offline multivolume files, see index entries: *inquiry option* and *offline multivolume file.*

Scheduler work area (SWA): The SWA is a 170-sector area reserved for use by components of the system control program.

Additional main storage dump area: This area is set aside for the 24K and 32K (K = 1024 bytes) main storage systems.

Library members: The library can contain load members, procedure members, source members, and subroutine members. Member names can be any combination of characters (numeric, alphabetic, and special) except commas, periods, single quotes ('), blanks, question marks (?), slash (/), and hyphen (-). The question mark (?), slash (/), and hyphen (-) have special meanings in procedures (see index entry: *procedure parameters*) and in certain control statements and should not be used in member names. The first character of a member name must be alphabetic (includes #, $, and @), and the number of characters in a member name must not exceed eight.

Organization of Library Members within the Library

Members are stored in the library serially; that is, a 20-sector member occupies 20 consecutive sectors.

New library members are placed in the library just as records are placed in an indexed file; that is, they are placed after the last active member, and their physical order in the library reflects the sequence in which they were entered.

When members are deleted from the library, only the space after the last active member is made available for new members. This means that if you copy a group of new members into the library, then later delete these same members before adding more, the space they occupied in the library is unavailable for adding other new members.

Gaps can occur in the library either when a member is deleted or when a member is replaced by a member that requires a different number of sectors, because sectors between members are unusable. If the number of unusable sectors becomes high, the library member area can be compressed by either the CONDENSE procedure, a BACKUP and a RELOAD sequence, or the APCHANGE procedure.

When a library is compressed, members are moved as close as possible to the beginning of the library member area so that no gaps are between the members. All unused space is collected into one free area at the end of the library.

To provide as much space as possible within the prescribed limits of the library, the system compresses procedure and source members by removing all duplicate blanks. When the members are retrieved, the blanks are reinserted.

Allocate Function

Allocate Uses

- Specify library size
- Increase library size
- Decrease library size

Allocate Control Statement Formats

Use	Control Statement
Specify library size	// ALLOCATE LIBRSIZE-number
Increase library size	// ALLOCATE INCREASE-number
Decrease library size	// ALLOCATE DECREASE-number

Allocate Parameters

LIBRSIZE-number	Specifies the size of the library in number of blocks (1 block = ten 256-byte sectors)
INCREASE-number	Increases the library size by the number of blocks indicated
DECREASE-number	Decreases the library size by the number of blocks indicated

Allocate OCL and Utility Control Statement Sequence

```
// LOAD $MAINT
// RUN
// ALLOCATE...
// END
```

Note: Within any one run of the $MAINT utility program (that is, for any one // LOAD $MAINT and // RUN sequence), you cannot increase, then decrease, then increase the library size, whether you use the INCREASE and DECREASE keywords or the LIBRSIZE keyword to change the library size.

Procedure Restrictions

If nested procedures are used, information contained in the scheduler work area can become invalid when a source library is reorganized or changed in size. Therefore, if a procedure is used to reallocate or reorganize libraries, any further procedures contained within that nested procedure should not be called from the source library that is being reallocated or reorganized.

- Print members, either of a specified type or of all types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R|ALL},NAME-{name|characters.ALL}

TO-PRINT,OMIT-{name|characters.ALL|SYSTEM}
```

- Print the directory entries for members of a certain type:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R},NAME-DIR,TO-PRINT
```

- Print all directory entries and system information from the directory area:

```
// COPY FROM-F1,LIBRARY-ALL,NAME-DIR,TO-PRINT
```

- Print the system information in the directory area:

```
// COPY FROM-F1,LIBRARY-SYSTEM,NAME-DIR,TO-PRINT
```

- Print directory entries, either for members of a specified type or for all members, omitting entries for members that have a certain name or names beginning with certain characters, or omitting all entries for SCP members:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R|ALL},NAME-DIR,TO-PRINT,

OMIT-{name|characters.ALL|SYSTEM}
```

Note: If the display screen and not the printer is used to list library members or directory entries, only the first 40 bytes of each output line are displayed. To ensure that all the information in a library member or directory entry is listed, assign the printer to list the output. You can use the STATUS procedure (see index entry: ***STATUS procedure***) to determine where system output is currently listed (that is, what the current SYSLIST assignment is); you can use the SYSLIST procedure (see index entry: ***SYSLIST procedure***) to change the current SYSLIST assignment.

Copy Parameters

FROM-READER	The member to be placed in the library is to be read (entered) from the keyboard or from a procedure that invokes the $MAINT copy function.
FROM-F1	Library members are located in the library.
FROM-DISK	Input is from a file on either the disk or a diskette. A FILE statement is required to identify the file.
LIBRARY-S	The specified member(s) is a source member.
LIBRARY-P	The specified member(s) is a procedure member.
LIBRARY-O	The specified member(s) is a load member.
LIBRARY-R	The specified member(s) is a subroutine member.
LIBRARY-ALL	• For copying library-to-file in record mode, specifies source (S) and procedure (P) members. • For printing from the directory area, specifies that system information as well as directory entries are to be printed. *Note:* For all uses of copy except the two just listed, specifies that all member types (S, P, O, and R) are involved.

- Print directory entries, either for members of a specified type or for all members, omitting entries for members that have a certain name or have names beginning with certain characters, or omitting all entries for SCP members.

Note: If the display screen and not the printer is used to list library members or directory entries, only the first 40 bytes of each output line are displayed. To ensure that all the information in a library member or directory entry is listed, assign the printer to list the output. You can use the STATUS procedure (see index entry: *STATUS procedure)* to determine where system output is currently listed (that is, what the current SYSLIST assignment is); and the SYSLIST procedure (see index entry: *SYSLIST procedure)* to change the current SYSLIST assignment.

Copy Control Statement Formats

Reader-to-Library: Control statements required for adding or replacing a procedure or source member are:

// COPY FROM-READER,LIBRARY-{P|S},NAME-name,TO-F1 [,RETAIN-P|,RETAIN-R] ,RECL-number

Library member (blanks are removed from statements before the statements are put in the library, and reinserted for printing)

// CEND Must always follow the procedure or source statements being placed in the library.

Library-to-Library: Control statements depend on the function required.

- Copy a member from the library to the library, changing the name of the member:

// COPY FROM-F1,LIBRARY-{S|P|O|R},NAME-name,TO-F1,NEWNAME-name [,RETAIN-P|,RETAIN-R]

- Copy a member having a certain name or all members having the name:

// COPY FROM-F1,LIBRARY-{S|P|O|R|ALL},NAME-name,TO-F1 [,RETAIN-P|,RETAIN-R] ,NEWNAME-name

- Copy members, either of a specified type or of all types, that have names beginning with certain characters:

```
// COPY FROM-F1,LIBRARY-{S}
                        {P}
                        {O},NAME-characters.ALL,
                        {R}
                        {ALL}
TO-F1 [,RETAIN-P] ,NEWNAME-characters
      [,RETAIN-R]
```

- Copy members, either of a specified type or of all types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members:

```
// COPY FROM-F1,LIBRARY-{S}                  {name           }
                        {P}                  {characters.ALL},
                        {O},NAME-
                        {R}
                        {ALL}
TO-F1 [,RETAIN-P] ,NEWNAME-characters,OMIT-{name           }
      [,RETAIN-R]                         {characters.ALL}
                                          {SYSTEM         }
```

Library-to-File, Record Mode: Control statements depend on the function required.

Note: Source or procedure members (record mode) copied from the library to a file must be converted to a basic data exchange diskette file by the TRANSFER procedure if the diskette file is to be used as input to other systems.

- Copy or add a library member to a file:

```
// COPY FROM-F1,TO-DISK,FILE-filename {,RECL-number},NAME-name,
                                      {,ADD-YES   }
LIBRARY-{S}
        {P}
```

- Copy or add a member having a certain name or both members—two permitted types—having the name:

```
// COPY FROM-F1,TO-DISK,FILE-filename {,RECL-number},NAME-name,
                                      {,ADD-YES   }
LIBRARY-{S  }
        {P  }
        {ALL}
```

- Copy or add members, either of a specified type or of all (two) permitted types, that have names beginning with certain characters:

```
// COPY FROM-F1,TO-DISK,FILE-filename {,RECL-number} ,NAME-characters.ALL,
                                      {,ADD-YES   }
LIBRARY-{S  }
        {P  }
        {ALL}
```

- Copy or add all members of the two permitted types except SCP members:

```
// COPY FROM-F1,TO-DISK,FILE-filename {,RECL-number} ,NAME-ALL,
                                      {,ADD-YES   }
LIBRARY-ALL
```

- Copy or add members, either of a specified type or of all (two) permitted types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members:

```
// COPY FROM-F1,TO-DISK,FILE-filename {,RECL-number} ,NAME- {name           }
                                      {,ADD-YES   }       {characters.ALL}
LIBRARY-{S  } ,OMIT- {name           }
        {P  }        {characters.ALL}
        {ALL}        {SYSTEM         }
```

Library-to-File, Sector Mode: Control statements depend on the function required.

- Copy or add a library member to a file:

```
// COPY FROM-F1,TO-DISK,FILE-filename,NAME-name,

LIBRARY- {S} [,ADD-YES]
         {P}
         {O}
         {R}
```

- Copy or add a member having a certain name or all members having the name:

```
// COPY FROM-F1,TO-DISK,FILE-filename,NAME-name,

LIBRARY-{S  } [,ADD-YES]
        {P  }
        {O  }
        {R  }
        {ALL}
```

- Copy or add members, either of a specified type or of all types, that have names beginning with certain characters:

```
// COPY FROM-F1,TO-DISK,FILE-filename,NAME-characters.ALL,
LIBRARY-{S|P|O|R|ALL}[,ADD-YES]
```

- Copy or add all members except SCP members:

```
// COPY FROM-F1,TO-DISK,FILE-filename,NAME-ALL,
LIBRARY-ALL [,ADD-YES]
```

- Copy or add members, either of a specified type or of all types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members:

```
// COPY FROM-F1,TO-DISK,FILE-filename,
NAME-{ALL|name|characters.ALL} ,LIBRARY-{S|P|O|R|ALL},OMIT-{name|characters.ALL|SYSTEM}[,ADD-YES]
```

- Copy or add all members (SCP and non-SCP) that have had a PTF applied to them, with the option to omit specified members or all SCP members:

```
// COPY FROM-F1,TO-DISK,FILE-filename,NAME-{ALL|name|characters.ALL} ,
LIBRARY-{S|P|O|R|ALL} ,PTF-YES [,OMIT-{name|characters.ALL|SYSTEM}] [,ADD-YES]
```

File-to-Library: Control statements depend on the function required.

- Copy a member or members from a file to the library:

```
// COPY FROM-DISK,TO-F1 [,RETAIN-P|,RETAIN-R] ,FILE-filename
```

- Copy a member that has had a particular PTF applied to it, from a file to the library:

```
// COPY FROM-DISK,TO-F1 [,RETAIN-P / ,RETAIN-R] ,FILE-filename,PTF-number
```

- Copy a member or members from a file to the library, omitting members that do not presently exist in the library.

```
// COPY FROM-DISK,TO-F1 [,RETAIN-P / ,RETAIN-R] ,FILE-filename,OMIT-NEW
```

Note: $MAINT can copy a sector mode file to the library only if the file was copied from the library by $MAINT. See the preceding description, *Library-to-file, Sector Mode* and index entry: *TOLIBR procedure.*

Library-to-Printer: Control statements depend on the function required.

- Print a member having a certain name or all members having the name:

```
// COPY FROM-F1,LIBRARY-{S/P/O/R/ALL} ,NAME-name,TO-PRINT
```

- Print all members of a certain type:

```
// COPY FROM-F1,LIBRARY-{S/P/O/R} ,NAME-ALL,TO-PRINT
```

- Print members, either of a specified type or of all types, that have names beginning with certain characters:

```
// COPY FROM-F1,LIBRARY-{S/P/O/R/ALL} ,NAME-characters.ALL,TO-PRINT
```

- Print members, either of a specified type or of all types, omitting members that have a certain name or have names beginning with certain characters, or omitting all SCP members:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R|ALL},NAME-{name|characters.ALL},
TO-PRINT,OMIT-{name|characters.ALL|SYSTEM}
```

- Print the directory entries for members of a certain type:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R},NAME-DIR,TO-PRINT
```

- Print all directory entries and system information from the directory area:

```
// COPY FROM-F1,LIBRARY-ALL,NAME-DIR,TO-PRINT
```

- Print the system information in the directory area:

```
// COPY FROM-F1,LIBRARY-SYSTEM,NAME-DIR,TO-PRINT
```

- Print directory entries, either for members of a specified type or for all members, omitting entries for members that have a certain name or names beginning with certain characters, or omitting all entries for SCP members:

```
// COPY FROM-F1,LIBRARY-{S|P|O|R|ALL},NAME-DIR,TO-PRINT,
OMIT-{name|characters.ALL|SYSTEM}
```

Note: If the display screen and not the printer is used to list library members or directory entries, only the first 40 bytes of each output line are displayed. To ensure that all the information in a library member or directory entry is listed, assign the printer to list the output. You can use the STATUS procedure (see index entry: *STATUS procedure*) to determine where system output is currently listed (that is, what the current SYSLIST assignment is); you can use the SYSLIST procedure (see index entry: *SYSLIST procedure*) to change the current SYSLIST assignment.

6 A program temporary fix (PTF) has been applied to this program.

7 This is a load member containing overlays.

Byte 1:

0 Reserved.

1 Reserved.

2 This program reads source itself. The member can contain a COMPILE statement (see index entry: *// COMPILE statement*) and a no-source-required attribute (bit 4 of byte 0 off—0).

3 This program requires that $WORK2 be allocated.

4 This SCP member has been translated from English into another language.

5 This program requires that a new load address be calculated at load time to ensure that it is placed in main storage at a point beyond its own common region.

6 This program reads utility control statements.

7 This program contains a where-to-go table. It is used by the transient cross reference resolver program (#OXRF).

LINK ADDR — For load members only. The main storage address, in decimal and hexadecimal, assigned to the member when it is linked in main storage with other load members.

RLD DISP — For load members only. Displacement, in decimal and hexadecimal, of first RLD (relocation dictionary) in member in first sector containing RLDs.

ENTRY ADDR — For load members only. Main storage address, of entry point of member in decimal and hexadecimal.

PROG SIZE — For load members only. Decimal and hexadecimal number of sectors required to run the program contained in the member.

LEVEL

- Release level of the system programs.
- For user's programs, this can be assigned by the overlay linkage editor.
- For source and procedure members, the release level when the members were created.
- RPG load members have a level number or zero.

Copy Examples

Library-to-Library: The following is an example of a library-to-library copy.

Copy a load member presently named ACCT in order to give it a new name, ACCT1:

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,LIBRARY-O,NAME-ACCT,TO-F1,NEWNAME-ACCT1
// END
```

Library-to-File: The following examples demonstrate copying from the library to a file.

Copy a procedure member named PAYROLL in sector mode (hexadecimal format, compressed data) to a disk file named PAY that is 30 sectors long and is to be retained permanently:

```
// LOAD $MAINT
// FILE NAME-PAY,UNIT-F1,BLOCKS-3,RETAIN-P
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAY,NAME-PAYROLL,LIBRARY-P
// END
```

Copy a source member named SAM in record mode (expanded format, includes blanks) with a record length of 80 to a disk file named BOB that is 20 sectors long and is to be retained only temporarily:

```
// LOAD $MAINT
// FILE NAME-BOB,UNIT-F1,BLOCKS-2,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-BOB,RECL-80,NAME-SAM,LIBRARY-S
// END
```

Copy a source member or procedure member named SAM in record mode (expanded format, includes blanks) with a record length of 80 to a disk file named BOB that is 20 sectors long and is to be retained only temporarily. Then using the TRANSFER procedure, convert the disk file named BOB to a basic data exchange diskette file named BOB, on a diskette with a vol-id of 111222 and a retention period of 30 days:

```
// LOAD $MAINT
// FILE NAME-BOB,UNIT-F1,BLOCKS-2,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-BOB,RECL-80,NAME-SAM,LIBRARY-S
// END
TRANSFER BOB,F1,751215,111222,30
```

Notes:

1. The date format on the TRANSFER procedure must be in yymmdd format if you are creating basic data exchange diskette files to use with other systems.
2. Source or procedure members (record mode) copied from the library to a file must be converted to a basic data exchange diskette file by the TRANSFER procedure if the diskette file is to be used as input to other systems.

Record Mode: Record mode is specified by the RECL parameter used with the TO-DISK parameter (each is described in a preceding paragraph). Record mode can be specified only for source and procedure members. Source and procedure member copies made in record mode are preceded by a // COPY utility control statement and followed by a // CEND utility control statement. (The format of the // COPY utility control statement is:

// COPY NAME-name,LIBRARY-$\left\{\begin{matrix}P\\S\end{matrix}\right\}$ where name is

the member name and P or S indicates procedure or source member. The format of the // CEND utility control statement is // CEND.) The member itself is in expanded format; that is, the data is not compressed—all blanks are included.

Sector Mode: The TO-DISK parameter without the RECL-number parameter specifies sector mode. A sector mode copy can be specified for any type (load, procedure, source, or subroutine) of library member. In sector mode, copies are in hex format and consist of control information and PTF (program temporary fix) numbers for any PTFs that have been applied to a member, followed by the member as it exists in the library.

ADD-YES

Add library member(s) to an existing file that contains library members. If ADD-YES is not specified, ADD-NO is assumed.

Notes:

1. When adding a member to a disk file, the file must contain enough unused space to hold the member. When adding a member to a diskette file, the file must be the last active (unexpired) file on the diskette.
2. The RECL parameter (described in preceding paragraph) is not allowed if ADD-YES is specified. The record length is determined by the record length of the existing file. The record length of the existing file also determines whether a member is added in record or sector mode. If the record length of the existing file is 40 to 120, the source or procedure member is added in record mode. If the record length of the existing file is 32, the member is added in sector mode.

NEWNAME-name

Name desired for a new member(s). Valid only for a library-to-library copy.

NEWNAME-characters

Beginning characters (maximum of seven) of the name desired for a new library member(s). Must be the same number of characters as specified in the NAME-characters.ALL parameter described in a preceding paragraph.

OMIT-name

Omit the entry specified by name.

OMIT-characters.ALL

Omit all entries whose names begin with the specified characters (maximum of seven characters).

OMIT-SYSTEM	Omit all SCP members.
OMIT-NEW	Omit copying all members that do not presently exist in the library.
FILE-filename	The name of the file given on the OCL FILE statement referring to the input or output file.
PTF-YES	Specifies that only members that have had a PTF applied to them are to be copied to the file. Valid only when copying in sector mode from the library to a file: FROM-F1,TO-DISK.
PTF-NO	The members that have had a PTF applied to them have no particular significance. They are copied if the name is the same as the specified name. If the PTF parameter is not specified, the default value is PTF-NO.
PTF-number	Only the member(s) with the specified PTF log number (00001 through 65535) are selected from the file and copied to the library. The PTF keyword having a numeric value is only valid when copying in sector mode from file to library: FROM-DISK,TO-F1.

Copy OCL and Utility Control Statement Sequence

```
// LOAD $MAINT
[// FILE ...]
// RUN
// COPY ...
// END
```

A FILE statement is required only if copying TO-DISK or FROM-DISK; that is, from the library to a file, or from a file to the library.

Using the Copy Function

Naming Library Members: Considerations that apply to naming library members are:

- The first character of each member name must be alphabetic, #, $, or @. Member names can be any combination of characters (numeric, alphabetic, and special) except commas, periods, single quotes ('), blanks, question marks (?), slash (/), and hyphen (-). The question mark (?), slash (/), and hyphen (-) have special meanings in procedures (see index entry: *procedure parameters*) and in certain control statements and should not be used in member names. The names of all IBM-supplied SCP load and subroutine members begin with a pound or dollar sign (# or $). Therefore, to avoid possible duplication, do not use a pound or dollar sign as the first character in names you assign.

- A name can be from one to eight characters long.

- ALL, DIR, and SYSTEM must not be used as member names. They have special meanings in the LIBRARY, NAME, and OMIT parameters.

- Members of the same type cannot have the same name, but members of different types can. For example, two procedure members cannot have the same name, but a procedure member and a source member can have the same name.

Printing from the Library: Library members can be printed by using a library-to-printer request. When the statements are printed, blanks are reinserted into statements contained in source and procedure members. Load and subroutine members are printed in hex format.

Library-to-printer requests can also be used to print system information contained in the library directory area and to print directory entries. Figure 7 shows a sample printout of system information contained in the library directory area. Figure 8 shows the information given in a printout of a directory entry. The figure is followed by an explanation of the fields shown.

```
SYSTEM INFORMATION  01-01-75

START SECTOR OF LIBRARY                      184/00B8

END SECTOR OF LIBRARY                       5600/15E0

TOTAL NUMBER OF LIBRARY BLOCKS               542/021E

START SECTOR OF DIRECTORY                    184/00B8

END SECTOR OF DIRECTORY                      243/00F3

DIRECTORY SECTORS                             60/003C

ACTIVE DIRECTORY ENTRIES                     345/0159

AVAILABLE DIRECTORY ENTRIES                  292/0124

START SECTOR, INQUIRY/OFFLINE AREA           244/00F4

END SECTOR OF INQUIRY/OFFLINE AREA           363/016B

START SECTOR OF SCHEDULER AREA               364/016C

END SECTOR OF SCHEDULER AREA                 533/0215

START SECTOR OF LIBRARY MEMBERS              534/0216

END SECTOR OF LIBRARY MEMBERS               5600/15E0

ACTIVE LIBRARY MEMBER SECTORS               1943/0797

AVAILABLE MEMBER SECTORS                    2549/09F5

NEXT AVAILABLE MEMBER SECTOR                3052/0BEC
```

Note: The first of the two numbers given in the column at the right is a decimal number, the second is hexadecimal. (In the example 184/00B8, 184 is the decimal value of hexadecimal 00B8.)

Figure 7. Sample Printout of System Information

```
LIBRARY DIRECTORY MM DD YY
TYPE  NAME    START ADDR  TOTAL    NUM TEXT/RECORD  ATTRIBUTES  LINK ADDR  RLD DISP  ENTRY ADDR  PROG SIZE  LEVEL
X     member  dec/hex     dec/hex  dec/hex          2 bytes     dec/hex    dec/hex   dec/hex     dec/hex    n
```

Figure 8. Information in Printout of Library Directory Entry

Following is an explanation of the fields shown in Figure 8.

TYPE — Type of library member described by the entry:
- S source member
- P procedure member
- O load member
- R subroutine member

NAME — Name of the library member.

START ADDR — Sector number of the first sector of the member in both decimal and hexadecimal.

TOTAL — Total number of sectors in the member, in decimal and hexadecimal.

NUM TEXT/RECORD — For source and procedure members, record length of the member, given in decimal and hexadecimal. For load members, the number of text sectors contained in the member, excluding RLDs—*relocation dictionaries*—which are the part of a load member used for adjusting main storage addresses when the member is moved to main storage. (For subroutine members, blank.)

ATTRIBUTES — Two bytes, 16 bits, of *attributes* giving detailed characteristics of the member.

Bit	Meaning When On (1)
Byte 0:	
0	This member is an SCP member. This bit is used to prevent SCP members from being deleted or removed.
1	Reserved.
2	Reserved.
3	Reserved.
4	This program requires that $WORK and $SOURCE be allocated. $SOURCE must be filled from the keyboard, a source member, or an inline source from a procedure (queued job stream).
5	This SCP module is not part of the basic SCP system.

6 A program temporary fix (PTF) has been applied to this program.

7 This is a load member containing overlays.

Byte 1:

0 Reserved.

1 Reserved.

2 This program reads source itself. The member can contain a COMPILE statement (see index entry: *// COMPILE statement*) and a no-source-required attribute (bit 4 of byte 0 off—0).

3 This program requires that $WORK2 be allocated.

4 This SCP member has been translated from English into another language.

5 This program requires that a new load address be calculated at load time to ensure that it is placed in main storage at a point beyond its own common region.

6 This program reads utility control statements.

7 This program contains a where-to-go table. It is used by the transient cross reference resolver program (#OXRF).

LINK ADDR — For load members only. The main storage address, in decimal and hexadecimal, assigned to the member when it is linked in main storage with other load members.

RLD DISP — For load members only. Displacement, in decimal and hexadecimal, of first RLD (relocation dictionary) in member in first sector containing RLDs.

ENTRY ADDR — For load members only. Main storage address, of entry point of member in decimal and hexadecimal.

PROG SIZE — For load members only. Decimal and hexadecimal number of sectors required to run the program contained in the member.

LEVEL

- Release level of the system programs.
- For user's programs, this can be assigned by the overlay linkage editor.
- For source and procedure members, the release level when the members were created.
- RPG load members have a level number or zero.

Copy Examples

Library-to-Library: The following is an example of a library-to-library copy.

Copy a load member presently named ACCT in order to give it a new name, ACCT1:

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,LIBRARY-O,NAME-ACCT,TO-F1,NEWNAME-ACCT1
// END
```

Library-to-File: The following examples demonstrate copying from the library to a file.

Copy a procedure member named PAYROLL in sector mode (hexadecimal format, compressed data) to a disk file named PAY that is 30 sectors long and is to be retained permanently:

```
// LOAD $MAINT
// FILE NAME-PAY,UNIT-F1,BLOCKS-3,RETAIN-P
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAY,NAME-PAYROLL,LIBRARY-P
// END
```

Copy a source member named SAM in record mode (expanded format, includes blanks) with a record length of 80 to a disk file named BOB that is 20 sectors long and is to be retained only temporarily:

```
// LOAD $MAINT
// FILE NAME-BOB,UNIT-F1,BLOCKS-2,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-BOB,RECL-80,NAME-SAM,LIBRARY-S
// END
```

Copy a source member or procedure member named SAM in record mode (expanded format, includes blanks) with a record length of 80 to a disk file named BOB that is 20 sectors long and is to be retained only temporarily. Then using the TRANSFER procedure, convert the disk file named BOB to a basic data exchange diskette file named BOB, on a diskette with a vol-id of 111222 and a retention period of 30 days:

```
// LOAD $MAINT
// FILE NAME-BOB,UNIT-F1,BLOCKS-2,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-BOB,RECL-80,NAME-SAM,LIBRARY-S
// END
TRANSFER BOB,F1,751215,111222,30
```

Notes:

1. The date format on the TRANSFER procedure must be in yymmdd format if you are creating basic data exchange diskette files to use with other systems.
2. Source or procedure members (record mode) copied from the library to a file must be converted to a basic data exchange diskette file by the TRANSFER procedure if the diskette file is to be used as input to other systems.

Copy all members named PAYDAY in sector mode to a disk file named PAYROLL that is 60 sectors long, starts at location 1500, and is a temporary file:

```
// LOAD $MAINT
// FILE NAME-PAYROLL,UNIT-F1,BLOCKS-6,LOCATION-1500,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAYROLL,NAME-PAYDAY,LIBRARY-ALL
// END
```

Copy source and procedure members named PAYDAY in record mode with a record length of 120 to a disk file named PAY that is 80 sectors long and is to be retained permanently:

```
// LOAD $MAINT
// FILE NAME-PAY,UNIT-F1,BLOCKS-8,RETAIN-P
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAY,RECL-120,NAME-PAYDAY,
   LIBRARY-ALL
// END
```

Copy in sector mode all members whose names begin with a dollar sign ($). Copy the members to a file named UTIL that has a retention period of 90 days and is on a diskette whose vol-id is UTILITY:

```
// LOAD $MAINT
// FILE NAME-UTIL,UNIT-I1,RETAIN-90,PACK-UTILITY
// RUN
// COPY FROM-F1,TO-DISK,FILE-UTIL,NAME-$.ALL,LIBRARY-ALL
// END
```

Copy all source and procedure members whose names begin with the characters RPU, in record mode, with an 80-byte record length. Copy the members to a disk file named RPSD that is 50 sectors long and is classified as a permanent file:

```
// LOAD $MAINT
// FILE NAME-RPSD,UNIT-F1,BLOCKS-5,RETAIN-P
// RUN
// COPY FROM-F1,TO-DISK,FILE-RPSD,RECL-80,NAME-RPU.ALL,
   LIBRARY-ALL
// END
```

Copy in sector mode all members whose names begin with the characters PA, omitting members whose names start with PAY. Copy the members to a disk file named PAYR that is 60 sectors long and is classified as a temporary file:

```
// LOAD $MAINT
// FILE NAME-PAYR,UNIT-F1,BLOCKS-6,RETAIN-T
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAYR,NAME-PA.ALL,LIBRARY-ALL,
   OMIT-PAY.ALL
// END
```

Message Text Statement

The format of the message text statement is:

MIC Text

- MIC (message identification code). The MIC must be specified as a 1 to 4 character decimal number from 0 to 9999 and must be left-justified within the first four positions of the message text statement. The MIC must be in ascending order, relative to the MIC for the preceding message text statement, or the same MIC specified on consecutive message text statements to concatenate the text area. The number of statements that can be concatenated is restricted to the minimum number required to specify up to 200 characters of message text area.

  The MICs for the command keys are listed with the corresponding data characters:

MIC	Command Key (lowercase)	MIC	Command Key (uppercase)
0001	1	0013	|
0002	2	0014	@
0003	3	0015	#
0004	4	0016	$
0005	5	0017	%
0006	6	0018	¬
0007	7	0019	&
0008	8	0020	*
0009	9	0021	(
0010	0	0022	)
0011	- (minus)	0023	_ (underscore)
0012	=	0024	+

- Text (text area of the message text statement). The text area of each message text statement starts at position six and extends to the end of the message text statement (length of statement depends on record length of message source member). The text for a message is the series of characters from the start of the text area to the last nonblank character of the text area for the MIC. If text cannot be contained on one message text statement, it can be continued on following message text statement(s) specifying the same MIC. The text area on following statement(s) is appended to the text area of the preceding statement before trailing blanks for the total text specified are dropped. A message text of one blank will be generated for a message text statement containing a blank text area.

Comment Statement (Optional)

The format of the comment statement is:

```
* ... comment ...
```

Comment statements must have an * as the first character. Comment statements can be interspersed with the message text statements. These statements, intended to provide additional information about the message, do not become part of the message load member.

An Example of Creating a Message Source Member and Load Member

Assume that you want to enter from the keyboard into the library a message source member named USERMI. See the reader-to-library copy function of $MAINT (see index entry: *$MAINT utility program*). For more examples see index entry: *creating and using messages.*

```
USERMSG,1          A message control statement, USERMSG is the load-name and
                   1 is the message level.

1234 ENTER YESTERDAY'S DATE.   Message text statements; 1234, 1235,
1235 ENTER TODAY'S DATE.       and 1236 are MICs. The message text
1236 ENTER TOMORROW'S DATE.    follows the MICs.

* THE ABOVE MESSAGES ARE FOR PROGX.   A comment statement.
```

To create a message source member named USERMI from the above statements, you would enter on the keyboard:

```
// LOAD $MAINT
// RUN
// COPY FROM-READER,LIBRARY-S,NAME-USERMI,TO-F1,RETAIN-P,RECL-45
USERMSG,1
1234 ENTER YESTERDAY'S DATE.
1235 ENTER TODAY'S DATE.
1236 ENTER TOMORROW'S DATE.
* THE ABOVE MESSAGES ARE FOR PROGX.
// CEND
// END
```

To create a message load member named USERMSG from the above source member (USERMI), you could use the CREATE procedure (see index entry: *CREATE procedure*), entering:

```
CREATE USERMI
```

An Example of Assigning a Command Key to a Procedure

Assume that you want to enter from the keyboard into the library a message source member named JOE with the following statements. See the reader-to-library function of $MAINT. (See index entry: *$MAINT utility program*.) Assume also that you want command key 1 to represent the CATALOG command statement and command key 2 to represent the DATE command statement. Then when you need to execute one of these two procedures you can press the CMD key, the upper or lower case assigned key, and the ENTER key, instead of entering the command statement for commonly used procedures.

```
##MSG3,2
```
A message control statement; ##MSG3 is the load-name and 2 is the level of load member being created (see index entry: *command keys, assigning*).

```
0001 CATALOG ALL,F1
0002 DATE 12/19/75
```
Message text statements; 0001 and 0002 are the command key MICs. The message text follows the MICs and represent procedures.

To create a message source member named JOE from the above statements, you enter on the keyboard:

```
// LOAD $MAINT
// RUN
// COPY FROM-READER,LIBRARY-S,NAME-JOE,TO-F1,RECL-120
##MSG3,2
0001 CATALOG ALL,F1
0002 DATE 12/19/75
// CEND
// END
```

To create a message load member named ##MSG3,2 from the source member (JOE), you can use the CREATE procedure. (See index entry: *CREATE procedure*.) The CREATE procedure entry from the keyboard is:

```
CREATE JOE
```

Now you must perform an IPL (initial program load).

$PACK—DISK REORGANIZATION UTILITY PROGRAM

$PACK reorganizes the disk so that all free space on the disk is collected in one area at the high end of the disk. The reorganization is accomplished by successively moving each data file as closely as possible to the library.

If a file is being moved to a space that is smaller than the file, $PACK must overlay portions of the file in the process of moving it. Consequently, $PACK takes special precautions to ensure that data is not lost if a system failure occurs while $PACK is being used. If it is possible that data may be lost after such a failure, $PACK must be the first program run, except for $LABEL (see index entry: *$LABEL utility program*), after successful restart of the system.

To determine if $PACK must be rerun after a system failure occurred while $PACK was being used, evoke the $LABEL utility to display the disk VTOC. If data integrity on the disk was unaffected by the system failure, each VTOC entry is displayed. If there is a chance that data may be lost from a file, instead of that file's label being displayed, the following message is displayed on the display screen:

$PACK MUST BE RUN BEFORE INFORMATION CAN BE OBTAINED FROM THIS FILE.

$PACK is evoked by the COMPRESS procedure and APCHANGE procedure (see index entries: *COMPRESS procedure* and *APCHANGE procedure*).

Note: Because files are physically moved by $PACK, the locations specified by LOCATION parameters in FILE statements for the moved files (see index entry: *// FILE statement*) will not be valid. To determine new file locations after using $PACK, use the $LABEL utility or CATALOG procedure to display the disk VTOC.

To accumulate the free space at the low end of the disk, see *$FREE—DISK Reorganization Utility Program.*

$PACK Utility Control Statement Format

Because $PACK always reorganizes the disk in the same manner, utility control statements are not used.

$PACK OCL Sequence

```
// LOAD $PACK
// RUN
```

Delete Examples

Delete a non-SCP source member named PAYROLL:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-S,NAME-PAYROLL
// END
```

Delete all non-SCP members whose names begin with the characters INV:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-ALL,NAME-INV.ALL
// END
```

Delete all non-SCP procedures:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-P,NAME-ALL
// END
```

Compress Function

Compress Use

Compress the library member area by collecting all unusable space into one area at the end of the last active library member sector.

Compress Restrictions

The following restrictions apply to using the compress function:

- The library member area cannot be greater than 2867 blocks.
- If a permanent disk error occurs while the compress function is executing, there is no error recovery. The library must be reloaded from diskette (see index entry: *RELOAD procedure*).

Compress Control Statement Format

```
// COMPRESS
```

Compress Parameters

None

Compress OCL and Utility Control Statement Sequence

```
// LOAD $MAINT
// RUN
// COMPRESS
// END
```

$MGBLD–CREATE MESSAGE MEMBER UTILITY PROGRAM

The $MGBLD utility program creates a message load member in the library. A message load member is the special type of library load member from which the SCP retrieves the text associated with the message identification code (MIC) specified by a calling program.

$MGBLD is evoked by the CREATE procedure (see index entry: *CREATE procedure*).

$MGBLD Utility Control Statement Format

// MGBLD SOURCE-sourcename [,SCP- {YES / NO}] [,REPLACE- {YES / NO}]

$MGBLD Parameters

SOURCE-sourcename		Specifies the name of the library message source member that contains the message control statement and message text statements (MIC and text) required to create a message load member. See *Message Source Member* following for information about message control statements and message text statements.
SCP-	YES NO	If YES is specified, the message load member created is identified as an SCP member and cannot be deleted by the REMOVE procedure (see index entry: *REMOVE procedure*). If NO is specified, the message load member is *not* identified as an SCP member. NO is the default.
REPLACE-	YES NO	If YES is specified, the message load member replaces an existing member with the same name. If NO is specified, an existing member with the same name is not replaced. An error message is displayed if an attempt is made to replace a member. NO is the default.

$REBLD OCL Sequence

```
// LOAD $REBLD
// RUN
```

$RENAM--RENAME DATA FILE UTILITY PROGRAM

The $RENAM utility program changes the name of a specified data file from its created name to the name supplied by the utility control statement or by the RENAME command statement. No other attribute of the selected file can be changed.

$RENAM Utility Control Statement Format

```
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE-{mmddyy|ddmmyy|yymmdd}]
```

$RENAM Parameters

LABEL-filename-1	Specifies the file name to be changed. A file by this name must exist on disk prior to evoking $RENAM.
NEWLABEL-filename-2	Specifies the name of the file after being renamed.
DATE-{mmddyy | ddmmyy | yymmdd}	Creation date of the disk file. When the file specified by the LABEL parameter is part of a group of files with like names but different creation dates, the DATE parameter permits selection of a specific file. If the DATE parameter is omitted, the file with the most recent date is renamed.

$RENAME OCL and Utility Control Statement Sequence

```
// LOAD $RENAM
// RUN
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE-{mmddyy|ddmmyy|yymmdd}]
// END
```

Note: Multiple RENAME statements may be entered before the END statement.

$RENAM Examples

Change the name of a disk file named OLD to NEW.

```
// LOAD $RENAM
// RUN
// RENAME LABEL-OLD,NEWLABEL-NEW
// END
```

Change the name of a disk file named A created on 10/17/78 to B when more than one file exists with the name A.

```
// LOAD $RENAM
// RUN
// RENAME LABEL-A,NEWLABEL-B,DATE-101778
// END
```

$SETCF—SET UTILITY PROGRAM

$SETCF is used to change the following items:

- System environment
- BSC environment
- SDLC environment
- Override BSC specifications
- Specify SDLC specifications
- Trace functions
- MICR document movement

When the system is created for the first time (the initial IPL), values for these items are recorded in the system. These values can be changed by $SETCF. If a value is never changed, it retains its original status. If an item is changed, the new value is reflected in subsequent IPLs until the item is changed again (except for the DEBUG-Y parameter which is reset by each IPL or by the TRACE procedure).

$SETCF is evoked by the SET, ALTERBSC, ALTERSDL, OVERRIDE, SPECIFY, TRACE, and SETMICR procedures (see index entries: *SET procedure, ALTERBSC procedure, ALTERSDL procedure, OVERRIDE procedure, SPECIFY procedure,* and *TRACE procedure*).

Note: The SETMICR procedure is used to modify the method of moving MICR documents through the 1255 Magnetic Character Reader attachment for diagnostic purposes. For further information on the SETMICR procedure and the 1255 Magnetic Character Reader, see *IBM System/32 1255 Magnetic Character Reader Reference and Logic Manual,* GC21-7692.

Set the System Environment

The following system environment items can be defined by $SETCF:

- BSC
- SDLC
- Number of lines printed per page
- Print belt image
- System date format
- System date

Utility Control Statement Format for Setting the System Environment

```
// SETCF [LINES-number] [,IMAGE- {YES}] [,FORMAT- {MDY}]
                              {NO }              {DMY}
                                                 {YMD}
```

Note: Though each individual parameter is optional, at least one parameter must be entered.

Parameters for Setting the System Environment

LINES-number	Specifies the number of lines to be printed per page. The value specified can be 1 through 84. *Note:* See index entry: // *FORMS statement* for the way the value specified is used to determine the actual number of lines printed per page.
IMAGE-YES	The print belt image is to be modified to reflect a changed print belt. An IMAGE OCL statement (see index entry: // *IMAGE statement*) identifying the new image must precede the accompanying // RUN statement if IMAGE-YES is specified in a // SETCF statement.
IMAGE-NO	The print belt image is not to be modified. IMAGE-NO is the default value if IMAGE-YES is not specified.
FORMAT-MDY	System date format is to be month-day-year.
FORMAT-DMY	System date format is to be day-month-year.
FORMAT-YMD	System date format is to be year-month-day. *Note:* Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.

OCL and Utility Control Statement Sequence for Setting the System Environment

```
// LOAD $SETCF
[// DATE . . . ]
```

Note: If a date is given, it becomes the system date.

```
[// IMAGE . . .]
```

Note: If a print belt image is specified by $SETCF, it becomes the image set by IPL. If an image is specified by the // IMAGE OCL statement and not by $SETCF, the image is established only until the next IPL is performed, at which time a different image may be specified for the system.

```
// RUN
// SETCF ...
// END
```

Comment Statement (Optional)

The format of the comment statement is:

```
* ... comment ...
```

Comment statements must have an * as the first character. Comment statements can be interspersed with the message text statements. These statements, intended to provide additional information about the message, do not become part of the message load member.

An Example of Creating a Message Source Member and Load Member

Assume that you want to enter from the keyboard into the library a message source member named USERMI. See the reader-to-library copy function of $MAINT (see index entry: *$MAINT utility program*). For more examples see index entry: *creating and using messages.*

```
USERMSG,1      { A message control statement, USERMSG is the load-name and
               { 1 is the message level.

1234 ENTER YESTERDAY'S DATE.   { Message text statements; 1234, 1235,
1235 ENTER TODAY'S DATE.       { and 1236 are MICs. The message text
1236 ENTER TOMORROW'S DATE.    { follows the MICs.

* THE ABOVE MESSAGES ARE FOR PROGX.   { A comment statement.
```

To create a message source member named USERMI from the above statements, you would enter on the keyboard:

```
// LOAD $MAINT
// RUN
// COPY FROM-READER,LIBRARY-S,NAME-USERMI,TO-F1,RETAIN-P,RECL-45
USERMSG,1
1234 ENTER YESTERDAY'S DATE.
1235 ENTER TODAY'S DATE.
1236 ENTER TOMORROW'S DATE.
* THE ABOVE MESSAGES ARE FOR PROGX.
// CEND
// END
```

To create a message load member named USERMSG from the above source member (USERMI), you could use the CREATE procedure (see index entry: *CREATE procedure*), entering:

```
CREATE USERMI
```

An Example of Assigning a Command Key to a Procedure

Assume that you want to enter from the keyboard into the library a message source member named JOE with the following statements. See the reader-to-library function of $MAINT. (See index entry: *$MAINT utility program*.) Assume also that you want command key 1 to represent the CATALOG command statement and command key 2 to represent the DATE command statement. Then when you need to execute one of these two procedures you can press the CMD key, the upper or lower case assigned key, and the ENTER key, instead of entering the command statement for commonly used procedures.

```
##MSG3,2
```

A message control statement; ##MSG3 is the load-name and 2 is the level of load member being created (see index entry: *command keys, assigning*).

```
0001 CATALOG ALL,F1
0002 DATE 12/19/75
```

Message text statements; 0001 and 0002 are the command key MICs. The message text follows the MICs and represent procedures.

To create a message source member named JOE from the above statements, you enter on the keyboard:

```
// LOAD $MAINT
// RUN
// COPY FROM-READER,LIBRARY-S,NAME-JOE,TO-F1,RECL-120
##MSG3,2
0001 CATALOG ALL,F1
0002 DATE 12/19/75
// CEND
// END
```

To create a message load member named ##MSG3,2 from the source member (JOE), you can use the CREATE procedure. (See index entry: *CREATE procedure*.) The CREATE procedure entry from the keyboard is:

```
CREATE JOE
```

Now you must perform an IPL (initial program load).

$PACK–DISK REORGANIZATION UTILITY PROGRAM

$PACK reorganizes the disk so that all free space on the disk is collected in one area at the high end of the disk. The reorganization is accomplished by successively moving each data file as closely as possible to the library.

If a file is being moved to a space that is smaller than the file, $PACK must overlay portions of the file in the process of moving it. Consequently, $PACK takes special precautions to ensure that data is not lost if a system failure occurs while $PACK is being used. If it is possible that data may be lost after such a failure, $PACK must be the first program run, except for $LABEL (see index entry: *$LABEL utility program*), after successful restart of the system.

To determine if $PACK must be rerun after a system failure occurred while $PACK was being used, evoke the $LABEL utility to display the disk VTOC. If data integrity on the disk was unaffected by the system failure, each VTOC entry is displayed. If there is a chance that data may be lost from a file, instead of that file's label being displayed, the following message is displayed on the display screen:

$PACK MUST BE RUN BEFORE INFORMATION CAN BE OBTAINED FROM THIS FILE.

$PACK is evoked by the COMPRESS procedure and APCHANGE procedure (see index entries: *COMPRESS procedure* and *APCHANGE procedure*).

Note: Because files are physically moved by $PACK, the locations specified by LOCATION parameters in FILE statements for the moved files (see index entry: *// FILE statement*) will not be valid. To determine new file locations after using $PACK, use the $LABEL utility or CATALOG procedure to display the disk VTOC.

To accumulate the free space at the low end of the disk, see *$FREE–DISK Reorganization Utility Program.*

$PACK Utility Control Statement Format

Because $PACK always reorganizes the disk in the same manner, utility control statements are not used.

$PACK OCL Sequence

```
// LOAD $PACK
// RUN
```

$QJOB—QUEUED JOB STREAM CARD-TO-LIBRARY UTILITY PROGRAM

The $QJOB utility program transfers a job stream containing procedure and source members created on cards to the System/32 library. $QJOB may be evoked by the JOBSTR procedure (see index entry: *JOBSTR procedure*).

The job stream you create on cards can consist of multiple procedure and source members. Each procedure and source member must begin with a COPY statement and end with a CEND statement. The last record in your card deck must be a /* statement and must immediately follow the last CEND statement.

The format of the COPY statement is:

```
// COPY NAME-name,LIBRARY-{P}
                          {S}
```

The name is the member name, and P or S indicates a procedure member or a source member.

The format of the CEND statement is:

```
// CEND
```

The CEND statement is not valid within a source or procedure member. It is valid only as the last statement for the source or procedure member.

A job stream created on cards could contain the following statements:

```
// COPY NAME-P1,LIBRARY-P
      :
      :
// CEND
// COPY NAME-P2,LIBRARY-P
      :
      :
// CEND
// COPY NAME-S1,LIBRARY-S
      :
      :
// CEND
/*
```

$QJOB Utility Control Statement Format

Utility control statements are not used.

$QJOB OCL Sequence

The following entries are required to load and run the program:

```
// LOAD $QJOB
// RUN
```

$REBLD—REBUILD DATA FILE UTILITY PROGRAM

For each file on the disk, a corresponding *format 1* record exists. A format 1 record contains system information that describes a file. $REBLD is used to restore, in the disk VTOC, format 1 records for disk output files that were being processed when a system failure occurred—a failure caused, for example, by a power failure or inadvertent IPL. When $REBLD is used after a system failure, the output files are closed and the format 1 records are written to the disk VTOC, allowing the data that was written to the files prior to the system failure to be accessible to the user. In effect, by restoring format 1 records to the VTOC, $REBLD restores data files that might otherwise be lost. If $REBLD is not used after a system failure, certain output files may not be accessible to the user.

Note: $REBLD must be the first program run after a system failure unless the system failed while the $PACK utility was being used (see index entry: *$PACK utility*).

$REBLD searches the scheduler work area in the library for format 1 records that are opened or opened and closed but not written to the disk VTOC. When such a format 1 is found, a check is made to determine if the format 1 is for an input file or for an output file. If the format 1 is for an input file, it is updated to a completed status and written to the disk VTOC as in normal end-of-job processing. If the format 1 is for an output file, another check is made to determine the file organization.

Sequential file and Pseudo tape file	The logical end of file is made equal to the physical end of file. The format 1 is updated to a completed status and written to the disk VTOC via normal end-of-job processing.
Indexed file	The last indexed entry is checked for a valid data record. If not valid, the previous indexed entry is checked, and so on, until an indexed entry with a valid data record is found. The format 1 is updated to a closed status. The keys are sorted and the format 1 is written to the disk VTOC via normal end-of-job processing.
Direct file	The format 1 is updated to a completed status and written to the disk VTOC via normal end-of-job processing.

Add and update files are treated as output files. $REBLD restores only temporary (RETAIN-T) and permanent (RETAIN-P) files. Scratch files (RETAIN-S) are not restored.

As $REBLD runs, messages are issued giving the labels, filenames (from // FILE statements), creation dates, organization (sequential, indexed, direct, or pseudo tape) of files restored, and the key of the last valid record for indexed files. If no files required restoring, a message to that effect is issued. $REBLD is evoked by the REBUILD procedure (see index entry: *REBUILD procedure*).

$REBLD Utility Control Statement Format

Utility control statements are not used.

$REBLD OCL Sequence

```
// LOAD $REBLD
// RUN
```

$RENAM—RENAME DATA FILE UTILITY PROGRAM

The $RENAM utility program changes the name of a specified data file from its created name to the name supplied by the utility control statement or by the RENAME command statement. No other attribute of the selected file can be changed.

$RENAM Utility Control Statement Format

```
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE-{mmddyy|ddmmyy|yymmdd}]
```

$RENAM Parameters

LABEL-filename-1	Specifies the file name to be changed. A file by this name must exist on disk prior to evoking $RENAM.
NEWLABEL-filename-2	Specifies the name of the file after being renamed.
DATE-{mmddyy|ddmmyy|yymmdd}	Creation date of the disk file. When the file specified by the LABEL parameter is part of a group of files with like names but different creation dates, the DATE parameter permits selection of a specific file. If the DATE parameter is omitted, the file with the most recent date is renamed.

$RENAM OCL and Utility Control Statement Sequence

```
// LOAD $RENAM
// RUN
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE-{mmddyy|ddmmyy|yymmdd}]
// END
```

Note: Multiple RENAME statements may be entered before the END statement.

$RENAM Examples

Change the name of a disk file named OLD to NEW.

```
// LOAD $RENAM
// RUN
// RENAME LABEL-OLD,NEWLABEL-NEW
// END
```

Change the name of a disk file named A created on 10/17/78 to B when more than one file exists with the name A.

```
// LOAD $RENAM
// RUN
// RENAME LABEL-A,NEWLABEL-B,DATE-101778
// END
```

$SETCF—SET UTILITY PROGRAM

$SETCF is used to change the following items:

- System environment
- BSC environment
- SDLC environment
- Override BSC specifications
- Specify SDLC specifications
- Trace functions
- MICR document movement

When the system is created for the first time (the initial IPL), values for these items are recorded in the system. These values can be changed by $SETCF. If a value is never changed, it retains its original status. If an item is changed, the new value is reflected in subsequent IPLs until the item is changed again (except for the DEBUG-Y parameter which is reset by each IPL or by the TRACE procedure).

$SETCF is evoked by the SET, ALTERBSC, ALTERSDL, OVERRIDE, SPECIFY, TRACE, and SETMICR procedures (see index entries: *SET procedure, ALTERBSC procedure, ALTERSDL procedure, OVERRIDE procedure, SPECIFY procedure,* and *TRACE procedure*).

Note: The SETMICR procedure is used to modify the method of moving MICR documents through the 1255 Magnetic Character Reader attachment for diagnostic purposes. For further information on the SETMICR procedure and the 1255 Magnetic Character Reader, see *IBM System/32 1255 Magnetic Character Reader Reference and Logic Manual,* GC21-7692.

Set the System Environment

The following system environment items can be defined by $SETCF:

- BSC
- SDLC
- Number of lines printed per page
- Print belt image
- System date format
- System date

Utility Control Statement Format for Setting the System Environment

```
// SETCF [LINES-number] [ ,IMAGE- {YES} ] [ ,FORMAT- {MDY} ]
                          {NO }          {DMY}
                                         {YMD}
```

Note: Though each individual parameter is optional, at least one parameter must be entered.

Parameters for Setting the System Environment

LINES-number — Specifies the number of lines to be printed per page. The value specified can be 1 through 84.

Note: See index entry: // *FORMS statement* for the way the value specified is used to determine the actual number of lines printed per page.

IMAGE-YES — The print belt image is to be modified to reflect a changed print belt. An IMAGE OCL statement (see index entry: // *IMAGE statement*) identifying the new image must precede the accompanying // RUN statement if IMAGE-YES is specified in a // SETCF statement.

IMAGE-NO — The print belt image is not to be modified. IMAGE-NO is the default value if IMAGE-YES is not specified.

FORMAT-MDY — System date format is to be month-day-year.

FORMAT-DMY — System date format is to be day-month-year.

FORMAT-YMD — System date format is to be year-month-day.

Note: Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.

OCL and Utility Control Statement Sequence for Setting the System Environment

```
// LOAD $SETCF
[// DATE . . . ]
```

Note: If a date is given, it becomes the system date.

```
[// IMAGE . . .]
```

Note: If a print belt image is specified by $SETCF, it becomes the image set by IPL. If an image is specified by the // IMAGE OCL statement and not by $SETCF, the image is established only until the next IPL is performed, at which time a different image may be specified for the system.

```
// RUN
// SETCF ...
// END
```

Example of Setting the System Environment

Replace the current print belt image with the image contained in the source member named BELT:

```
// LOAD $SETCF
// IMAGE MEM,BELT
// RUN
// SETCF IMAGE-YES
// END
```

Set the BSC Environment

The following BSC (binary synchronous communications) items can be set by $SETCF:

- Bits per seconds rate (bps)
- Modem clocking
- Debug facility
- Error retry count
- Standby line
- Modem test
- Non-USA tone

Note: The items listed are all related to data communications programming that uses BSC. For background information on binary synchronous communications, see *General Information—Binary Synchronous Communications,* GA27-3004.

For information on data communications programming, see *IBM System/32 Data Communications Reference Manual,* GC21-7691.

SETB Utility Control Statement Format for Setting the BSC Environment

Use the SETB utility control statement to set the BSC environment:

```
// SETB [BRATE-{F}] [,CLOCK-{Y}] [,DEBUG-{Y}] [,ERC-{number}]
        {H}       {N}          {N}          {7}
[,SLINE-{Y}] [,TEST-{Y}] [,TONE-{Y}]
        {N}        {N}        {N}
```

For an explanation of the SETB parameters, see *ALTERBSC Command Statement Format.*

Note: Though each individual parameter is optional, at least one parameter must be specified.

If a parameter is omitted, the previous value is retained until a default value is given (except for the DEBUG-Y parameter which is reset to DEBUG-N by each IPL or by the TRACE procedure). If DEBUG-Y is specified, the system TRACE procedure (see index entry: *TRACE procedure*) is replaced by the BSC trace function.

Parameters for Setting the BSC Environment

Parameter	Meaning
BRATE-F	Use the full rated speed of the modem.
H	Use only half the rated speed of the modem.
CLOCK-Y	The System/32 must provide the programmed clocking facility.
N	Modem has the clocking facility.
DEBUG-Y	Built-in debug facility is required, BSC trace is requested.
N	Built-in debug facility is not required, BSC trace is not requested.
ERC-number	Error retry count. The standard number of retries provided is seven (the default number); if more than seven are desired, enter
7	a number up to 255. Valid numbers are 7 through 255.
SLINE-Y	Switched standby line is used for a point-to-point line.
N	The nonswitched line is used.
TEST-Y	IBM modem is being used. Automatic wrap test includes modem testing when a permanent error occurs, unless the user program specified a permanent error indicator for the BSC file.
N	Non-IBM modem is being used. Automatic wrap test does not include modem testing.
TONE-Y	Non-USA special tone is required for manual answering and automatic answering.
N	Non-USA special tone is not required for manual answering and automatic answering.

Notes:

1. If the SLINE-Y parameter is specified, then the line type (LINE) in the // SETR utility control statement automatically defaults to a point-to-point switched line (LINE-S).
2. If the SLINE-N parameter is specified, then the line type (LINE) in the // SETR utility control statement automatically defaults to the line type specified in the user program source statements (LINE-R).

Example of Setting the BSC Environment

Change the current BSC error retry count to 10:

```
// LOAD $SETCF
// RUN
// SETB ERC-10
// END
```

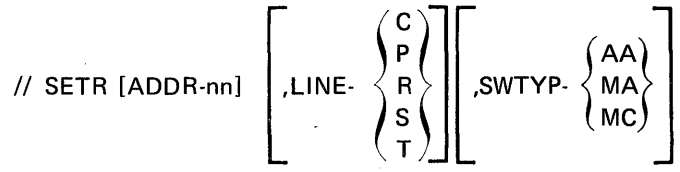
Override BSC Specifications

The following BSC (binary synchronous communications) specifications can be overridden by $SETCF:

- Tributary station address
- Line type
- Switch type

Note: The items listed are all related to data communications programming that uses BSC. For background information on binary synchronous communications, see *General Information—Binary Synchronous Communications,* GA27-3004. For information on data communications programming, see *IBM System/32 Data Communications Reference Manual,* GC21-7691.

Utility Control Statement Format for Overriding BSC Specifications



Notes:
1. Though each parameter is optional, at least one parameter must be specified.
2. To reset the ADDR parameter to the addressing characters specified by the user program specifications, reenter a valid // SETR control statement omitting the ADDR parameter. The addressing characters will default to the user program specifications.

Parameters for overriding BSC specifications:

ADDR-nn	Hexadecimal equivalent of one of the pair of tributary station addressing characters. See Appendix G for the System/32 tributary station polling and addressing characters. Defaults to the user program specifications.
LINE-C	CDSTL (connect data set to line) switched line (World Trade only)
P	Point-to-point nonswitched line.
R	Line type specified in the user program source statements.
S	Point-to-point switched line.
T	Tributary station on multipoint line.
SWTYP-AA	If switched line (LINE-C or LINE-S) is specified and the modem is in auto-answer mode, then the System/32 automatically answers the call.
MA	If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually answers the call.
MC	If switched line (LINE-C or LINE-S) is specified, then the System/32 operator manually initiates the call.

Introduction To System Configuration, Installation, and Modification

System/32 programs are supported through the distribution of sequentially numbered versions or modifications. A version replaces an entire program; a modification generally replaces only the changed portions of a program. Each program has a version number and a modification level associated with it.

A release is a group of programs made available at the same time. Release generally refers to the period of time for which it is supported; however, a release may consist of programs with a different version/modification level identification. For example, release 2 of SCP and program products may include three programs designated version 02 modification 00, and one program designated version 01, modification 00.

The initial availability of a program is usually called version 01, modification 00. Each subsequent modification raises the modification level by one. Each version raises the version number by one and resets the modification level to zero.

Versions and modifications are made available in one of two ways. Some are sent automatically by the program library to all users, and all others are sent when ordered by the user. In the latter case, ordering instructions are sent to users by the program library.

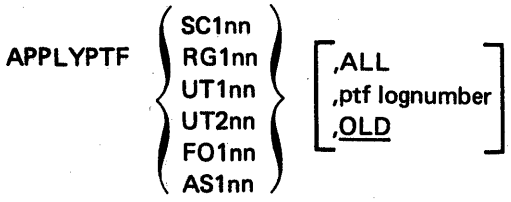
The version number and modification level of each program is indicated on the machine readable material and in the documentation sent with the program from the program library. In cases where a version number or modification level is skipped, the documentation from the program library notes such action.

This part includes:

- How to configure and load System/32 system control programming and related PTFs (program temporary fixes), whether you are loading your initial version of the system control programming or a subsequent version
- How to install a system containing System/32 system control programming and selected System/32 program products and applications, together with related PTFs
- How to install individual System/32 program products and related PTFs and verify the correct installation of System/32 program products
- How to modify your system by deleting system control programming components and/or program products from the library so that you have more disk space available for other library members or for data files
- A version update instruction summary

Note: Installing the word processing program product is described in the *Word Processor/32 Installation and Procedures Manual,* SH30-0114.

Three SCP procedures are described in this part: CNFIGSCP, INSTALL, and APPLYPTF. The formats of the command statements that evoke these procedures are:



CNFIGSCP

INSTALL [DFU] [,SEU] [,SORT] [,RPG]
[,FORT] [,FCU] [,ASM]

This section describes how to configure and load both your initial version and subsequent versions of System/32 system control programming. This section also describes how to apply any required PTFs to the System/32 SCP and to the program products you intend to install with the SCP. (Installation of program products with the SCP is described in the next section, *System Installation*.)

DISKETTES REQUIRED

The diskettes required to perform system configuration are:

- PID (program information department) distribution diskettes, called SCP diskettes. These diskettes contain the following:

  1-2 System control programming.
  3 Optional SCP support for data communications, RPG, and data recorder attachment[1].
  4 Optional SCP support for 1255 Magnetic Character Reader attachment[1], FORTRAN IV, basic assembler[1], overlay linkage editor[1], and queued job stream[1].
  [5] Optional SCP support for word processing communications utility and word processing, which includes the mag card attachment, dual case keyboard and display, and half line space printing.

- PID distribution diskettes containing any program products ordered.

- Diskettes on which a backup of the system control program can be made. They are called backup diskettes. The number of backup diskettes depends on the optional SCP support you require.

- Backup diskettes for each program product ordered.

To determine the number of diskettes required by a program product, see index entry: *backup copy of a program product.*

Note: Your IBM service representative can tell you if there are any PTFs applicable to your version of the SCP, or to your version of any program products. If there are PTFs, make arrangements with IBM to have the PTF diskette available when you configure and load your SCP. The PTF diskette contains all applicable PTFs.

---

[1] If you install the optional SCP support for this function without reconfiguring your system, you must also install message member MSGMBR, which is on the fourth PID diskette, and then do another IPL for the system.

INFORMATION REQUIRED

During the configuration of the SCP, you will be prompted for the following information:

- Print belt image for your system.
- Number of printed lines per page.
- The date format you will be using.
- Is SCP support for data communications desired?

  If your response is YES you are prompted for the following:
  - Is BSC support desired?
  - Is MRJE support desired?

  If your response is NO to both the BSC support and the MRJE support you are prompted for:
  - Is batch work station support desired?

  By specifying BSC support, MRJE support, or batch work station support (SDLC) the following initial configuration options are set for you:
  - Line rate will be full.
  - Standby line option is NO.
  - Error retry count is 7. (Set only for BSC and MRJE support.)
  - Debug option is NO.

  After initial configuration options are set, you are prompted for World Trade answer tone.
  - Is World Trade answer tone required?

  The line type option for BSC, MRJE and batch work station support (SDLC) follows. You respond with a character C, P, R, S, or T to indicate the following:

C	CDSTL (connect data set to line) switched line (World Trade only).
P	Point-to-point nonswitched line.
R	The line type specified in the user program source statements. (Line type R does not appear for SDLC–batch work station support.)
S	Point-to-point switched line.
T	Tributary station line on multipoint.

If your response for the preceding line type option is either a C or an S (switched line), the prompt for the switched line type option appears. You respond with one of the following sets of characters:

AA	The System/32 automatically answers the call. The modem must also be in autoanswer mode.
MA	The System/32 operator manually answers the call.
MC	The System/32 operator manually initiates the call.

The final two prompts for BSC, MRJE, and batch work station support (SDLC) are:
- Does the modem perform clocking?
- Is an IBM modem installed?

- Is SCP support for RPG desired?

  If your response is YES, you are prompted for the following:
  - Is data communications support for RPG desired?

- Is SCP support for the data recorder attachment desired?

- Is SCP support for word processing desired?

  If your response is YES, you are prompted for the following:
  - Is SCP support for word processing communications utility desired?

  If your response for word processing support is YES, you are prompted for the following:
  - Select country code
    Options are:
    01 = USA
    02 = United Kingdom
    03 = Germany & Austria
    04 = France (AZERTY)
    05 = Italy
    06 = Denmark
    17 = France (QWERTY)

    Enter a two-character country code
    Options: (01, 02, 03, 04, 05, 06, 17)

- Is SCP support for 1255 Magnetic Character Reader attachment desired?

- Is SCP support for FORTRAN IV desired?[1]

- Is SCP support for basic assembler desired?[1]

  If your response is YES, you are prompted for the following:
  - Are SCP base assembler macros desired?
  - Are SCP BSC assembler macros desired?
  - Are SCP scientific macros desired?

- Is SCP support for overlay linkage editor desired?[1]

- Is SCP support for queued job stream desired?

- Is there a PTF master diskette?

See index entry: *CNFIGSCP procedure* for the prompted parameter information you supply in step 10 of the System Configuration Steps.

Using the information supplied to the prompts, an SCP is built that contains the support that you request. If a PTF diskette is available, the PTFs are applied to the SCP on the disk.

---

[1] When you specify SCP support for either basic assembler or FORTRAN IV, SCP support for the overlay linkage editor is also provided. The prompt for SCP support for the overlay linkage editor does not appear.

SYSTEM CONFIGURATION STEPS

CAUTION
The system configuration steps remove the current library (if any) from the disk. Save all library members you want to retain (see index entry: *FROMLIBR procedure*) before executing the system configuration steps. When installing version updates, if IBM program products are installed on the system, remove the IBM program products.

The program products can be removed by entering the following appropriate commands for the program product you have installed:

SEUDROP (if SEU is installed)

DFUDROP (if DFU is installed)

SORTDROP (if SORT is installed)

RPGDROP (if RPG is installed)

FCUDROP (if FCU is installed)

FORTDROP (if FORTRAN IV is installed)

ASMDROP (if basic assembler is installed)

The system configuration steps are:

1. Set the IPL switch (on the CE control panel) to DISKETTE and set the IMPL switch (on the CE control panel) to DISK.

2. Insert the first PID SCP diskette.

3. Press the LOAD key on the operator panel. The following example display appears:

```
---> LIBRARY DIRECTORY SECTORS = 0037
     INCLUDE INQUIRE/OFFLINE   = NO
     TOTAL LIBRARY BLOCKS      = 0281
```

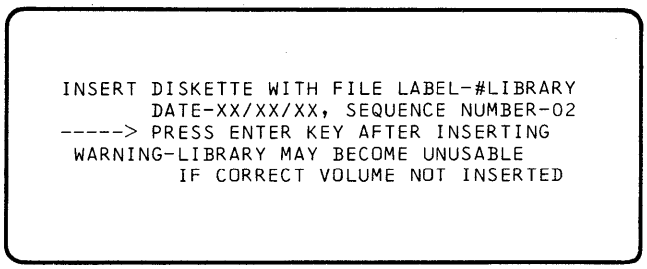
4. The preceding values displayed are those of a sample PID diskette. If an error message is displayed, see index entry: *system configuration error messages.*

5. If you want a smaller library, you must decrease the number of directory sectors and library blocks allocated. If you want a larger library, you should now allocate enough directory sectors and library blocks to contain the program products, the SCP, and any other programs. See index entry: *RELOAD display,* for a description of how to change the number of allocated directory sectors and library blocks. See index entry: *library requirements,* to determine the number of directory sectors and library blocks required by the SCP and the program products.

6. If any error messages are displayed, see index entry: *system configuration error messages.*

7. If no error messages are displayed, press the ENTER key to copy the SCP to the disk.

8. When the following display appears, remove the first PID SCP diskette and insert the second PID SCP diskette.



To copy the second PID SCP diskette to the disk, press the ENTER key.

This page intentionally left blank

XFER-Y	Each execution of the XFER instruction is to be traced.
N	Executions of the XFER instruction are not to be traced.

OCL and Utility Control Statement Sequence for Setting Functions to be Traced

```
// LOAD $SETCF
// RUN
// TRACE ...
// END
```

Example of Setting the Functions to be Traced

Trace evocations of the wait function:

```
// LOAD $SETCF
// RUN
// TRACE WAIT-Y
// END
```

OCL and Utility Control Statement Sequence for Modifying MICR Document Movement

```
// LOAD $SETCF
// RUN
// SETR CYCLE-{Y}
              {N}
// END
```

$STATS—STATUS DISPLAY UTILITY PROGRAM

$STATS displays current system information on the display screen, and prints it, if the printer is assigned for logging (see index entry: *LOG procedure*), so that you can determine whether or not certain items need to be changed for a job. A detailed description of the information displayed by $STATS is given with the description of the STATUS procedure, which evokes $STATS. (See index entry: *STATUS procedure*.)

$STATS Utility Control Statement Format

Utility control statements are not used.

$STATS OCL Sequence

```
// LOAD $STATS
// RUN
```

This page intentionally left blank.

Part 5

System Configuration, Installation, and Modification

System/32 programs are supported through the distribution of sequentially numbered versions or modifications. A version replaces an entire program; a modification generally replaces only the changed portions of a program. Each program has a version number and a modification level associated with it.

A release is a group of programs made available at the same time. Release generally refers to the period of time for which it is supported; however, a release may consist of programs with a different version/modification level identification. For example, release 2 of SCP and program products may include three programs designated version 02 modification 00, and one program designated version 01, modification 00.

The initial availability of a program is usually called version 01, modification 00. Each subsequent modification raises the modification level by one. Each version raises the version number by one and resets the modification level to zero.

Versions and modifications are made available in one of two ways. Some are sent automatically by the program library to all users, and all others are sent when ordered by the user. In the latter case, ordering instructions are sent to users by the program library.

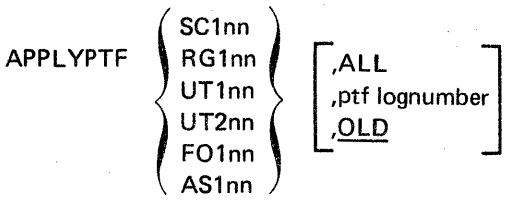
The version number and modification level of each program is indicated on the machine readable material and in the documentation sent with the program from the program library. In cases where a version number or modification level is skipped, the documentation from the program library notes such action.

This part includes:

- How to configure and load System/32 system control programming and related PTFs (program temporary fixes), whether you are loading your initial version of the system control programming or a subsequent version
- How to install a system containing System/32 system control programming and selected System/32 program products and applications, together with related PTFs
- How to install individual System/32 program products and related PTFs and verify the correct installation of System/32 program products
- How to modify your system by deleting system control programming components and/or program products from the library so that you have more disk space available for other library members or for data files
- A version update instruction summary

Note: Installing the word processing program product is described in the *Word Processor/32 Installation and Procedures Manual,* SH30-0114.

Three SCP procedures are described in this part: CNFIGSCP, INSTALL, and APPLYPTF. The formats of the command statements that evoke these procedures are:



CNFIGSCP

INSTALL [DFU] [,SEU] [,SORT] [,RPG]
[,FORT] [,FCU] [,ASM]

This section describes how to configure and load both your initial version and subsequent versions of System/32 system control programming. This section also describes how to apply any required PTFs to the System/32 SCP and to the program products you intend to install with the SCP. (Installation of program products with the SCP is described in the next section, *System Installation*.)

DISKETTES REQUIRED

The diskettes required to perform system configuration are:

- PID (program information department) distribution diskettes, called SCP diskettes. These diskettes contain the following:

  1-2 System control programming.
  3 Optional SCP support for data communications, RPG, and data recorder attachment[1].
  4 Optional SCP support for 1255 Magnetic Character Reader attachment[1], FORTRAN IV, basic assembler[1], overlay linkage editor[1], and queued job stream[1].
  [5] Optional SCP support for word processing communications utility and word processing, which includes the mag card attachment, dual case keyboard and display, and half line space printing.

- PID distribution diskettes containing any program products ordered.

- Diskettes on which a backup of the system control program can be made. They are called backup diskettes. The number of backup diskettes depends on the optional SCP support you require.

- Backup diskettes for each program product ordered.

To determine the number of diskettes required by a program product, see index entry: *backup copy of a program product.*

Note: Your IBM service representative can tell you if there are any PTFs applicable to your version of the SCP, or to your version of any program products. If there are PTFs, make arrangements with IBM to have the PTF diskette available when you configure and load your SCP. The PTF diskette contains all applicable PTFs.

---

[1] If you install the optional SCP support for this function without reconfiguring your system, you must also install message member MSGMBR, which is on the fourth PID diskette, and then do another IPL for the system.

INFORMATION REQUIRED

During the configuration of the SCP, you will be prompted for the following information:

- Print belt image for your system.
- Number of printed lines per page.
- The date format you will be using.
- Is SCP support for data communications desired?[1]

  If your response is YES you are prompted for the following:
  - Is BSC support desired?
  - Is MRJE support desired?

  If your response is NO to both the BSC support and the MRJE support you are prompted for:
  - Is batch work station support desired?

  By specifying BSC support, MRJE support, or batch work station support (SDLC) the following initial configuration options are set for you:
  - Line rate will be full.
  - Standby line option is NO.
  - Error retry count is 7. (Set only for BSC and MRJE support.)
  - Debug option is NO.

  After initial configuration options are set, you are prompted for World Trade answer tone.
  - Is World Trade answer tone required?

  The line type option for BSC, MRJE and batch work station support (SDLC) follows. You respond with a character C, P, R, S, or T to indicate the following:

C	CDSTL (connect data set to line) switched line (World Trade only).
P	Point-to-point nonswitched line.
R	The line type specified in the user program source statements. (Line type R does not appear for SDLC—batch work station support.)
S	Point-to-point switched line.
T	Tributary station line on multipoint.

If your response for the preceding line type option is either a C or an S (switched line), the prompt for the switched line type option appears. You respond with one of the following sets of characters:

AA	The System/32 automatically answers the call. The modem must also be in autoanswer mode.
MA	The System/32 operator manually answers the call.
MC	The System/32 operator manually initiates the call.

The final two prompts for BSC, MRJE, and batch work station support (SDLC) are:
- Does the modem perform clocking?
- Is an IBM modem installed?

---

[1] If SCP support for word processing communications utility is required, you must also specify SCP support for data communications.

- Is SCP support for RPG desired?

  If your response is YES, you are prompted for the following:
  - Is data communications support for RPG desired?

- Is SCP support for the data recorder attachment desired?

- Is SCP support for word processing desired?

  If your response is YES, you are prompted for the following:
  - Is SCP support for word processing communications utility desired?[2]

  If your response for word processing support is YES, you are prompted for the following:
  - Select country code
    Options are:
    01 = USA
    02 = United Kingdom
    03 = Germany & Austria
    04 = France (AZERTY)
    05 = Italy
    06 = Denmark
    17 = France (QWERTY)

    Enter a two-character country code
    Options: (01, 02, 03, 04, 05, 06, 17)

- Is SCP support for 1255 Magnetic Character Reader attachment desired?

- Is SCP support for FORTRAN IV desired?[1]

- Is SCP support for basic assembler desired?[1]

  If your response is YES, you are prompted for the following:
  - Are SCP base assembler macros desired?
  - Are SCP BSC assembler macros desired?
  - Are SCP scientific macros desired?

- Is SCP support for overlay linkage editor desired?[1]

- Is SCP support for queued job stream desired?

- Is there a PTF master diskette?

See index entry: *CNFIGSCP procedure* for the prompted parameter information you supply in step 10 of the System Configuration Steps.

Using the information supplied to the prompts, an SCP is built that contains the support that you request. If a PTF diskette is available, the PTFs are applied to the SCP on the disk.

---

[1] When you specify SCP support for either basic assembler or FORTRAN IV, SCP support for the overlay linkage editor is also provided. The prompt for SCP support for the overlay linkage editor does not appear.

[2] When you specify SCP support for word processing communications utility, you must also specify SCP support for data communications.

SYSTEM CONFIGURATION STEPS

CAUTION

The system configuration steps remove the current library (if any) from the disk. Save all library members you want to retain (see index entry: *FROMLIBR procedure*) before executing the system configuration steps. When installing version updates, if IBM program products are installed on the system, remove the IBM program products.

The program products can be removed by entering the following appropriate commands for the program product you have installed:

SEUDROP (if SEU is installed)

DFUDROP (if DFU is installed)

SORTDROP (if SORT is installed)

RPGDROP (if RPG is installed)

FCUDROP (if FCU is installed)

FORTDROP (if FORTRAN IV is installed)

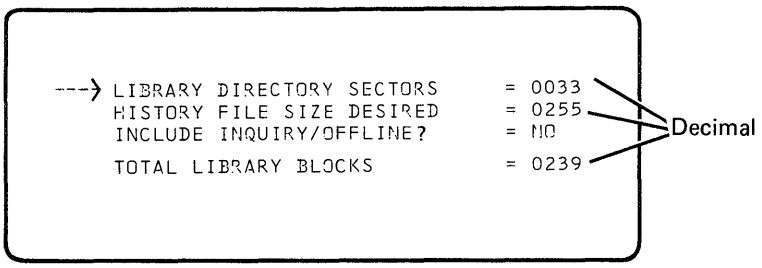
ASMDROP (if basic assembler is installed)

The system configuration steps are:

1. Set the IPL switch (on the CE control panel) to DISKETTE and set the IMPL switch (on the CE control panel) to DISK.

2. Insert the first PID SCP diskette.

3. Press the LOAD key on the operator panel. The following example display appears:



APPLYPTF PROCEDURE

The APPLYPTF procedure applies PTFs to the SCP and program products in the library. It is called by the CNFIGSCP procedure during system configuration, or directly, by the APPLYPTF command.

PTFs applied by the APPLYPTF procedure are read from the PTF diskette. If you apply SCP PTFs and you are not installing a new version of System/32, then you must make sure that your PTFs are placed on your tailored system SCP diskettes. This is done by the following:

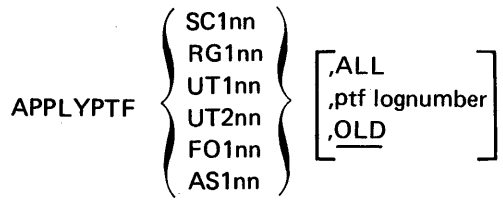
1. Apply the PTFs
2. Use the BACKUP procedure
3. Use the RELOAD procedure

The BACKUP procedure is used to obtain system diskettes that include the applied PTFs. The RELOAD procedure is used to maintain the correct library size. (See index entries: *RELOAD procedure* and *BACKUP procedure*.)

Note: Your PID SCP diskettes do not contain the applied PTFs.

The APPLYPTF procedure evokes the $MAINT utility (see index entry: *$MAINT utility program*).

APPLYPTF Command Statement Format



APPLYPTF Parameters

SC1nn	PTFs to change the SCP and SCP support for word processing are applied; nn is the version number of the system.
RG1nn	PTFs to change the RPG II program product are applied; nn is the version number of the program product.
UT1nn	PTFs that change the IBM System/32 utilities program product (DFU/SEU/Sort) are applied; nn is the version number of the utilities program product.
UT2nn	PTFs to change the IBM System/32 File Conversion Utility (FCU) program product are applied; nn is the version number of the program product.
FO1nn	PTFs to change the FORTRAN IV program product are applied; nn is the version number of the program product.
AS1nn	PTFs to change the basic assembler program product are applied; nn is the version number of the program product.
OLD	Apply only those PTFs from the PTF file that match the PTF members currently in the library. Any PTF members that do not currently exist in the library are not applied from the PTF file.
ALL	Apply all PTFs from the selected PTF file.
ptf log number	Apply only the PTF corresponding to this number. This number is the PTF log number and is indicated on the cover letter for each PTF. It is also indicated in the PTFXREF source member on each PTF diskette. To list the contents of this source member, enter the TOLIBR procedure (TOLIBR PTFXREF) and the LISTLIBR procedure (LISTLIBR PTFXREF). The ptf log number must be five digits (leading zeros are required).

CNFIGSCP PROCEDURE

The CNFIGSCP procedure is used for system configuration. It is distributed with each version of the system on an SCP diskette and is removed from the library after system configuration is complete. The CNFIGSCP procedure prompts you for the information it requires to build an SCP that contains the support you request. The information you supply to the prompts is recorded in the system. The system that you create, using the CNFIGSCP procedure, can be modified by using the $SETCF utility program as your requirements change.

The CNFIGSCP procedure evokes the $MAINT and $SETCF utilities (see index entries: *$MAINT utility program* and *$SETCF utility program*).

CNFIGSCP Command Statement Format

CNFIGSCP

To make your backup copy of the SCP, you must initialize two to five diskettes (number used depends on the optional SCP support you require). Use the INIT procedure (see index entry: *INIT procedure*) to do this. The diskettes can be initialized with the same volume identification, or each diskette can have a unique volume identification. The volume identification of the first diskette is the volume identification you must specify for the BACKUP procedure or error message 1493 appears when the second diskette is inserted. Take option 0 to continue processing.

After the diskettes are initialized, copy the SCP onto the diskettes. Use the BACKUP procedure (see index entry: *BACKUP procedure*) to make the backup copy.

Backup of Program Products

Before you install the selected program products and application programs with your configured SCP, apply any necessary PTFs to the program products you intend to install, and create a backup copy of each program product that you use.

Before a PTF can be applied to a program product, the program product must be in the library on the disk. See index entry: *program product installation* for a description of how to copy a program product to the library. After a program product is in the library, use the APPLYPTF procedure (see index entry: *APPLYPTF procedure*) to apply any necessary PTFs to the program product.

Create a backup copy of each program product after all PTFs, if any, are applied. See index entry: *backup copy of a program product* for a description of how to create a backup copy of a program product. The backup copies of the program products are then used during system installation to create the unique system you want to use.

Note: Even if no PTFs are applied, it is recommended that you make a backup copy of the program product PID distribution diskettes and use that copy when installing the program products with the SCP. The PID diskettes should be safely stored until the next version from PID is distributed.

System Configuration Error Messages

Several error messages are possible during system configuration.

INVALID VTOC/LIBRARY FOUND

Save all of the disk data files if you have not already done so (for information on how to save the disk data files, see index entry: *SAVE procedure*). After the files are saved, go back to system configuration Step 1.

If you have already saved, or if you do not want to save your data files, the following action deletes them and corrects the INVALID VTOC/LIBRARY error:

CAUTION
The following action deletes all data files from the disk.

- Hold down the SHIFT key and press the DUP key.
- Key a hyphen (-), then a plus (+); press the REC ADV key and check the display for any other error messages. If there are none, go to system configuration Step 5.

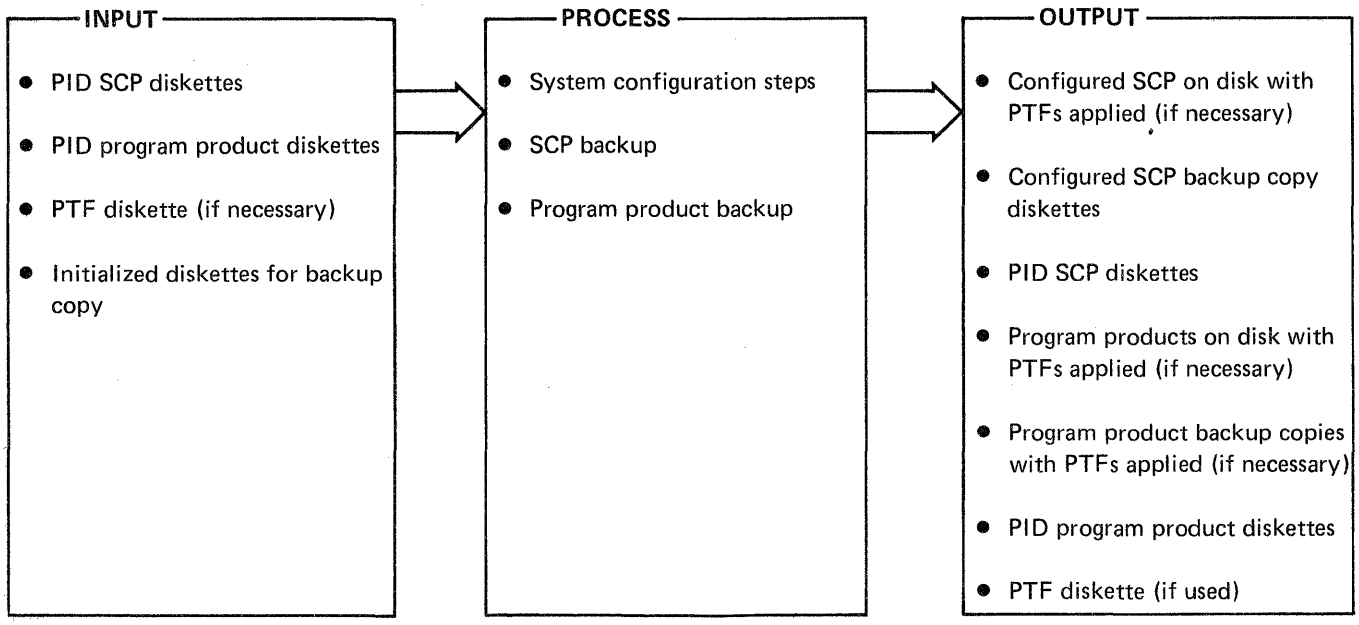
TOO MANY BLOCKS REQUESTED or INSUFFICIENT AVAILABLE SPACE

Perform the action for the INVALID VTOC/LIBRARY FOUND message, described in the preceding paragraphs, or go to system configuration Step 5 and decrease the library size.

Other Error Messages

Other system configuration error messages probably are the result of a mistake made in system configuration Step 5. Return to Step 5 and adjust the directory sector and library block allocations. Press the ENTER+ key after each entry for Step 5. Continue making entries for Step 5 until the error messages no longer appear. Then go to System Configuration Step 7.

SYSTEM CONFIGURATION SUMMARY



This section describes how to install program products and application programs with your configured SCP to create the unique system you want.

DISKETTES REQUIRED

The diskettes required for system installation are:

- Diskettes containing the SCP backup copy created by system configuration, as described in the preceding section.
- Diskettes containing backup copies of the program products you want to install with necessary PTFs included, or PID program product diskettes if backup copies are not available. Creation of program product backup copies should always be performed as part of system configuration, as described in the preceding section.
- Diskettes containing application programs you want to install.
- Backup diskettes onto which you can copy your entire system after it is installed.

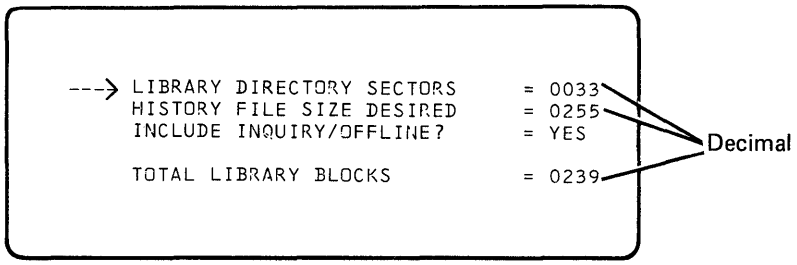
INFORMATION REQUIRED

If you intend to change the size of your current library during step 1 of System Installation, decide now exactly how many directory sectors and library blocks you want. See index entry: *system modification* for a description of library requirements in directory sectors and library blocks. You should also be familiar with the RELOAD display and how to change it. See index entry: *RELOAD display.*

To complete system installation you must use the information printed from the system directory during System Installation Step 6. See index entry: *printing from the library* for a description of the system directory information that is printed.

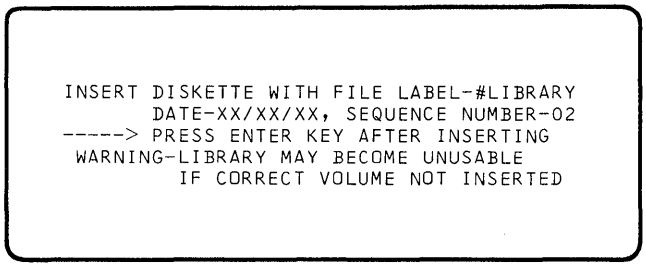
SYSTEM INSTALLATION STEPS

1. Insert the first SCP backup diskette created during the system configuration steps described in the preceding section. Enter the RELOAD command statement (see index entry: *RELOAD procedure*). When the following display appears, check the values displayed and change them, if necessary. See the preceding description of *Information Required* for index entries to determine and change the values displayed.

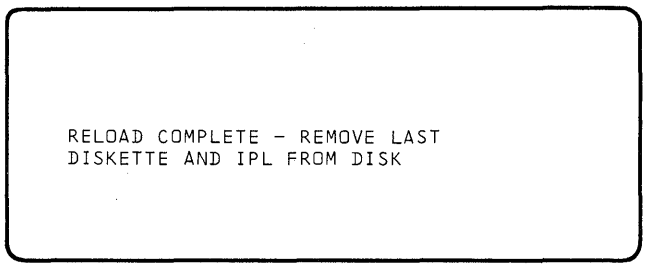


Note: The values shown in the preceding display are sample values only.

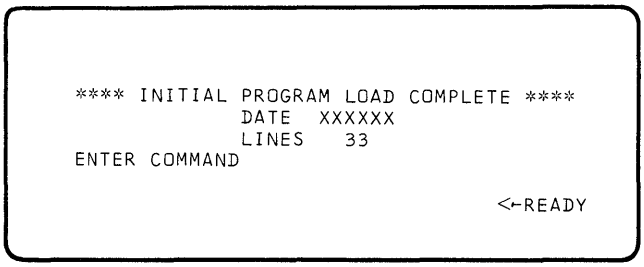
2. When the following display appears, insert the second SCP backup diskette.



3. When the following display appears, press the LOAD key.



When the LOAD key is pressed, the following display appears:



4. When the ENTER COMMAND message appears, enter the DATE command statement (see index entry: *DATE procedure*) to set the system date to the current date.

5. If you want to have any application programs on the backup copy of this system, they should be installed at this time. Library members of your application programs can be copied into the library using the TOLIBR procedure (see index entry: *TOLIBR procedure*). The letter accompanying each IBM Industry Application Program describes how to put these programs onto the system. Refer to that letter for installation instructions for an IBM Industry Application Program.

6. Enter the INSTALL command statement. The parameters you enter depend on the program products you want to install. See index entry: *INSTALL procedure* for a description of the command statement parameters.

   After the INSTALL command statement is entered, you are prompted for the diskettes that contain the program products to be installed. The diskettes you insert must be either the program product backup diskettes or, if backup copies are not available, the PID program product diskettes.

   After the INSTALL procedure is completed, the system prints system directory information. You will need this information for step 7. If you are not familiar with the kind of information in the system directory, see index entry: *printing from the library.*

   Note: After the INSTALL procedure is completed, it is deleted from the library. It remains on the PID SCP diskettes and the SCP backup copy created in system configuration.

7. You are prompted to initialize the number of diskettes needed to back up the system. See *Calculating the Number of Backup Diskettes Required for the System,* which follows, for a description of how to determine the number of diskettes you must initialize. If they are already initialized, you do not have to initialize them again. Otherwise, initialize them now.

   Note: The diskettes initialized in this step are only renamed, they are not formatted. That is, the effect is that of the INIT procedure with the RENAME, not the FORMAT or FORMAT2, parameter specified. If active files are on the diskettes to be initialized, they will be deleted (INIT procedure with DELETE parameter specified). For a description of the INIT procedure, see index entry: *INIT procedure.*

   As initialized diskettes are inserted, the system, composed of SCP, program products, and application programs, is copied on them. The final message will be SYSTEM INSTALLATION COMPLETE. If you want additional unique systems, repeat the system installation steps 1 through 7 as often as necessary.

CALCULATING THE NUMBER OF BACKUP DISKETTES REQUIRED FOR THE SYSTEM

To determine the number of diskettes required to make a backup copy of a system, you need system directory information. If you are installing a system, this information is printed at system installation, step 6. If you are modifying the system, you can have the system directory information printed by using the LISTLIBR procedure or the $MAINT utility program (see index entries: *LISTLIBR procedure* and *$MAINT utility program*—a sample of the information printed is given under index entry: *printing from the library*).

After printing the system directory information, determine the number of backup diskettes you need by following these steps:

1. Add decimal 23 to the decimal number of active directory entries.

2. Divide the result of step 1 by 11, rounding to the next highest number if you have a remainder, to determine the number of active directory sectors.

3. Add the result of step 2 to the decimal number of active library member sectors to determine the total library sectors referred to in the chart following step 4.

4. Use the result of step 3 and either table 1 or table 2 to determine the number of diskettes needed to contain your system. Use Table 1 for basic data exchange diskettes (128 bytes per sector). Use Table 2 for extended format diskettes (512 bytes per sector).

Table 1. Basic Data Exchange Diskettes (128 bytes per sector)

Total Library Sectors	Diskettes Required
906	1
1868	2
2830	3
3792	4
4754	5
5716	6
6678	7
7640	8
8602	9
9564	10
10526	11
11488	12
12450	13
13412	14
14372	15

Table 2. Extended Format Diskettes (512 bytes per sector)

Total Library Sectors	Diskettes Required
1128	1
2312	2
3496	3
4680	4
5864	5
7048	6
8232	7
9416	8
10600	9
11784	10
12968	11
14152	12
15336	13
16520	14
17704	15

PROGRAM PRODUCT INSTALLATION

The following list of IBM System/32 program products shows the diskette volume identification for each program product:

IBM System/32 Program Product	Diskette Volume Identification
Data File Utility (DFU)	PPUTIL
Source Entry Utility (SEU)	PPUTIL
Sort	PPUTIL
File Conversion Utility (FCU)	FCUFCU
RPG II	RPGRPG (distributed on two diskettes, each with the same vol-id)
Basic Assembler	PPASM
FORTRAN IV	PPFORT

The method for installing these program products individually and creating a backup copy of each is described here.

To Install a Program Product

1. If RPG II, basic assembler, or FORTRAN IV is to be installed, the SCP support for these program products must also be installed. First, insert the SCP diskette that contains the SCP support for the program product you are installing, and then enter the CNFIGSCP command statement. Answer the prompts according to the SCP support you need for the program product you will install in step 3.

2. Insert the appropriate PID program product diskette for the command to be entered in the following step (some program products may require more than one diskette).

3. Enter the TOLIBR command statement where the *filename* parameter is the identifier of the function being installed:

DFU	Data File Utility
SEU	Source Entry Utility
SORT	Sort
RPG	RPG II
FCU	File Conversion Utility
ASMCOMP	Basic Assembler
FORTRAN	FORTRAN IV

Note: For a description of the TOLIBR procedure, see index entry: *TOLIBR procedure.*

4. Enter nameLOAD (DFULOAD, SEULOAD, SORTLOAD, and RPGLOAD) for each function(s) being installed. This step does not apply to the FCU, ASM, and FORT functions. These three functions are completely installed in step 3.

5. If RPG II is being installed, message 1485 (END OF RD VOLUME–INSERT NEXT DISKETTE) appears after the first diskette is read. When the message appears, remove the first diskette, insert the second, and select option 0 to continue.

6. Apply any required PTFs to the program products you install. See index entry: *APPLYPTF procedure.*

To Create a Backup Copy of a Program Product

After a program product is installed, create a backup copy by following these two steps:

1. Initialize a diskette(s) with the appropriate volume identification to contain the copy.

Function to be Copied	Volume Identification
DFU/SEU/Sort	PPUTIL (one diskette)
RPG II	RPGRPG (two diskettes)
FCU	FCUFCU (one diskette)
Basic Assembler	PPASM (one diskette)
FORTRAN IV	PPFORT (one diskette)

Note: Use the INIT procedure to initialize diskettes–see index entry: *INIT procedure.*

2. Enter nameSAVE for each installed function to be saved, such as:

DFUSAVE
SEUSAVE
SORTSAVE
RPGSAVE
FCUSAVE
ASMSAVE
FORTSAVE

PROGRAM PRODUCT INSTALLATION VERIFICATION

You can verify the installation of SEU, RPG II, basic assembler, FORTRAN IV, and FCU.

SEU Installation Verification

1. Starting in column 1, key: SEU SEUTEST, R. Press the ENTER key. The display screen will appear as follows.

```
001         0    A096  0001.00       S
_

ENTER/UPDATE STATEMENT NUMBER:   0001.00
```

2. Starting in column 1, key: THIS WILL VERIFY THAT SEU IS INSTALLED. The display screen will appear as follows.

```
039         0    A096  0001.00       S
THIS WILL VERIFY THAT SEU IS INSTALLED_

ENTER/UPDATE STATEMENT NUMBER    0001.00
```

3. Press the ENTER key. The display screen will appear as follows.

```
001         0    A096  0002.00       S
_

ENTER/UPDATE STATEMENT NUMBER:   0002.00
```

4. Press the SELECT FORMAT command key. Key an F, then press the ENTER key. The display screen will appear as follows.

```
001         F     K005  0002.00       S
_     F

ENTER/UPDATE STATEMENT NUMBER:    0002.00
```

5. Press the REC ADV key. The display will flash and appear as follows.

```
PRESS ERROR RESET KEY TO CONTINUE
SEU 1002
FILENAME (POS 7-14) IS INVALID OR
SPECIFIED IMPROPERLY.
```

6. Press the ERROR RESET key and then the EOJ command key. The end of job options are displayed and the screen will appear as follows.

```
0 RETURN TO PROCESSING--NO EOJ
1 END OF JOB--NO ADDITIONAL OPTIONS
2 END OF JOB WITH LISTING
3 END OF JOB WITH SERIALIZATION
4 END OF JOB WITH LIST AND SERIALIZATION
END OF JOB OPTION:
```

7. Key a 2 and press the ENTER key. The statement you entered in step 3 (THIS WILL VERIFY THAT SEU IS INSTALLED) is printed if SEU is properly installed.

8. Enter the REMOVE command statement to remove from the library the member created to verify the SEU installation:

```
REMOVE SEUTEST,SOURCE
```

Prompted Parameters for CNFIGSCP

Belt Image Option

48
48HN
64
96

Sets the print belt image and its number of characters in the system configuration record, a record in the library directory that defines the system in terms of its components. A length of 48, 48HN, 64, or 96 can be entered.

Note: The serial printer requires a response of 64, the dual case keyboard requires a response of 96, and the special 48-character print belt requires a response of 48HN.

All remaining prompts and responses will be logged to the system printer.

Number of Lines Per Page Option

1 to 84 Sets the number of printed lines per page.

Note: The value commonly used is 66.

Date Format Option

YMD
MDY
DMY

Sets the system date format in the system configuration record: enter year-month-day (YMD), month-day-year (MDY), or day-month-year (DMY).

Notes:

1. Use yymmdd format if you are creating basic data exchange format diskettes to use with other systems.
2. Select the date format option to coincide with the format selected for the preceding version. Otherwise, there is a possibility that the date dependent output from an RPG II object program, using the UDAY, UMONTH, and UYEAR reserved fields, will be in error.

SCP Support For Data Communications

YES Data communication SCP support is copied.

NO Data communication SCP support is not copied.

BSC Support Option

YES Copies the optional BSC support.

NO BSC support is not copied.

MRJE Support Option

YES Copies the optional MRJE work station support.

NO MRJE work station support is not copied.

Batch Work Station Support Option

YES Copies the optional Batch Work Station support

NO Batch Work Station support is not copied

SCP Support For RPG

YES Copies the optional SCP support for RPG.

NO RPG SCP support is not copied.

Data Communication Support for RPG

YES Copies the data communication RPG support.

NO Data communication support for RPG is not copied.

SCP Support For Data Recorder Attachment

YES Copies the optional SCP support for the data recorder attachment.

NO Data recorder attachment SCP support is not copied.

SCP Support For Word Processing

YES Copies the optional SCP support for word processing.

NO Word processing SCP support is not copied.

Word Processing Communications Utility Option

YES Copies the optional word processing communications utility support.

NO Word processing communications utility support is not copied.

SCP Support For 1255 Magnetic Character Reader Attachment

YES Copies the optional SCP support for the 1255 Magnetic Character Reader attachment.

NO The 1255 Magnetic Character Reader attachment SCP support is not copied.

5.

SYSTEM DATE — CASH RECEIPTS REGISTER — PAGE 1

REGION	ACCOUNT NUMBER	ACCOUNT NAME	INVOICE NUMBER	INVOICE DATE	DATE PAID	AMOUNT OWED	DISCOUNT TAKEN	AMOUNT PAID	BALANCE DUE	EXCESS DISCOUNT
1	11243	JONES HARDWARE	27541	2/11/75	2/21/75	23.75	.47	23.28		
1	11352	NU-STYLE CLOTHIERS	27987	2/14/75	2/26/75	87.07		40.00	47.07	
1	11886	MIDI FASHIONS INC	15771	2/04/75	2/14/75	107.22	2.14	105.08		
1	12874	ULOOK INTERIORS	25622	2/09/75	2/23/75	67.95		67.95		
1	18274	STREAMLINE PAPER INC	29703	2/21/75	2/30/75	274.03	2.38	170.55	101.10	
					REGION TOTALS	560.02	4.99	406.86	148.17	
2	23347	RITE-BEST PENS CO	20842	2/18/75	2/20/75	15.80		10.00	5.80	
2	25521	IMPORTS OF NM	29273	2/20/75	2/27/75	797.40	11.93	585.47	200.00	
2	26723	ALRIGHT CLEANERS	19473	2/07/75	2/23/75	462.00		462.00		
2	28622	NORTH CENTRAL SUPPLY	17816	2/05/75	2/22/75	75.97		75.97		
2	29871	FERGUSON DEALERS	27229	2/10/75	2/22/75	61.91		61.91		
					REGION TOTALS	1,413.08	11.93	1,195.35	205.80	
3	30755	FASTWAY AIRLINES	26158	2/06/75	2/19/75	742.72	16.85	725.87		1.90
3	31275	ENVIRONMENT CONCERNS	20451	2/06/75	2/30/75	29.43		15.00	14.43	
3	32457	B SOLE SILOS	27425	2/10/75	2/20/75	110.05		110.05		
3	37945	HOFFTA BREAKS INC	18276	2/06/75	2/23/75	47.23		47.23		
					REGION TOTALS	929.43	16.85	898.15	14.43	1.90
4	42622	EASTLAKE GRAVEL CO	16429	2/05/75	2/23/75	29.37		29.37		
					REGION TOTALS	29.37		29.37		
					COMPANY TOTALS	2,931.90	33.77	2,529.73	368.40	1.90

FORTRAN IV Installation Verification

Sample program modules are provided with the IBM System/32 FORTRAN IV program product. When FORTRAN IV was installed, these modules were loaded from the PID program product distribution diskette (PPFORT) and are executed by entering the command statement FORTSMPL. This command statement causes two FORTRAN IV programs (KBINCO and SAMPLE) to be compiled, executed, and then deleted from disk. A program listing, compiler storage map, informational messages, overlay linkage editor map, and the output of the sample program are printed on the printer.

Figure 9 shows an example of the printed output from the FORTSMPL procedure using a 48-character FORTRAN print belt. Each compiler printed page heading shows the current version number, modification number, date, and page number. The sample includes:

- **A** The source module listing
- **B** The compiler storage map
- **C** The informational messages
- **D** The overlay linkage editor map
- **E** The sample program output

Prompted Parameters for INSTALL

Diskette Volume ID

`------` Enter a name with a maximum of six characters. This name is placed in the vol-id field on the diskettes if diskettes are initialized. It is also used as the vol-id parameter when the system is copied to the backup diskettes.

Diskettes to be Initialized

YES The diskette inserted is initialized using the vol-id specified. (After each diskette is initialized, you are prompted to insert the next diskette.

NO No diskettes need to be initialized. The INSTALL procedure copies the system onto diskettes that are already initialized. You are prompted for these diskettes.

This page intentionally left blank.

PROGRAM PRODUCT INSTALLATION

The following list of IBM System/32 program products shows the diskette volume identification for each program product:

IBM System/32 Program Product	Diskette Volume Identification
Data File Utility (DFU)	PPUTIL
Source Entry Utility (SEU)	PPUTIL
Sort	PPUTIL
File Conversion Utility (FCU)	FCUFCU
RPG II	RPGRPG (distributed on two diskettes, each with the same vol-id)
Basic Assembler	PPASM
FORTRAN IV	PPFORT

The method for installing these program products individually and creating a backup copy of each is described here.

To Install a Program Product

1. If RPG II, basic assembler, or FORTRAN IV is to be installed, the SCP support for these program products must also be installed. See index entry: *system installation*.

2. Insert the appropriate PID program product diskette for the command to be entered in the following step (some program products may require more than one diskette).

3. Enter the TOLIBR command statement where the *filename* parameter is the identifier of the function being installed:

DFU	Data File Utility
SEU	Source Entry Utility
SORT	Sort
RPG	RPG II
FCU	File Conversion Utility
ASMCOMP	Basic Assembler
FORTRAN	FORTRAN IV

Note: For a description of the TOLIBR procedure, see index entry: *TOLIBR procedure.*

4. Enter nameLOAD (DFULOAD, SEULOAD, SORTLOAD, and RPGLOAD) for each function(s) being installed. This step does not apply to the FCU, ASM, and FORT functions. These three functions are completely installed in step 3.

5. If RPG II is being installed, message 1485 (END OF RD VOLUME–INSERT NEXT DISKETTE) appears after the first diskette is read. When the message appears, remove the first diskette, insert the second, and select option 0 to continue.

6. Apply any required PTFs to the program products you install. See index entry: *APPLYPTF procedure.*

To Create a Backup Copy of a Program Product

After a program product is installed, create a backup copy by following these two steps:

1. Initialize a diskette(s) with the appropriate volume identification to contain the copy.

Function to be Copied	Volume Identification
DFU/SEU/Sort	PPUTIL (one diskette)
RPG II	RPGRPG (two diskettes)
FCU	FCUFCU (one diskette)
Basic Assembler	PPASM (one diskette)
FORTRAN IV	PPFORT (one diskette)

Note: Use the INIT procedure to initialize diskettes–see index entry: *INIT procedure.*

2. Enter nameSAVE for each installed function to be saved, such as:

DFUSAVE
SEUSAVE
SORTSAVE
RPGSAVE
FCUSAVE
ASMSAVE
FORTSAVE

Basic Assembler Installation Verification

A sample program (ASSMPL), input data file (INPUT), and procedure (ASMSAMPL) are provided with the IBM System/32 basic assembler program product. After basic assembler is installed, by entering the command statement ASMSAMPL, you will be prompted to insert the assembler program product diskette (PPASM).

```
ASMSAMPL
 INSERT ASSEMBLER PROGRAM PRODUCT
 DISKETTE.
ACTION  SCP 1162  CRPS  OPTIONS  0
PAUSE -- WHEN READY, ENTER 0 TO CONTINUE
```

The ASMSAMPL procedure will then copy to disk from diskette the ASSMPL source program and the input data file. The ASSMPL program will then be assembled, link edited, and executed.

```
ASSMPL WILL BE ASSEMBLED, LINKED,
AND EXECUTED.  AT EXECUTION TIME A
FILE WILL BE READ AND PUT TO THE
PRINTER.
ASM PROCEDURE EXECUTING
MACRO PROCESSOR EXECUTING
```

After execution, the ASSMPL source, object, and load modules, the input data file, and the ASMSAMPL procedure will be deleted from the disk.

The printed output from this verification sample is; a list of options, an external symbol list, source statement list, cross reference list, overlay linkage editor map, and the message THE ASSEMBLER SAMPLE PROGRAM IS EXECUTING PROPERLY. After this message is printed, the display screen will display EOF ON SYSIN and will then appear as below.

```
VERIFICATION IS COMPLETE.  THE
FOLLOWING WILL NOW BE DELETED
ASSMPL SOURCE, OBJECT, AND LOAD
MODULE - THE INPUT FILE - AND
THE ASMSAMPL PROCEDURE.
REMOVE PROCEDURE EXECUTING
```

The following is an example of the source statement listing and the final printed message of properly installed basic assembler program product.

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT      VER XX MOD XX   XX-XX-XX  PAGE   3

                                1         ICTL  1,71                                                    00020000
                                2         ISEQ  73,80                                                   00030000
                                3         PRINT NOGEN,NODATA                                            00040000
                                5 ************************************************************************ 00060000
                                6 * THIS PROGRAM READS A FILE FROM THE DISK AND LISTS IT                * 00070000
                                7 * ON THE PRINTER.                                                     * 00080000
                                8 *                                                                     * 00090000
                                9 *   THERE ARE THREE POSSIBLE MESSAGES ISSUED BY THIS PROGRAM:         * 00100000
                               10 *     MESSAGE                        MEANING                         * 00110000
                               11 *    'EOF ON SYSIN'          END OF FILE ENCOUNTERED FROM DISK READ.  * 00120000
                               12 *                             THE PROGRAM ISSUES THE MESSAGE         * 00130000
                               13 *                             AND GOES TO EOJ.                       * 00140000
                               14 *    'PRINTER ERROR'         THERE HAS BEEN A PERMANENT PRINTER      * 00150000
                               15 *                             ERROR.  THE PROGRAM ISSUES THE         * 00160000
                               16 *                             MESSAGE AND GOES TO END OF JOB.        * 00170000
                               17 *    'SYSIN ERROR'           THERE HAS BEEN A PERMANENT READ         * 00180000
                               18 *                             ERROR.  THE PROGRAM ISSUES THE         * 00190000
                               19 *                             MESSAGE AND GOES TO END OF JOB.        * 00200000
                               20 ************************************************************************ 00210000


0800                           22 ASSMPL START X'0800'                                                  00230000

                         0001  24         EXTRN #$CSIP                                                  00250000
                         0002  25         EXTRN #$BDMC                                                  00260000


                               27 * PREPARE THE FILES FOR USE (DTFS ARE CHAINED)                        00280000


                               29 *         $ALOC DTF-DSKDTF              ALLOCATE ALL FILES          00300000


                               35 *         $OPEN DTF-DSKDTF              OPEN ALL FILES              00320000


                               40 * READ FROM SYSTEM SOURCE LIBRARY AND PRINT RECORDS UNTIL END OF FILE 00340000
                         0812  41 REDAGN EQU   *                                                       00350000
                               42 *         $GETD ACCESS-CG,DTF-DSKDTF,ERR-SYSER,EOF-EOF               00360000


                               51 *         $PUTP DTF-PRTDTF,ERR-PRNERR,SPACEA-1,PRINT-Y               00380000


0340 C0 87 0812                61         B     REDAGN                 BRANCH BACK AND READ AGAIN      00400000
```

RPG II Installation Verification

Sample programs are provided with the IBM System/32 RPG II program product. After RPG II is installed, these programs can be loaded from the PID program product distribution diskette (RPGRPG) and executed by entering the command statement RPGSAMPL. This command statement causes three RPG II and two auto report programs to be compiled, executed, and then deleted from the disk. The first RPG II program prompts the operator as follows:

Prompt	Operator Response
KEY	123
DESC	DRESS
VALUEA	10
VALUEB	30
VALUEC	20
KEY	124
DESC	COAT
VALUEA	40
VALUEB	50
VALUEC	30

After the 10 fields are entered, the operator must press the CMD key and then the / key to indicate the end of input.

If RPG II is properly installed, printed output from the five sample programs is:

1.
```
NO TRANSACTIONS LOADED
 2 MASTERS LOADED
```

2.
```
                          IBM SYSTEM/32
SYSTEM DATE          SAMPLE UPDATE PROGRAM                PAGE 0001

                        NEW         NEW         NEW
KEY  DESCRIPTION    VALUE A     VALUE B     VALUE C

        NO TRANSACTION RECORDS ENTERED
```

3.
```
                          IBM SYSTEM/32
SYSTEM DATE      SAMPLE INDEXED FILE LISTING              PAGE 0001

KEY  DESCRIPTION  VALUE A    VALUE B    VALUE C   TOTAL A+B-C

123   DRESS          10         30         20         20
124   COAT           40         50         30         60

      FINAL TOTAL    50         80         50         80
```

4. DATA FOR SAMPLE PROGRAM

```
11243JONES HARDWARE       27541021175  2375CASH   47   47  2328022175
11352NU-STYLE CLOTHIERS   27987021475  8707CASH  174       4000022675
11886MIDI FASHIONS INC    15771020475 10722CASH  214  214 10508021475
12874ULOOK INTERIORS      25622020975  6795CASH  136       6795022375
18274STREAMLINE PAPER INC29703022175 27403      548  238 17055023075
23347RITE-BEST PENS CO    20842021875  1580       31       1000022075
25521IMPORTS OF NM        29273022075 79740     1593 1193 58547022775
26723ALRIGHT CLEANERS     19473020775 46200CASH  924      46200022375
28622NORTH CENTRAL SUPPLY17816020575  7597CASH  152       7597022275
29871FERGUSON DEALERS     27229021075  6191CASH  124       6191022275
30755FASTWAY AIRLINES     26158020675 74272CASH 1495 1685 72587021975
31275ENVIRONMENT CONCERNS20451020675  2943       59       1500023075
32457B SOLE SILOS         27425021075 11005CASH  220      11005022075
37945HOFFTA BREAKS INC    18276020675  4723CASH   94       4723022375
42622EASTLAKE GRAVEL CO   16429020575  2937CASH   58       2937022375
```

FCU Installation Verification

Two sets of sample data files and conversion specifications are provided with the IBM System/32 FCU program product. After the FCU is installed, either set of data files and conversion specifications can be loaded from the PID program product distribution diskette (FCUFCU) and executed by entering either of the following command statements:

```
FCUSAMPL DP
FCUSAMPL WP
```

Entering FCUSAMPL DP does the following operations:

- Loads a sample sequential file and specification source member from the diskette to disk.
- Executes the FCU specification phase to create a specification load member.
- Executes the FCU conversion phase to create an indexed sequential output file.
- Automatically deletes the FCUSAMPL DP procedure, the sample data files,the specification statements, and the load module.

The following is an example of the printed output of a properly installed FCU sample program.

```
FCU SPECIFICATION LISTING FOR MEMBER #FCUDP                          DATE XX/XX/XX

0001       FIS                                                          00020000
0002       FOI  40 6   1                                               00030000
0003       CP  1   1   6U0          1   1   6U0     ACCOUNT NUMBER      00040000
0004       CP  2   7   10P2         6   33  39U2    CURRENT BALANCE     00050000
0005       CP  3   11  14P2         5   26  32U2    NEW CHARGES         00060000
0006       CP  4   15  18P2         3   12  18U2    PAST DUE AMOUNT     00070000
0007       CP  5   19  22P2         4   19  25U2    PAYMENTS            00080000
0008       CP  6   23  25P0         2   7   11U0    CREDIT LIMIT        00090000
0009       CC                       7   40  40A   1 'DELETE FIELD'      00100000


      0305  SPECIFICATION LOAD MODULE CREATED
```

FCU CONVERSION PHASE PROCESSING MEMBER #FCUDP DATE XX/XX/XX

RECORD KEY 113520
1135200090000121760000000000126140024790

RECORD KEY 118860
1188600080000617350000200004172100834 56

RECORD KEY 953210
9532100050000117400011740000215000021 50

RECORD KEY 233470
2334700090000637850000400001753000413 15

RECORD KEY 286220
2862200050000671410067141005194000519 40

RECORD KEY 825130
8251300300003196670157950031179304735 10

RECORD KEY 312750
3127500090000777600050000005399700817 57

RECORD KEY 324570
3245700040000532000032610002914000497 30

RECORD KEY 298710
2987100090000421360042136003749100374 91

RECORD KEY 437150
4371500080000731910000400000000000331 91

RECORD KEY 439370
4393700080000093100005620000413000078 20

RECORD KEY 451370
4513700050000197170019717002233700223 37

RECORD KEY 469180
4691800100000682350000300006319401014 29

RECORD KEY 583130
5831301000003374150311471005739100833 35

RECORD KEY 791190
7911900080000217190021719001174500117 45

RECORD KEY 913700
9137000080000549730000400007417000891 43

RECORD KEY 987160
9871600080000015420001542008858500885 85

RECORD KEY 307550
3075500060000078160007816006350000635 00

Entering FCUSAMPL WP does the following operations:

- Loads sample sequential and indexed data files and a specification source member from the diskette to the disk.
- Executes the FCU specification phase to create a specification load module.
- Executes the FCU conversion phase to create a tabular document in a document library.
- Automatically deletes the FCUSAMPL WP procedure, the sample data files, the specification statements, and the load module.

Note: This sample requires that System/32 SCP Feature Number 6002 (word processing support) and a 96-character print belt be installed.

The following is an example of the printed output of a properly installed FCU sample program.

```
FCU SPECIFICATION LISTING FOR MEMBER #FCUWP                                DATE XX/XX/XX

0001      FIS                                                                00020000
0002      FSI                                                                00030000
0003      FOL                FCU     WPSAMPLE                                00040000
0004      Q   100GTF110                                                      00050000
0005      CK  1   1   6U0                                 KEY FIELD          00060000
0006      CS 10   1   6A          1       A             CHARGE #           00070000
0007      CS 20  33  33A  TITLES  2       A             TITLE              00080000
0008      CS 30  22  31A          3       A P           FIRST NAME         00090000
0009      CS 40  32  32A          4       A U           MIDDLE INITIAL     00100000
0010      CS 50   7  21A  NAMES   5       A P           LAST NAME          00110000
0011      CS 60  34  53A  NSEW    6       A P           STREET             00120000
0012      CS 70  54  73A          7       A P           CITY               00130000
0013      CS 80  74  75A  STATES  8       A             STATE NAME         00140000
0014      CS 90  76  80A          9       A             ZIP CODE           00150000
0015      CC                     10       A     1$      DOLLAR SIGN        00160000
0016      CP100   7  10P2S      11       D2            CURRENT BALANCE    00170000
0017      CC                     12       A     1$      DOLLAR SIGN        00180000
0018      CP110  23  25P0       13       D0            CREDIT LIMIT       00190000
0019      ATITLES 7 71013                                                  00200000
0020      .1  Mr.                                                          00210000
0021      .2  Ms.                                                         00220000
0022      .3  Mrs.                                                        00230000
0023      .4  Miss                                                        00240000
0024      .5  Dr.                                                         00250000
0025      ANAMES  7101517*                                                 00260000
0026      .MC*     Mc*                                                     00270000
0027      .O'*     O'*                                                     00280000
0028      ANSEW   7101316*                                                 00290000
0029      .NE *  NE *                                                      00300000
0030      .NW *  NW *                                                      00310000
0031      .SE *  SE *                                                      00320000
0032      .SW *  SW *                                                      00330000
0033      ASTATES 7 81335                                                  00340000
0034      .OH    Ohio                                                      00350000
0035      .FL    Florida                                                   00360000
0036      .AL    Alabama                                                   00370000
0037      .GA    Georgia                                                   00380000
0038      .LA    Louisiana                                                 00390000
0039      .SC    South Carolina                                            00400000
0040      .MS    Mississippi                                               00410000
0041      .WV    West Virginia                                             00420000
0042      .MD    Maryland                                                  00430000
0043      .IL    Illinois                                                  00440000
0044      .DC    D. C.                                                     00450000
0045      .KY    Kentucky                                                  00460000


      0305  SPECIFICATION LOAD MODULE CREATED
```

```
 FCU CONVERSION PHASE PROCESSING MEMBER #FCUWP                                  DATE XX/XX/XX

 RECORD NO  000001
118860   Barbara      McGuire          470 Live Oak Place   Albany               Georgia
      $ 800.

 RECORD NO  000002
286220 Mr.  Joseph     A Abruzzo          3500 Gault Ocean Dr  New Orleans          Louisiana
        $ 500.

 RECORD NO  000003
825130                 A-1 Used Cars    200 SE 124 St.       Maywood              Illinois
      $ 3,000.

 RECORD NO  000004
324570 Mr.  Robert     Q Dobbs            Buttonwood Drive     Rome                 Georgia
        $ 400.

 RECORD NO  000005
469180 Miss Margaret   E Monroe           9 Pine Tree Lane     Sunny South          Alabama
        $ 1,000.

 RECORD NO  000006
913700 Ms.  Janice     L Comstock         2637 Marion Dr       Ellensburg           D. C.
        $ 800.

 RECORD NO  000007
987160 Mr.  Charles    N McCall           669 W Campus Circle  Williston            South Carolina
        $ 800.

 RECORD NO  000008
307550   Horace      M De Angelo        8150 Cypress Road    Everglades           Florida
      $ 600.

          FIELD  100      SUM =              10012.93      MAX =          4735.10      MIN =
```

This page intentionally left blank

Some members can be deleted from the system library to release library space for other members or to make disk space available for data files by reducing the size of the library. You can also delete program products from the library.

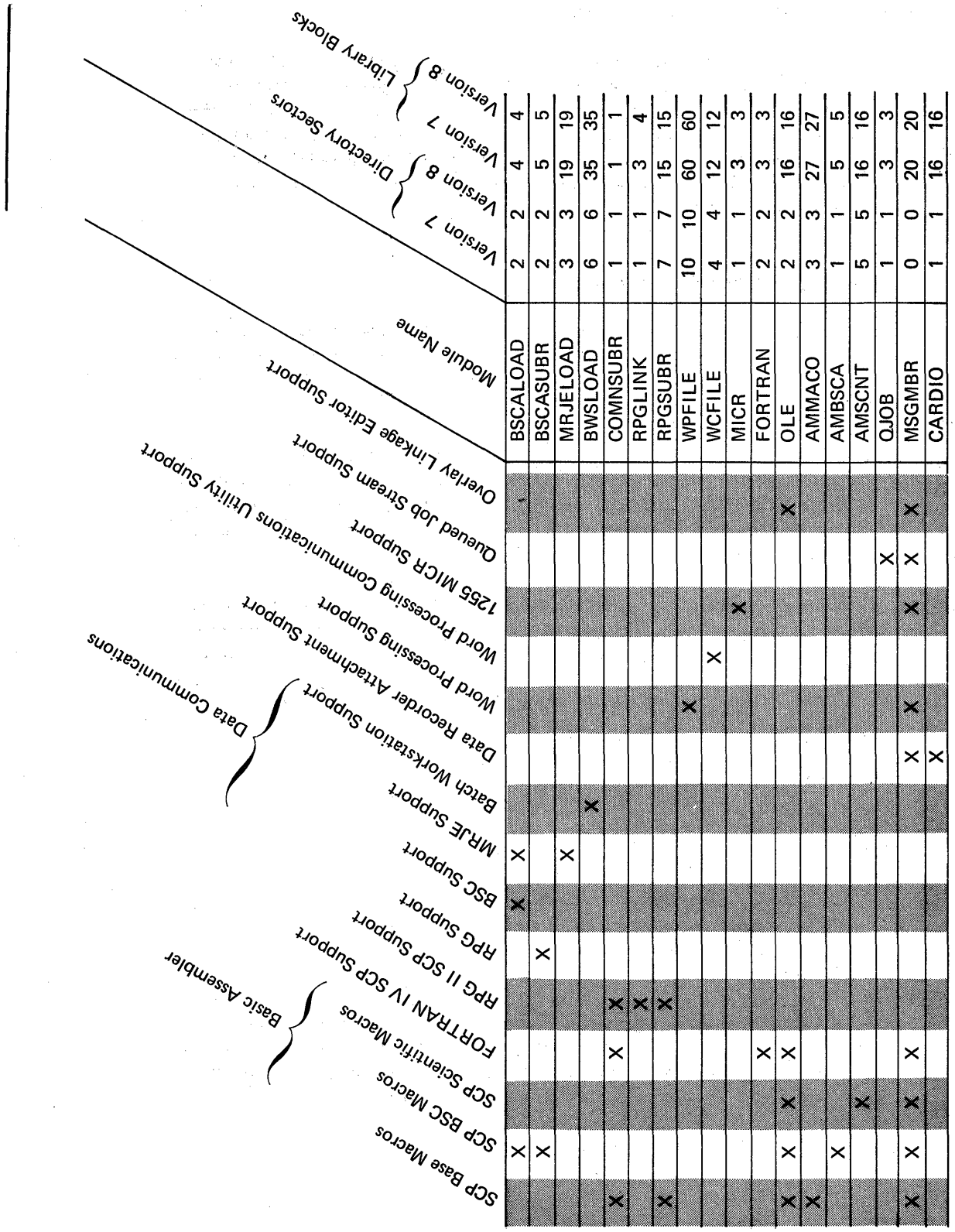
LIBRARY REQUIREMENTS

The library requirements of the minimum IBM System/32 system control programming are fixed at 33 directory sectors and 239 total library blocks. Control storage increment feature support adds four library blocks. Inquiry/offline support adds 11 library blocks to a 16K system (K = 1024 bytes), 14 library blocks to a 24K system, and 17 blocks to a 32K system.

In addition to the preceding minimum SCP library requirements, the requirements for DFU, SEU, SORT, FCU, RPG, FORTRAN IV, and basic assembler program products are shown in the following chart. The version 6 and version 7 columns show the change (if any) in the requirements from the previous version.

Library Function	Directory Sectors		Library Blocks	
	Version 7	Version 8	Version 7	Version 8
DFU	4	4	36	36
SEU	4	4	38	38
SORT	5	5	34	34
FCU	4	4	42	42
RPG II	16	16	147	147
FORTRAN IV	19	19	96	96
Basic Assembler	4	4	27	27

Use the following chart to determine how many directory sectors and library blocks must be added for the program products and SCP support needed for your system. If more than one type of support (program product and SCP) requires the same module, you only need to add that module once.


Module Name
Version 7
Version 8
Version 7
Version 8
Version 8
Directory Sectors
Library Blocks
Overlay Linkage Editor Support
Queued Job Stream Support
1255 MICR Support
Word Processing Communications Utility Support
Word Processing Support
Data Recorder Attachment Support
Batch Workstation Support
Data Communications
MRJE Support
BSC Support
RPG Support
RPG II SCP Support
FORTRAN IV SCP Support
SCP Scientific Macros
SCP BSC Macros
SCP Base Macros
Basic Assembler
BSCALOAD 2 2 4 4
BSCASUBR 2 2 5 5
MRJELOAD 3 3 19 19
BWSLOAD 6 6 35 35
COMNSUBR 1 1 1 1
RPGLINK 1 1 3 4
RPGSUBR 7 7 15 15
WPFILE 10 10 60 60
WCFILE 4 4 12 12
MICR 1 1 3 3
FORTRAN 2 2 3 3
OLE 2 2 16 16
AMMACO 3 3 27 27
AMBSCA 1 1 5 5
AMSCNT 5 5 16 16
QJOB 1 1 3 3
MSGMBR 0 0 20 20
CARDIO 1 1 16 16

Example:

A user required the SEU program product, data recorder support, word processing support, and inquiry/offline support on a 16K system.

Directory Sectors	Library Blocks	Library Function
33	239	Minimum System/32 SCP
4	38	SEU
	11	Inquiry/offline support on 16K
1	16	CARDIO (data recorder)
6	60	WPFILE (word processing)
	20	MSGMBR (required by both word processing and data recorder but needs to be added only once)

The total number of directory sectors is 44; the total number of library blocks is 384. These totals are the minimum numbers of directory sectors and library blocks for the requested system.

DELETING FROM THE LIBRARY

Before deleting members from the library, determine how much space is presently available for new members, or how much disk space is available for additional data files.

This page intentionally left blank

Determining Space Available in the Library

To determine how much space is available in the library, use the LISTLIBR procedure or the copy function of the $MAINT utility to print the system information from the directory area (see index entries: *$MAINT utility program* and *LISTLIBR procedure*). The system information listed will specify the number of additional entries the directory can contain (AVAILABLE DIRECTORY ENTRIES) and how many sectors are available in the library for additional members (AVAILABLE MEMBER SECTORS).

Determining Space Available on the Disk

To determine how much space exists on the disk for additional data files, use the CATALOG procedure or the $LABEL utility (see index entries: *$LABEL utility program* and *CATALOG procedure*) to display the disk VTOC. Available disk space is specified in every disk VTOC display.

Note: You can also use CATALOG or $LABEL to display all disk VTOC entries to determine which files can be deleted (see index entries: *$DELET utility program* and *DELETE procedure*). Use the COMPRESS procedure or $FREE utility (see index entries: *$FREE utility program* and *COMPRESS procedure*) to collect unused disk space in one area.

To determine how much space will be available for user programs and data files, take the total library requirements of your planned system and subtract this number from the number of disk blocks on your system. (see index entry: *library requirements*)

Note: Convert the sectors to blocks (1 block equals 10 sectors). If there is a remainder, round off that remainder to the next whole number.

Disk blocks available on the IBM System/32 are:

1248 blocks on a 3.2 megabyte disk

1968 blocks on a 5.0 megabyte disk

3576 blocks on a 9.1 megabyte disk

5376 blocks on a 13.7 megabyte disk

Example: The library requirements of the minimum IBM System/32 system control programming are 33 directory sectors and 239 library blocks. This totals 243 blocks (33 sectors converted to blocks rounds to 4 blocks). A 3.2 megabyte disk system leaves 1005 blocks available for user programs and data files.

1248 (blocks on a 3.2 megabyte disk)
-243 (total blocks library requirements)
1005 (total blocks available to the user)

Selecting Members to Delete

The following members can be deleted from the library without affecting other members or SCP functions:

Name	Member Type	Description
##MSG1	0 (load)	Level 1 error messages
##MSG4	0 (load)	Level 2 error messages
Selected procedure (see note)	P (procedure)	Procedures

Note: When you delete a library member, be sure not to delete a procedure within a nested procedure(s) or a procedure called by all procedures. For example, #ERR is a nested procedure available to all procedures for error detection. If ##MSG1 and/or ##MSG4 are deleted, there will be no message text when an error occurs.

In addition to deleting the preceding members you can delete inquiry/offline multivolume support and any program product installed on the system without affecting other system functions.

Deleting (or not including) inquiry/offline support (INCLUDE INQUIRY/OFFLINE? = NO on the RELOAD display—see index entry: *RELOAD display*) saves 11 blocks of library space on a 16K system, 14 blocks on a 24K system, and 17 blocks on a 32K system. Use the LISTLIBR procedure or the copy function of $MAINT to list library directory entries to determine space gained by deleting procedure members, ##MSG1, and ##MSG4. See index entry: *library requirements* to see how much space is gained by deleting a program product.

Note: After deleting members, use the CONDENSE procedure to collect all available space into one area at the end of the library.

Deleting Members

- ##MSG1, ##MSG4, and procedure members are deleted by using the delete function of $MAINT. See index entry: *$MAINT utility program.*

- Inquiry/offline support is deleted by specifying NO to the INQUIRY/OFFLINE option of the RELOAD display. The RELOAD display is described in following paragraphs.

- Program products are deleted by entering a nameDROP command statement for each function to be deleted (DFUDROP, SEUDROP, SORTDROP, RPGDROP, ASMDROP, FORTDROP, and/or FCUDROP). The procedures evoked by these command statements are deleted from the system when the related program product functions are deleted.

After you have deleted members, you can change space allocated to the library by using the RELOAD display, described in the following paragraphs.

Notes:
1. Do not delete any procedure that is used by a procedure that you are not deleting.
2. To gather the disk space created by deleting members from the library into one usable area, you can use the CONDENSE procedure. See index entry: *CONDENSE procedure.*

RELOAD DISPLAY

The RELOAD procedure (described under index entry: *RELOAD procedure*) is used to perform an IPL from diskettes onto which the library was copied by the BACKUP procedure (described under index entry: *BACKUP procedure*). RELOAD creates a new library on the disk, but does not disturb data files on the disk.

The RELOAD display appears when you insert the first backup diskette (for a particular copy of the library) and enter the RELOAD command statement (described under index entry: *RELOAD command statement*) or when you press the LOAD key with the IPL switch on the CE control panel set to DISKETTE.

The RELOAD display shows the number of sectors allocated for the library directory, indicates whether or not inquiry or offline multivolume files are supported, and shows the total number of blocks allocated for the library (system file #LIBRARY). A sample display follows:

```
----> LIBRARY DIRECTORY SECTORS   = 0033
      INCLUDE INQUIRY/OFFLINE?    = NO
      TOTAL LIBRARY BLOCKS        = 0239
```

This page intentionally left blank

```
000 TOTAL ERRORS FOR THIS COMPILATION
```

B

```
      STATEMENT ALLOCATIONS
5    =056A  4    =0583  3    =05C4  2    =0654  10   =0693  20   =070F  30   =0747
```

```
                OVERLAY LINKAGE EDITOR STORAGE USAGE MAP  XX/XX/XX

 START     OVERLAY   CATEGORY  NAME AND      CODE LENGTH
ADDRESS  NUMBER AREA            ENTRY    HEXADECIMAL DECIMAL

 0800                    0      SAMPLE          0764    1892
 0F5C                           #UNITB
 0908                           #ERBUF
 0884                           #IOBUF
 0F64                    0      @FOE0           013D     317
 1045                           #MNTRY
 1078                           #SNTRY
 107E                           #RNTRY
 0F64                           #D
 1081                           #RETRN
 1084                           DLDIRG
 1084                           RESUME
 10A1                    0      @FOB1           0066     102
 10D5                           #DE04
 10F7                           #DEDZ0
 1107                    0      @FOI0           010F     271
 11A1                           #ELST
 11AD                           #ELST2
 118F                           #DERR
 117D                           #IOINT
 1169                           #IOCOM
 118A                           #ENDEQ
 11C5                           #ERREQ
 11DE                           #OUTBL
 11D0                           #INTBL
 11EC                           #IO@@@
 11ED                           #FLRP2
 1216                    4      @FOVC           001C      28
 1222                           #FLOAT
 1232                    4      @FOVA           0007       7
 1239                    5      @FOB2           013A     314
 1340                           #FRET
 1348                           @FOB2A
 12C0                           @FOB2B
 1352                           @FOB2C
 1373                    5      @FOC3           00AD     173
 1420                    5      @FOB8           004E      78
 146E                    5      @FOB9           0018      24
 1486                    5      @FOBA           001D      29
 14A3                    5      @FOCA           002B      43
 14CE                    6      @FOIC           00D3     211
 1573                           #ERTST
 15A1                    6      @FOIB           00D8     216
 1679                    6      @FOI3           00CB     203
 167F                           #FOI3A
 1714                           #FOI3
 1744                    6      @FOVP           0019      25
 1750                    6      @FOBB           0038      56
 1795                    6      @FOD7           012C     300
 18C1                   20      KBINCO          01D3     467
```

D

Figure 9 (Part 3 of 4). FORTRAN IV Verification Sample Program Output

C

```
3130 I SAMPLE MODULE'S TOTAL MAIN STORAGE SIZE IS
          4756 DECIMAL
3131 I 0800 IS THE START CONTROL ADDRESS OF THIS MODULE
3134 I SAMPLE MODULE IS CATALOGED INTO THE LIBRARY WITH THE FOLLOWING INFORMATION
         19 TOTAL NUMBER OF LIBRARY SECTORS
```

E

SYSTEM/32 FORTRAN IV SAMPLE TEST CASE

		K									
N		1	2	3	4	5	6	7	8	9	10
1		1	0	0	0	0	0	0	0	0	0
2		2	1	0	0	0	0	0	0	0	0
3		3	3	1	0	0	0	0	0	0	0
4		4	6	4	1	0	0	0	0	0	0
5		5	10	10	5	1	0	0	0	0	0
6		6	15	20	15	6	1	0	0	0	0
7		7	21	35	35	21	7	1	0	0	0
8		8	28	56	70	56	28	8	1	0	0
9		9	36	84	126	126	84	36	9	1	0
10		10	45	120	210	252	210	120	45	10	1
11		11	55	165	330	462	462	330	165	55	11
12		12	66	220	495	792	924	792	495	220	66
13		13	78	286	715	1287	1716	1716	1287	715	286
14		14	91	364	1001	2002	3003	3432	3003	2002	1001
15		15	105	455	1365	3003	5005	6435	6435	5005	3003
16		16	120	560	1820	4368	8008	11440	12870	11440	8008
17		17	136	680	2380	6188	12376	19448	24310	24310	19448
18		18	153	816	3060	8568	18564	31824	43758	48620	43758
19		19	171	969	3876	11628	27132	50388	75582	92378	92378
20		20	190	1140	4845	15504	38760	77520	125970	167960	184756

Figure 9 (Part 4 of 4). FORTRAN IV Verification Sample Program Output

Basic Assembler Installation Verification

A sample program (ASSMPL), input data file (INPUT), and procedure (ASMSAMPL) are provided with the IBM System/32 basic assembler program product. After basic assembler is installed, by entering the command statement ASMSAMPL, you will be prompted to insert the assembler program product diskette (PPASM).

```
ASMSAMPL
 INSERT ASSEMBLER PROGRAM PRODUCT
 DISKETTE.
ACTION  SCP 1162  CRPS  OPTIONS  0
PAUSE -- WHEN READY, ENTER 0 TO CONTINUE
```

The ASMSAMPL procedure will then copy to disk from diskette the ASSMPL source program and the input data file. The ASSMPL program will then be assembled, link edited, and executed.

```
ASSMPL WILL BE ASSEMBLED, LINKED,
AND EXECUTED.  AT EXECUTION TIME A
FILE WILL BE READ AND PUT TO THE
PRINTER.
ASM PROCEDURE EXECUTING
MACRO PROCESSOR EXECUTING
```

After execution, the ASSMPL source, object, and load modules, the input data file, and the ASMSAMPL procedure will be deleted from the disk.

The printed output from this verification sample is; a list of options, an external symbol list, source statement list, cross reference list, overlay linkage editor map, and the message THE ASSEMBLER SAMPLE PROGRAM IS EXECUTING PROPERLY. After this message is printed, the display screen will display EOF ON SYSIN and will then appear as below.

```
VERIFICATION IS COMPLETE.  THE
FOLLOWING WILL NOW BE DELETED
ASSMPL SOURCE, OBJECT, AND LOAD
MODULE - THE INPUT FILE - AND
THE ASMSAMPL PROCEDURE.
REMOVE PROCEDURE EXECUTING
```

The following is an example of the source statement listing and the final printed message of properly installed basic assembler program product.

```
ASSMPL DISK FILE TO PRINTER (80/80 LIST PROGRAM)

ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT      VER XX MOD XX   XX-XX-XX  PAGE   3

                                1         ICTL  1,71                                                    00020000
                                2         ISEQ  73,80                                                   00030000
                                3         PRINT NOGEN,NODATA                                            00040000
                                5 ************************************************************************ 00060000
                                6 * THIS PROGRAM READS A FILE FROM THE DISK AND LISTS IT                * 00070000
                                7 * ON THE PRINTER.                                                     * 00080000
                                8 *                                                                     * 00090000
                                9 *   THERE ARE THREE POSSIBLE MESSAGES ISSUED BY THIS PROGRAM:         * 00100000
                               10 *     MESSAGE                          MEANING                       * 00110000
                               11 *    'EOF ON SYSIN'          END OF FILE ENCOUNTERED FROM DISK READ.  * 00120000
                               12 *                             THE PROGRAM ISSUES THE MESSAGE         * 00130000
                               13 *                             AND GOES TO EOJ.                       * 00140000
                               14 *    'PRINTER ERROR'         THERE HAS BEEN A PERMANENT PRINTER      * 00150000
                               15 *                             ERROR.  THE PROGRAM ISSUES THE         * 00160000
                               16 *                             MESSAGE AND GOES TO END OF JOB.        * 00170000
                               17 *    'SYSIN ERROR'           THERE HAS BEEN A PERMANENT READ         * 00180000
                               18 *                             ERROR.  THE PROGRAM ISSUES THE         * 00190000
                               19 *                             MESSAGE AND GOES TO END OF JOB.        * 00200000
                               20 ************************************************************************ 00210000

0800                           22 ASSMPL START X'0800'                                                   00230000

                     0001      24         EXTRN #$CSIP                                                  00250000
                     0002      25         EXTRN #$BDMC                                                  00260000

                               27 * PREPARE THE FILES FOR USE (DTFS ARE CHAINED)                         00280000

                               29 *         $ALOC DTF-DSKDTF              ALLOCATE ALL FILES          00300000

                               35 *         $OPEN DTF-DSKDTF              OPEN ALL FILES              00320000

                               40 * READ FROM SYSTEM SOURCE LIBRARY AND PRINT RECORDS UNTIL END OF FILE  00340000
                     0812      41 REDAGN EQU   *                                                         00350000
                               42 *         $GETD ACCESS-CG,DTF-DSKDTF,ERR-SYSER,EOF-EOF                 00360000

                               51 *         $PUTP DTF-PRTDTF,ERR-PRNERR,SPACEA-1,PRINT-Y                 00380000

0340 C0 87 0812                61         B     REDAGN                  BRANCH BACK AND READ AGAIN      00400000
```

Version Update Instruction Summary

The following instructions are intended to be used as a guide for installing a version update on an IBM System/32. These instructions are a summary of the detailed instructions that are presented in Part 5, *System Configuration, Installation, and Modification.* Index entries follow each step for the detailed description.

Note: Your IBM service representative can tell you if there are any PTFs applicable to your version of the SCP, or to your version of any program product. If there are PTFs, make arrangements with your IBM representative to have the PTF diskette available when you do your version update. The PTF diskette contains all applicable PTFs.

To install a version update on the IBM System/32 execute the following steps:

1. Print the system information from the system library to determine the total number of library blocks, the directory size, and if you are using inquiry/offline. Save the printed listing for step 6.

   Enter: LISTLIBR DIR,SYSTEM

   (See index entries: *printing from the library* and *LISTLIBR procedure.*)

2. Delete IBM program products that are installed on the system so that the system contains only user programs. User programs are saved in step 3.

   Enter the following appropriate command for the program product you have installed:

   SEUDROP (if SEU is installed)
   DFUDROP (if DFU is installed)
   SORTDROP (if SORT is installed)
   RPGDROP (if RPG is installed)
   FCUDROP (if FCU is installed)
   FORTDROP (if FORTRAN IV is installed)
   ASMDROP (if basic assembler is installed)

   (See index entry: *deleting members.*)

3. Save your user programs on a diskette file (filename used here is USERLIBR). The USERLIBR file is restored to disk in step 9.

   a. Initialize enough diskettes to contain your user programs (vol-id used here is USER).

      Note: FORMAT2 may require fewer diskettes.

      Enter: INIT USER,,FORMAT2

      Note: Files that are on the diskettes being initialized in this step are deleted, so make sure these files are not needed. (See index entry: *INIT procedure.*)

   b. Use the diskettes initialized in part *a* of this step to save your user programs.

      Enter: FROMLIBR ALL,LIBRARY,USERLIBR,,999,USER

      (See index entry: *FROMLIBR procedure.*)

      Note: If you have not initialized enough diskettes, return to part *a* of this step and initialize more diskettes (the diskettes already used in part *b* of this step must be deleted.) (See index entry: *INIT procedure.*)

4. List the disk VTOC and save this list to compare with the list that will be printed in step 10 to verify that no data files were lost.

   Enter: CATALOG

   (See index entry: *CATALOG procedure.*)

5. Save your data files on a diskette file.

   a. Initialize enough diskettes to contain your data files (vol-id used here is DFSAVE).

      Note: FORMAT2 may require fewer diskettes.

      Enter: INIT DFSAVE,,FORMAT2

      Note: Files that are on the diskettes being initialized in this step are deleted, so make sure that these files are not needed.

      (See index entry: *INIT procedure.*)

   b. Use the diskettes initialized in part *a* of this step to save your data files.

      Enter: SAVE ALL,,,DFSAVE

      (See index entry: *SAVE procedure.*)

      Note: If you have not initialized enough diskettes, return to part *a* of this step and initialize more diskettes (the diskettes already used in part *b* of this step must be deleted.) See index entry: *INIT procedure.*

6. Install the version update of system control programming.

   Enter: RELOAD

   (See index entry: *RELOAD procedure.*)

   Note: The values used on your last version for the RELOAD display are in the list printed in step 1. These values may have to be increased if additional optional functions or program products are being added on this version or if the library requirements have increased from the last version (see index entry: *library requirements*).

   If inquiry/offline support was included on your last version, the list printed in step 1 will include an inquiry/offline area.

7. When the message ENTER COMMAND appears, start the system configuration.

   a. Enter: CNFIGSCP

      (See index entry: *CNFIGSCP procedure.*)

   b. Follow the instructions on the display screen and respond to the prompts.

      Note: If you have a PTF diskette, add the PTFs when prompted.

      (See index entry: *APPLYPTF procedure.*)

   c. When the message SYSTEM CONFIGURATION COMPLETE REMOVE DISKETTE AND IPL FROM DISK appears, ensure that both the IPL and IMPL switches are set to DISK and press the LOAD key. The version update is now loaded and the configuration is complete.

8. Install the program products that you wish to have on your system.

   a. Enter: INSTALL [DFU] [,SEU] [,SORT] [,RPG] [,FCU] [,FORT] [,ASM]

      (See index entry: *INSTALL procedure.*)

   b. Insert the PID program product diskette prompted for and follow the instructional messages that are displayed.

   c. When the prompt for the volume-id of the backup diskettes is displayed, press the INQ key and select option 2. This terminates the INSTALL procedure.

      Note: If you have a PTF diskette, add the program product PTFs at this time.

      (See index entry: *APPLYPTF procedure.*)

9. Insert the USER diskettes that were used in step 3 to restore user programs.

   Enter: TOLIBR USERLIBR

   (See index entry: *TOLIBR procedure.*)

10. List the disk VTOC and compare this list to the list printed in step 4.

    Enter: CATALOG

    (See index entry: *CATALOG procedure.*)

    Note: If the lists are the same (your data files were not affected by the version update) go to step 12. If the lists are not the same go to step 11 to restore your data files.

11. Restore all data files saved in step 5.

    Enter: RESTORE

    (See index entry: *RESTORE procedure.*)

12. Backup your complete system. The diskettes used in steps 3 and 5 are no longer needed and may be used here (delete and rename them). (See index entry: *INIT procedure.*)

    a. Initialize enough diskettes to contain your complete system (vol-id used here is SYSTEM).

       Note: FORMAT2 may require fewer diskettes.

       Enter: INIT SYSTEM,,FORMAT2

       Note: Files that are on the diskettes being initialized in this step will be deleted, so make sure that these files are not needed.

       (See index entry: *INIT procedure.*)

    b. Use the diskettes initialized in part *a* of this step to back up your system.

       Enter: BACKUP SYSTEM,999

       (See index entries: *Backup configured SCP* and *BACKUP procedure.*)

       Note: If you have not initialized enough diskettes, return to part *a* of this step and initialize more diskettes (the diskettes already used in part *b* of this step must be deleted). (See index entry: *INIT procedure.*)

13. An individual backup copy of each program product may be made.

    (See index entry: *backup copy of a program product.*)

```
ASSMPL DISK FILE TO PRINTER (80/80 LIST PROGRAM)

ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT      VER XX MOD XX   XX-XX-XX  PAGE   4

                                63 * END OF FILE ON SYSIN                                          00420000
    0844 C2 02 0BC4             64 EOF    LA    EOFMSG,LOG                                         00430000
                                65 *        $LOG                    EOF MESSAGE                    00440000
    084D C0 87 0866             68        B     EOJ                     INVALID REPLY, TRY AGAIN     00450000

                                70 * ERROR ON DISK READ                                            00470000
    0851 C2 02 0BC9             71 SYSER  LA    SERMSG,LOG                                         00480000
                                72 *        $LOG                    DISK READ ERROR MESSAGE        00490000
    085A F2 87 09               75        J     EOJ                     GO TO EOJ                    00500000

                                77 * ERROR ON PRINTER                                              00510000
    085D C2 02 0BCE             78 PRNERR LA    PERMSG,LOG                                         00520000
                                79 *        $LOG                    PRINTER ERROR MESSAGE          00530000

                                83 * END OF JOB ROUTINE                                            00550000
                           0866 84 EOJ    EQU   *                                                 00560000
                                85 *        $CLOS DTF-DSKDTF        CLOSE ALL FILES                00570000
                                89 *        $EOJ                    END JOB                        00580000

                                94 * CONSTANTS AND DATA AREAS                                      00600000

                                96 * DISK FILE TABLES ETC.                                         00620000
                                97 *SKDTF   $DTFD ACCESS-CG,RECL-80,NAME-INPUT,BLKL-512,IOAREA-INBUF,  *00630000
                                98 *              CHAIN-PRTDTF,RCAD-INRCRD                         00640000

                               122 * BUFFER AND WORK AREAS FOR DISK INPUT INTERFACE                 00660000
                           08A5 123 INBUF  EQU   *                                                 00670000
    08A5                   08BA 124 IOB    DS    CL22                                              00680000
    08B8                   0ABA 125 INAREA DS    2CL256                                            00690000
                           0ABB 126 INRCRD EQU   *                                                 00700000
    0ABB                   0B0A 127 DSKREC DS    CL80                                              00710000

                               129 * PRINT FILE TABLES ETC.                                        00730000
                               130 *RTDTF   $DTFP RCAD-INRCRD,IOAREA-OUTPUT,RECL-80                00740000

                               148 * BUFFER AND WORK AREAS FOR PRINTER INTERFACE                   00760000
                           0B32 149 OUTPUT EQU   *                                                 00770000
    0B32                   0BC3 150 IOAREA DS    CL146                                             00780000

                               152 * SYSTEM LOG TABLES                                             00800000

                               154 *OFMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-EOFMGC            X00820000
```

```
                                161 *ERMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-SERMGC                    X00840000

                                168 *ERMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-PERMGC                    X00860000

                          0BD3  175 EOFMGC EQU   *                                                          00880000
    0BD3 C5D6C640D6D540E2 0BE1  176         DC    CL15'EOF ON SYSIN   '                                    00890000

                          0BE2  178 SERMGC EQU   *                                                          00910000
    0BE2 E2E8E2C9D540C5D9 0BF0  179         DC    CL15'SYSIN ERROR    '                                    00920000

                          0BF1  181 PERMGC EQU   *                                                          00940000
    0BF1 D7D9C9D5E3C5D940 0BFF  182         DC    CL15'PRINTER ERROR  '                                    00950000

                                184 * OFFSETS FOR ALL DTFS DEFINED IN THIS PROGRAM                         00970000

                                186 *         $DTFD DISK-Y,PRT-Y,FIELD-Y                                   00990000

                                492 * REGISTER LABELS                                                      01010000
                          0002  493 $DTF    EQU   2                                                        01020000
                          0002  494 SYS     EQU   2                    SYSIN PARAMETER LIST POINTER        01030000
                          0002  495 LOG     EQU   2                    SYSLOG PARAMETER LIST POINTER       01040000

                          0800  497         END   ASSMPL                                                   01060000
```

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY--   0

TOTAL SEQUENCE ERRORS IN THIS ASSEMBLY--   0

```
*****************************************************************************
*                                                                           *
*          THE ASSEMBLER SAMPLE PROGRAM IS EXECUTING PROPERLY.              *
*        THIS IS THE PRINTED OUTPUT FROM THE LOAD MODULE OF ASSMPL           *
*                                                                           *
*****************************************************************************
```

FCU Installation Verification

Two sets of sample data files and conversion specifications are provided with the IBM System/32 FCU program product. After the FCU is installed, either set of data files and conversion specifications can be loaded from the PID program product distribution diskette (FCUFCU) and executed by entering either of the following command statements:

```
FCUSAMPL DP
FCUSAMPL WP
```

Entering FCUSAMPL DP does the following operations:

- Loads a sample sequential file and specification source member from the diskette to disk.
- Executes the FCU specification phase to create a specification load member.
- Executes the FCU conversion phase to create an indexed sequential output file.
- Automatically deletes the FCUSAMPL DP procedure, the sample data files,the specification statements, and the load module.

The following is an example of the printed output of a properly installed FCU sample program.

```
FCU SPECIFICATION LISTING FOR MEMBER #FCUDP                          DATE XX/XX/XX

0001       FIS                                                              00020000
0002       FOI  40 6   1                                                   00030000
0003       CP  1   1   6U0          1   1   6U0     ACCOUNT NUMBER        00040000
0004       CP  2   7  10P2          6  33  39U2     CURRENT BALANCE       00050000
0005       CP  3  11  14P2          5  26  32U2     NEW CHARGES           00060000
0006       CP  4  15  18P2          3  12  18U2     PAST DUE AMOUNT       00070000
0007       CP  5  19  22P2          4  19  25U2     PAYMENTS              00080000
0008       CP  6  23  25P0          2   7  11U0     CREDIT LIMIT          00090000
0009       CC                       7  40  40A    1  'DELETE FIELD'      00100000


     0305  SPECIFICATION LOAD MODULE CREATED
```

RECORD KEY 113520
1135200090000121760000000000126140024790

RECORD KEY 118860
1188600080000617350000200004172100834 56

RECORD KEY 953210
9532100050000117400011740000215000021 50

RECORD KEY 233470
2334700090000637850000400001753000413 15

RECORD KEY 286220
2862200050000671410067141005194000519 40

RECORD KEY 825130
8251300300003196670157950031179304735 10

RECORD KEY 312750
3127500090000777600050000005399700817 57

RECORD KEY 324570
3245700040000532000032610002914000497 30

RECORD KEY 298710
2987100090000421360042136003749100374 91

RECORD KEY 437150
4371500080000731910000400000000000331 91

RECORD KEY 439370
4393700080000093100005620000413000078 20

RECORD KEY 451370
4513700050000197170019717002233700223 37

RECORD KEY 469180
4691800100000682350000300006319401014 29

RECORD KEY 583130
5831301000003374150311471005739100833 35

RECORD KEY 791190
7911900080000217190021719001174500117 45

RECORD KEY 913700
9137000080000549730000400007417000891 43

RECORD KEY 987160
9871600080000015420001542008858500885 85

RECORD KEY 307550
3075500060000078160007816006350000635 00

Entering FCUSAMPL WP does the following operations:

- Loads sample sequential and indexed data files and a specification source member from the diskette to the disk.
- Executes the FCU specification phase to create a specification load module.
- Executes the FCU conversion phase to create a tabular document in a document library.
- Automatically deletes the FCUSAMPL WP procedure, the sample data files, the specification statements, and the load module.

Note: This sample requires that System/32 SCP Feature Number 6002 (word processing support) and a 96-character print belt be installed.

The following is an example of the printed output of a properly installed FCU sample program.

```
FCU SPECIFICATION LISTING FOR MEMBER #FCUWP                    DATE XX/XX/XX

0001      FIS                                                          00020000
0002      FSI                                                          00030000
0003      FOL                FCU     WPSAMPLE                          00040000
0004      Q   100GTF110                                               00050000
0005      CK  1   1   6U0                                 KEY FIELD    00060000
0006      CS 10   1   6A          1       A             CHARGE #     00070000
0007      CS 20  33  33A  TITLES  2       A             TITLE        00080000
0008      CS 30  22  31A          3       A P           FIRST NAME   00090000
0009      CS 40  32  32A          4       A U           MIDDLE INITIAL 00100000
0010      CS 50   7  21A  NAMES   5       A P           LAST NAME    00110000
0011      CS 60  34  53A  NSEW    6       A P           STREET       00120000
0012      CS 70  54  73A          7       A P           CITY         00130000
0013      CS 80  74  75A  STATES  8       A             STATE NAME   00140000
0014      CS 90  76  80A          9       A             ZIP CODE     00150000
0015      CC                     10       A     1$      DOLLAR SIGN  00160000
0016      CP100   7  10P2S       11       D2            CURRENT BALANCE 00170000
0017      CC                     12       A     1$      DOLLAR SIGN  00180000
0018      CP110  23  25P0        13       D0            CREDIT LIMIT 00190000
0019      ATITLES 7 71013                                            00200000
0020      .1  Mr.                                                    00210000
0021      .2  Ms.                                                    00220000
0022      .3  Mrs.                                                   00230000
0023      .4  Miss                                                   00240000
0024      .5  Dr.                                                    00250000
0025      ANAMES  7101517*                                           00260000
0026      .MC*     Mc*                                               00270000
0027      .O'*     O'*                                               00280000
0028      ANSEW   7101316*                                           00290000
0029      .NE *  NE *                                                00300000
0030      .NW *  NW *                                                00310000
0031      .SE *  SE *                                                00320000
0032      .SW *  SW *                                                00330000
0033      ASTATES 7 81335                                            00340000
0034      .OH    Ohio                                                00350000
0035      .FL    Florida                                             00360000
0036      .AL    Alabama                                             00370000
0037      .GA    Georgia                                             00380000
0038      .LA    Louisiana                                           00390000
0039      .SC    South Carolina                                      00400000
0040      .MS    Mississippi                                         00410000
0041      .WV    West Virginia                                       00420000
0042      .MD    Maryland                                            00430000
0043      .IL    Illinois                                            00440000
0044      .DC    D. C.                                               00450000
0045      .KY    Kentucky                                            00460000


0305  SPECIFICATION LOAD MODULE CREATED
```

FCU CONVERSION PHASE PROCESSING MEMBER #FCUWP DATE XX/XX/XX

RECORD NO 000001
118860 Barbara McGuire 470 Live Oak Place Albany Georgia
$ 800.

RECORD NO 000002
286220 Mr. Joseph A Abruzzo 3500 Gault Ocean Dr New Orleans Louisiana
$ 500.

RECORD NO 000003
825130 A-1 Used Cars 200 SE 124 St. Maywood Illinois
$ 3,000.

RECORD NO 000004
324570 Mr. Robert Q Dobbs Buttonwood Drive Rome Georgia
$ 400.

RECORD NO 000005
469180 Miss Margaret E Monroe 9 Pine Tree Lane Sunny South Alabama
$ 1,000.

RECORD NO 000006
913700 Ms. Janice L Comstock 2637 Marion Dr Ellensburg D. C.
$ 800.

RECORD NO 000007
987160 Mr. Charles N McCall 669 W Campus Circle Williston South Carolina
$ 800.

RECORD NO 000008
307550 Horace M De Angelo 8150 Cypress Road Everglades Florida
$ 600.

FIELD 100 SUM = 10012.93 MAX = 4735.10 MIN =

This page intentionally left blank

APAR Parameters

vol-id	Volume identification of the diskette to contain the two files APARFILE and FIXDFILE.
object program name	The name of the object program causing the program check interrupt.
source program name	The name of the source program from which the object program causing the program check interrupt was created.

BUILD PROCEDURE

The BUILD procedure helps you correct data on the disk if an error occurs during a disk read or write operation. The BUILD procedure evokes the $BUILD utility program to display and print unreadable data so you can find and correct it. See index entry: *$BUILD utility program,* for a description of how to display and correct data after a disk read or write error occurs.

BUILD Command Statement Format

BUILD

BUILD Parameters

None

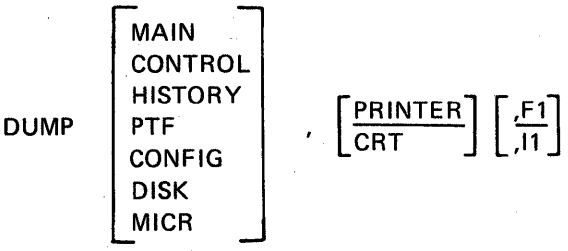
DUMP PROCEDURE

The DUMP procedure prints or displays information saved on the CE cylinder and other protected sectors on the disk. This information, consisting of the contents of main and control storage and the last 20 sectors recorded in the history file, may have been saved because of a program check interrupt or may have been saved because the RESET and then the CE START keys on the CE console were pressed.

DUMP also prints or displays the PTF (program temporary fix) log module and system configuration record. If DISK is specified, selected sectors from the disk (if F1) or a diskette (if I1) are displayed or printed. If MAIN, CONTROL, HISTORY, PTF, CONFIG, or MICR are specified with I1, the specified items are printed or displayed from a diskette file created by the APAR command. (See index entry: *APAR procedure.*) The sectors you select to print or display must be entered as hexadecimal numbers.

The DUMP procedure evokes the $FEDMP utility program.

DUMP Command Statement Format



DUMP Parameters

MAIN — The system status, system communication area (SCA), program level communication area (PLCA), DTFs (define the files) and IOBs (input/output blocks) are dumped; a prompt for main storage address limits (a starting storage address and an ending storage address) follows. After the selected area of storage is dumped, a new limits prompt is issued. You have the END option (terminate the DUMP) after each prompt for main storage limits. MAIN is the default.

CONTROL — The control storage direct area is dumped; a prompt for the control store address limits follows. You can respond with the limits or END.

HISTORY — Dump the saved HISTORY file.

PTF — Dump the PTF log module.

CONFIG — Dump the system configuration record.

DISK — Selected sectors of F1 or I1 can be dumped. Prompts are issued for the starting sector number and number of sectors to be dumped (must be entered in hexadecimal). At the completion of that dump, prompts are issued for the next group of sectors. You can respond with the limits or END.

MICR — Dump the magnetic character reader controller storage area.

PRINTER — Output is on the printer. PRINTER is the default.

CRT — Output is on the display screen, 240 characters at a time. The keyboard function keys can be used to display different portions of the dump.

F1 — The disk contains the information requested by the MAIN, CONTROL, HISTORY, PTF, CONFIG, or DISK parameter. F1 is the default value.

I1 — The diskette contains the information requested by the MAIN, CONTROL, HISTORY, PTF, CONFIG, or DISK parameter.

Example:

A user required the SEU program product, data recorder support, word processing support, and inquiry/offline support on a 16K system.

Directory Sectors	Library Blocks	Library Function
33	239	Minimum System/32 SCP
4	38	SEU
	11	Inquiry/offline support on 16K
1	16	CARDIO (data recorder)
6	60	WPFILE (word processing)
	20	MSGMBR (required by both word processing and data recorder but needs to be added only once)

The total number of directory sectors is 44; the total number of library blocks is 384. These totals are the minimum numbers of directory sectors and library blocks for the requested system.

DELETING FROM THE LIBRARY

Before deleting members from the library, determine how much space is presently available for new members, or how much disk space is available for additional data files.

Determining Space Available in the Library

To determine how much space is available in the library, use the LISTLIBR procedure or the copy function of the $MAINT utility to print the system information from the directory area (see index entries: *$MAINT utility program* and *LISTLIBR procedure*). The system information listed will specify the number of additional entries the directory can contain (AVAILABLE DIRECTORY ENTRIES) and how many sectors are available in the library for additional members (AVAILABLE MEMBER SECTORS).

Determining Space Available on the Disk

To determine how much space exists on the disk for additional data files, use the CATALOG procedure or the $LABEL utility (see index entries: *$LABEL utility program* and *CATALOG procedure*) to display the disk VTOC. Available disk space is specified in every disk VTOC display.

Note: You can also use CATALOG or $LABEL to display all disk VTOC entries to determine which files can be deleted (see index entries: *$DELET utility program* and *DELETE procedure*). Use the COMPRESS procedure or $FREE utility (see index entries: *$FREE utility program* and *COMPRESS procedure*) to collect unused disk space in one area.

To determine how much space will be available for user programs and data files, take the total library requirements of your planned system and subtract this number from the number of disk blocks on your system. (see index entry: *library requirements*)

Note: Convert the sectors to blocks (1 block equals 10 sectors). If there is a remainder, round off that remainder to the next whole number.

Disk blocks available on the IBM System/32 are:

1248 blocks on a 3.2 megabyte disk

1968 blocks on a 5.0 megabyte disk

3576 blocks on a 9.1 megabyte disk

5376 blocks on a 13.7 megabyte disk

Example: The library requirements of the minimum IBM System/32 system control programming are 33 directory sectors and 239 library blocks. This totals 243 blocks (33 sectors converted to blocks rounds to 4 blocks). A 3.2 megabyte disk system leaves 1005 blocks available for user programs and data files.

1248 (blocks on a 3.2 megabyte disk)
-243 (total blocks library requirements)
1005 (total blocks available to the user)

Selecting Members to Delete

The following members can be deleted from the library without affecting other members or SCP functions:

Name	Member Type	Description
##MSG1	0 (load)	Level 1 error messages
##MSG4	0 (load)	Level 2 error messages
Selected procedure (see note)	P (procedure)	Procedures

Note: When you delete a library member, be sure not to delete a procedure within a nested procedure(s) or a procedure called by all procedures. For example, #ERR is a nested procedure available to all procedures for error detection. If ##MSG1 and/or ##MSG4 are deleted, there will be no message text when an error occurs.

In addition to deleting the preceding members you can delete inquiry/offline multivolume support and any program product installed on the system without affecting other system functions.

Deleting (or not including) inquiry/offline support (INCLUDE INQUIRY/OFFLINE? = NO on the RELOAD display—see index entry: *RELOAD display*) saves 11 blocks of library space on a 16K system, 14 blocks on a 24K system, and 17 blocks on a 32K system. Use the LISTLIBR procedure or the copy function of $MAINT to list library directory entries to determine space gained by deleting procedure members, ##MSG1, and ##MSG4. See index entry: *library requirements* to see how much space is gained by deleting a program product.

Note: After deleting members, use the CONDENSE procedure to collect all available space into one area at the end of the library.

Deleting Members

- ##MSG1, ##MSG4, and procedure members are deleted by using the delete function of $MAINT. See index entry: *$MAINT utility program.*

- Inquiry/offline support is deleted by specifying NO to the INQUIRY/OFFLINE option of the RELOAD display. The RELOAD display is described in following paragraphs.

- Program products are deleted by entering a nameDROP command statement for each function to be deleted (DFUDROP, SEUDROP, SORTDROP, RPGDROP, ASMDROP, FORTDROP, and/or FCUDROP). The procedures evoked by these command statements are deleted from the system when the related program product functions are deleted.

After you have deleted members, you can change space allocated to the library by using the RELOAD display, described in the following paragraphs.

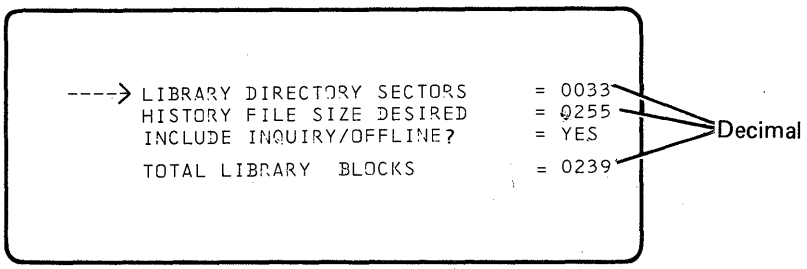
Notes:
1. Do not delete any procedure that is used by a procedure that you are not deleting.
2. To gather the disk space created by deleting members from the library into one usable area, you can use the CONDENSE procedure. See index entry: *CONDENSE procedure.*

RELOAD DISPLAY

The RELOAD procedure (described under index entry: *RELOAD procedure*) is used to perform an IPL from diskettes onto which the library was copied by the BACKUP procedure (described under index entry: *BACKUP procedure*). RELOAD creates a new library on the disk, but does not disturb data files on the disk.

The RELOAD display appears when you insert the first backup diskette (for a particular copy of the library) and enter the RELOAD command statement (described under index entry: *RELOAD command statement*) or when you press the LOAD key with the IPL switch on the CE control panel set to DISKETTE.

The RELOAD display shows the number of sectors allocated for the library directory, indicates whether or not inquiry or offline multivolume files are supported, and shows the total number of blocks allocated for the library (system file #LIBRARY). A sample display follows:



If Values in the RELOAD Display are Correct

If the values shown in the RELOAD display are not correct, see the following page to change the values; otherwise, press the ENTER key (*not* the ENTER+ or ENTER− key). The library is read from the diskette to the disk and the following display appears:

```
INSERT DISKETTE WITH FILE LABEL-#LIBRARY
       DATE-XX/XX/XX, SEQUENCE NUMBER-02
-----> PRESS ENTER KEY AFTER INSERTING
 WARNING-LIBRARY MAY BECOME UNUSABLE
         IF CORRECT VOLUME NOT INSERTED
```

The INSERT DISKETTE display always appears after a diskette is read to the disk. When the display appears, remove the diskette and insert the next diskette as indicated. When all the diskettes are read, the following display appears:

```
RELOAD COMPLETE - REMOVE LAST
DISKETTE AND IPL FROM DISK
```

Remove the diskette, set the IPL and IMPL switches on the CE control panel to DISK, and press the LOAD key. The following display appears:

```
**** INITIAL PROGRAM LOAD COMPLETE ****
             DATE  XXXXXX
             LINES   33
ENTER COMMAND

                                  <-READY
```

Enter a DATE command statement (see index entry: *DATE procedure*) or a SET command statement (see index entry: *SET procedure*) if the date or number of lines printed per page is to be changed.

If Values in the RELOAD Display are to be Changed

To change the values displayed, do the following:

- When the arrow is pointing to the first line (LIBRARY DIRECTORY SECTORS):

  1. If this line is correct, press the REC ADV key. The arrow and cursor move to the second line.

  2. If you want to change the first line, enter the change over the existing data (you may omit leading zeros) and then press the ENTER+ key. The arrow and cursor move to the second line.

     Note: The formula for computing the number of entries the directory can hold is number of directory sectors times 11 minus 23. A directory entry is required for each member in the library.

  3. If all lines of the display are now correct, press the ENTER key (*not* the ENTER+ or ENTER− key). Data is read from the diskette onto the disk, and the INSERT DISKETTE display appears.

- When the arrow is pointing to the second line (HISTORY FILE SIZE DESIRED):

  1. If this line is correct, press the REC ADV key. The arrow and cursor move to the third line.

  2. If you want to change the second line, enter the change over the existing data (you may omit leading zeros) and then press the ENTER+ key. The arrow and cursor move to the third line.

  3. If all lines of the display are now correct, press the ENTER key (not the ENTER+ or ENTER- key). Data is read from the diskette onto the disk, and the INSERT DISKETTE display appears.

  Note: HISTORY file size must be set within the range of 39-255 sectors.

- When the arrow is pointing to the third line (INCLUDE INQUIRY/OFFLINE?):

  1. If this line is correct, press the REC ADV key. The arrow and cursor move to the fourth line.

  2. If you want to change the third line, enter the change (YES or NO) over the existing data and then press the ENTER+ key. The arrow and cursor move to the fourth line.

     Note: The inquiry/offline option requires a disk area in which to roll out an interrupted program or to process an offline multivolume file segment. The size of this area is 11 blocks on a 16K system, 14 blocks on a 24K system, and 17 blocks on a 32K system. This area must be represented in the total number of library blocks if inquiry/offline support is included.

  3. If all lines of the display are now correct, press the ENTER key (*not* the ENTER+ or ENTER− key). Data is read from the diskette onto the disk, and the INSERT DISKETTE display appears.

- If the arrow is pointing to the fourth line (TOTAL LIBRARY BLOCKS):

  1. If this line is correct, press the REC ADV key. The arrow and cursor move to the first line.

  2. If you want to change the fourth line, enter the change over the existing data (you may omit any leading zeros) and then press the ENTER+ key.

     Note: The number of blocks assigned to the library must be sufficient to contain the library directory and the disk area (rollout area) required by inquiry/offline support, if it is included, as well as all library members. You should also allow some space in the library for new members because of the inconvenience of expanding the library once the remaining disk space is allocated to data files.

  3. If all lines of the display are now correct, press the ENTER key (*not* the ENTER+ or ENTER− key). Data is read from the diskette onto the disk, and the INSERT DISKETTE display appears.

The following instructions are intended to be used as a guide for installing a version update on an IBM System/32. These instructions are a summary of the detailed instructions that are presented in Part 5, *System Configuration, Installation, and Modification.* Index entries follow each step for the detailed description.

Note: Your IBM service representative can tell you if there are any PTFs applicable to your version of the SCP, or to your version of any program product. If there are PTFs, make arrangements with your IBM representative to have the PTF diskette available when you do your version update. The PTF diskette contains all applicable PTFs.

To install a version update on the IBM System/32 execute the following steps:

1. Print the system information from the system library to determine the total number of library blocks, the directory size, and if you are using inquiry/offline. Save the printed listing for step 6.

   Enter: LISTLIBR DIR,SYSTEM

   (See index entries: *printing from the library* and *LISTLIBR procedure.*)

2. Delete IBM program products that are installed on the system so that the system contains only user programs. User programs are saved in step 3.

   Enter the following appropriate command for the program product you have installed:

   ✓SEUDROP (if SEU is installed)
   ✓DFUDROP (if DFU is installed)
   ✓SORTDROP (if SORT is installed)
   ✓RPGDROP (if RPG is installed)
   FCUDROP (if FCU is installed)
   FORTDROP (if FORTRAN IV is installed)
   ASMDROP (if basic assembler is installed)

   (See index entry: *deleting members.*)

3. Save your user programs on a diskette file (filename used here is USERLIBR). The USERLIBR file is restored to disk in step 9.

   a. Initialize enough diskettes to contain your user programs (vol-id used here is USER).

      Note: FORMAT2 may require fewer diskettes.

      Enter: INIT USER,,FORMAT2

      Note: Files that are on the diskettes being initialized in this step are deleted, so make sure these files are not needed. (See index entry: *INIT procedure.*)

   b. Use the diskettes initialized in part *a* of this step to save your user programs.

      Enter: FROMLIBR ALL,LIBRARY,USERLIBR,,999,USER

      (See index entry: *FROMLIBR procedure.*)

      Note: If you have not initialized enough diskettes, return to part *a* of this step and initialize more diskettes (the diskettes already used in part *b* of this step must be deleted.) (See index entry: *INIT procedure.*)

4. List the disk VTOC and save this list to compare with the list that will be printed in step 10 to verify that no data files were lost.

   Enter: CATALOG

   (See index entry: *CATALOG procedure.*)

5. Save your data files on a diskette file.

   a. Initialize enough diskettes to contain your data files (vol-id used here is DFSAVE).

      Note: FORMAT2 may require fewer diskettes.

      Enter: INIT DFSAVE,,FORMAT2

      Note: Files that are on the diskettes being initialized in this step are deleted, so make sure that these files are not needed.

      (See index entry: *INIT procedure.*)

   b. Use the diskettes initialized in part *a* of this step to save your data files.

      Enter: SAVE ALL,,,DFSAVE

      (See index entry: *SAVE procedure.*)

      Note: If you have not initialized enough diskettes, return to part *a* of this step and initialize more diskettes (the diskettes already used in part *b* of this step must be deleted.) See index entry: *INIT procedure.*

Load SCP support for data recorder attachment (the nested procedure name is CNFICDIO):

```
// LOAD $MAINT
// FILE NAME-CARDIO,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-CARDIO
// END
```

Load SCP support for word processing (the nested procedure name is CNFIGULF):

```
// LOAD $MAINT
// FILE NAME-WPFILE,UNIT-I1
// FILE NAME-WCFILE,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-WPFILE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-WCFILE
// END
```

Set country code options for word processing (the nested procedure name is CNFIGULF):

```
// LOAD $WPSET
// RUN
// CC nn
// END
```

Load OCL support for the 1255 Magnetic Character Reader attachment (the nested procedure name is CNFIMICR):

```
// LOAD $MAINT
// FILE NAME-MICR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-MICR
// END
```

Load SCP support for FORTRAN IV (the nested procedure name is CNFIFORT):

```
// LOAD $MAINT
// FILE NAME-FORTRAN,UNIT-I1
// FILE NAME-OLE,UNIT-I1
// FILE NAME-COMNSUBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-FORTRAN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-COMNSUBR
// END
```

Load SCP support for basic assembler (the nested procedure name is CNFIAMPR):

```
// LOAD $MAINT
// FILE NAME-OLE,UNIT-I1
// FILE NAME-AMMACO,UNIT-I1
// FILE NAME-RPGSUBR,UNIT-I1
// FILE NAME-COMNSUBR,UNIT-I1
// FILE NAME-AMBSCA,UNIT-I1
// FILE NAME-BSCALOAD,UNIT-I1
// FILE NAME-BSCASUBR,UNIT-I1
// FILE NAME-AMSCNT,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMMACO
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-RPGSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-COMNSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMBSCA
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCALOAD
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCASUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMSCNT
// END
```

Load SCP support for the overlay linkage editor (the nested procedure name is CNFIOLED):

```
// LOAD $MAINT
// FILE NAME-OLE,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// END
```

Load SCP support for queued job stream (the nested procedure name is CNFIQJOB):

```
// LOAD $MAINT
// FILE NAME-QJOB,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-QJOB
// END
```

Load SCP support for optional messages (the nested procedure name is CNFIMSGS):

```
// LOAD $MAINT
// FILE NAME-MSGMBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-MSGMBR
// END
```

Apply PTFs to SCP and optional programs (the nested procedure name is CNFIPTFS), see index entry: *APPLYPTF procedure.*

Remove the CNFIGSCP procedures from the library:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-P,NAME-CNFI.ALL,RETAIN-S
// END
```

COMPRESS

```
// LOAD $PACK
// RUN
```

CONDENSE

```
// LOAD $MAINT
// RUN
// COMPRESS
// END
```

CONVERT

```
// LOAD $CNVRT
// RUN
```

COPYI1

```
// LOAD $DUPRD
// FILE NAME-COPYI1[,DATE-date],UNIT-I1
// RUN
// COPYI1 NAME-{filename / ALL},PACK-vol-id [,DELETE-{YES / NO}]
    [,PRESERVE-{YES / NO}] [,COPIES-{number of copies / 1}]
// END
```

CREATE

```
// LOAD $MGBLD
// RUN
// MGBLD SOURCE-sourcename,REPLACE-{YES / NO}
// END
```

DATE

```
// DATE-date
```

DCPRINT

```
// LOAD $DCSUP
// RUN
[// COPYFILE NAME-filename,OUTPUT-PRINT]
[// GO]
[// END]
```

DELETE

```
// LOAD $DELET
// RUN
// SCRATCH LABEL-filename [,DATE-date] ,UNIT- {F1 / I1}
   and/or
// REMOVE LABEL-filename,DATA- {YES / NO} [,DATE-date] ,UNIT- {F1 / I1}
// END
```

DISPLAY

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-filename [,DATE-date] ,UNIT-F1
// RUN
// COPYFILE OUTPTX-PRINT
[// SELECT RECORD,FROM-number-1 [,TO-number-2]]
// END
```

DUMP

```
// LOAD $FEDMP
// RUN
// DUMP [ LIST- { MAIN / CONTROL / HISTORY / PTF / CONFIG / DISK / MICR } ] [,OUTPUT- {PRINTER / CRT}] [,INPUT- {F1 / I1}]
// END
```

FROMLIBR

```
// LOAD $MAINT
// FILE NAME-  [filename-1 / library-name-1 / filename-2 / name-1]  ,RETAIN-  [P / S / T] (If F1 is specified.) / [retention-days / 1]  [,blocks / ,8]
                                                                                                                  or
                                                                                                                ,PACK-vol-id
   UNIT-  {F1 / I1}

   or, if ADD is specified,

// FILE NAME-  [filename-1 / library-name-1 / filename-2 / name-1]  [,PACK-vol-id] (If UNIT-I1)  ,UNIT-  {F1 / I1}
// RUN
// COPY FROM-F1,LIBRARY-  {S / P / O / R / ALL}  ,NAME-  {library-name-1 / name-1.ALL}
   FILE-  [filename-1 / library-name-1 / filename-2 / name-1]  ,TO-DISK,OMIT-SYSTEM [,ADD-YES]
// END
```

HISTORY

```
// LOAD $HIST
// RUN
   [// DISPLAY [ALL]]      } If NOLIST is not specified
// END

// LOAD $HINT              } If RESET is specified
// END
```

INIT

```
// LOAD $INIT
// RUN
// UIN OPTION-  {FORMAT / FORMAT2 / DELETE / RENAME}
[ // VOL PACK-  {vol-id / system date}  ,ID-  {owner-id / OWNERID} ]
// END
```

INSTALL

(The following are some of the OCL statements for INSTALL.)

Print System Directory:

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,TO-PRINT,LIBRARY-SYSTEM,NAME-DIR
// END
```

Delete INSTALL procedures from tailored system:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-P,NAME-INST.ALL,RETAIN-S
// END
```

JOBSTR

For card input

```
// LOAD $QJOB
// RUN
```

For diskette input:

```
// LOAD $BICR
// FILE NAME-COPYIN,LABEL-filename,UNIT-I1
// FILE NAME-COPYO,LABEL-filename,UNIT-F1,RECORDS-number
// RUN
// TRANSFER
// END

// LOAD $MAINT
// FILE NAME-filename,UNIT-F1
// RUN
// COPY FROM-DISK,FILE-filename,TO-F1
// END

// LOAD $DELET
// RUN
// REMOVE LABEL-filename,UNIT-F1
// END
```

For executing a procedure if the procedure name is specified:

```
// INCLUDE procedurename
```

For deleting the procedurename if the NOSAVE parameter is specified:

```
// LOAD $MAINT
// RUN
// DELETE NAME-procedurename,LIBRARY-P
// END
```

Appendix C. Diskette Formats and Diskette Data Files

Diskette data files for IBM System/32 reside on diskettes that are initialized in one of two physical formats.

DISKETTE FORMATS

IBM System/32 processes diskettes that are initialized in either the 128-bytes per sector basic data exchange format or the 512-bytes per sector extended format. The INIT procedure and $INIT system utility can initialize diskettes in either format. (See index entries: *INIT procedure* and *$INIT utility program.*)

The sectors in track 0 (index track) of both formats are 128-bytes. Data sectors on the 128-bytes per sector format diskette are also 128-bytes. Data sectors on the 512-bytes per sector format diskette are 512-bytes.

DISKETTE DATA FILES

IBM System/32 creates and processes two kinds of diskette data files: basic data exchange files and IBM System/32 system files.

Basic Data Exchange Files

Basic data exchange files can reside only on diskettes initialized in the 128-bytes per sector format on tracks 1-73. These files can be used for exchanging diskettes between systems or devices. See *The IBM Diskette General Information Manual,* GA21-9182 for a description of the data set label fields.

Basic data exchange files are created by the TRANSFER procedure and $BICR system utility. The copy of a diskette file created by the COPYI1 procedure or $DUPRD utility is a basic data exchange file if the original diskette file is a basic data exchange file. (For a description of the procedures and utilities just mentioned, see index entries: *COPYI1 procedure, TRANSFER procedure, $BICR utility program,* and *$DUPRD utility program.*)

System Files

System files can reside on diskettes initialized in either the 128-bytes per sector format or the 512-bytes per sector format on tracks 1-74. These files can be used on the IBM System/32 only. See *The IBM Diskette General Information Manual,* GA21-9182 for a description of data set label fields.

System files are created by the BACKUP, FROMLIBR, ORGANIZE, and SAVE procedures, and by the $BACK, $COPY, and $MAINT utilities. The copy of a diskette file created by the COPYI1 procedure or $DUPRD utility is a system file if the original diskette file is a system file. (For a description of the procedures and utilities just mentioned, see index entries: *BACKUP procedure, COPYI1 procedure, FROMLIBR procedure, ORGANIZE procedure, SAVE procedure, $BACK utility program, $COPY utility program, $DUPRD utility program,* and *$MAINT utility program*.)

REMOVE

```
// LOAD $MAINT
// RUN
// DELETE NAME- { library-name } ,LIBRARY- { S }
                { name.ALL   }             { P }
                { ALL        }             { O }
                                           { R }
                                           { ALL }
// END
```

RENAME

```
// LOAD $RENAM
// RUN
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE- { mmddyy }]
                                                      { ddmmyy }
                                                      { yymmdd }
// END
```

RESTORE

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL- { filename-1 }  [,DATE-date],UNIT-I1
                           { #SAVE     }
                           { filename-2 }
// FILE NAME-COPYO[,LABEL-filename-2] [{,RECORDS-value-1}] [,UNIT-F1]
                                      {,BLOCKS-value-2 }
// RUN
// COPYALL TO-F1
   or
// COPYFILE OUTPUT-DISK,REORG-NO
// END
```

SAVE

```
// LOAD $COPY
// FILE NAME-COPYIN[,LABEL-filename-2] [,DATE-date] [,UNIT-F1]
// FILE NAME-COPYO [,RETAIN- { retention-days }][,LABEL- { filename-2 }],
                             { 1             }          { filename-1 }
                                                        { #SAVE,     }
   PACK-vol-id,UNIT-I1
// RUN
// COPYALL TO-I1
   or
// COPYFILE OUTPUT-DISK,REORG-NO
   or
// COPYADD
// END
```

SET

```
// LOAD $SETCF
[// IMAGE MEM,source-name]
[// DATE date]
// RUN
// SETCF [LINES-number] [,FORMAT- {MDY|DMY|YMD}] [,IMAGE- {YES|NO}]
// END
```

SETMICR

```
// LOAD $SETCF
// RUN
// SETR CYCLE- {Y|N}
// END
```

SPECIFY

```
// LOAD $SETCF
// RUN
// SETS [ADDR-nn] [,LINE- {C|P|S|T}] [,SWTYP- {AA|MA|MC}] [,ID-nnnnn]
// END
```

STATUS

```
// LOAD $STATS
// RUN
```

SYSLIST

```
// SYSLIST [PRINTER|CRT|OFF]
```

TOLIBR

```
// LOAD $MAINT
// FILE NAME-filename[,DATE-date],UNIT- {F1|I1}
// RUN
// COPY FROM-DISK,FILE-filename,RETAIN- {P|R},TO-F1
// END
```

TRACE

```
// LOAD $SETCF
// RUN
// TRACE [ ALL-Y ] [,FDIOS-Y] [,CSFDIOS-Y] [,PUSH-Y] [,PULL-Y]
         [ WAIT-N ] [,FDIOS-N] [,CSFDIOS-N] [,PUSH-N] [,PULL-N]
         [ WAIT-Y ]
 [,DISABLE-Y] [,ENABLE-Y] [,QUEUE-Y] [,LDCS-Y] [,LOADER-Y]
 [,DISABLE-N] [,ENABLE-N] [,QUEUE-N] [,LDCS-N] [,LOADER-N]
 [,XIENT-Y] [ ,XFER-Y ]
 [,XIENT-N] [ ,XFER-N ]
// END
```

This page intentionally left blank

APAR Parameters

vol-id	Volume identification of the diskette to contain the two files APARFILE and FIXDFILE.
object program name	The name of the object program causing the program check interrupt.
source program name	The name of the source program from which the object program causing the program check interrupt was created.

BUILD PROCEDURE

The BUILD procedure helps you correct data on the disk if an error occurs during a disk read or write operation. The BUILD procedure evokes the $BUILD utility program to display and print unreadable data so you can find and correct it. See index entry: *$BUILD utility program,* for a description of how to display and correct data after a disk read or write error occurs.

BUILD Command Statement Format

BUILD

BUILD Parameters

None

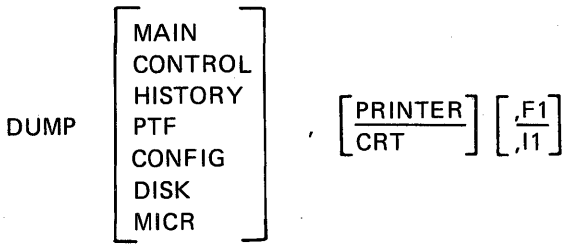
DUMP PROCEDURE

The DUMP procedure prints or displays information saved on the CE cylinder and other protected sectors on the disk. This information, consisting of the contents of main and control storage and the last 20 sectors recorded in the history file, may have been saved because of a program check interrupt or may have been saved because the RESET and then the CE START keys on the CE console were pressed.

DUMP also prints or displays the PTF (program temporary fix) log module and system configuration record. If DISK is specified, selected sectors from the disk (if F1) or a diskette (if I1) are displayed or printed. If MAIN, CONTROL, HISTORY, PTF, CONFIG, or MICR are specified with I1, the specified items are printed or displayed from a diskette file created by the APAR command. (See index entry: *APAR procedure.*) The sectors you select to print or display must be entered as hexadecimal numbers.

The DUMP procedure evokes the $FEDMP utility program.

DUMP Command Statement Format



DUMP Parameters

MAIN — The system status, system communication area (SCA), program level communication area (PLCA), DTFs (define the files) and IOBs (input/output blocks) are dumped; a prompt for main storage address limits (a starting storage address and an ending storage address) follows. After the selected area of storage is dumped, a new limits prompt is issued. You have the END option (terminate the DUMP) after each prompt for main storage limits. MAIN is the default.

CONTROL — The control storage direct area is dumped; a prompt for the control store address limits follows. You can respond with the limits or END.

HISTORY — Dump the saved HISTORY file.

PTF — Dump the PTF log module.

CONFIG — Dump the system configuration record.

DISK — Selected sectors of F1 or I1 can be dumped. Prompts are issued for the starting sector number and number of sectors to be dumped (must be entered in hexadecimal). At the completion of that dump, prompts are issued for the next group of sectors. You can respond with the limits or END.

MICR — Dump the magnetic character reader controller storage area.

PRINTER — Output is on the printer. PRINTER is the default.

CRT — Output is on the display screen, 240 characters at a time. The keyboard function keys can be used to display different portions of the dump.

F1 — The disk contains the information requested by the MAIN, CONTROL, HISTORY, PTF, CONFIG, or DISK parameter. F1 is the default value.

I1 — The diskette contains the information requested by the MAIN, CONTROL, HISTORY, PTF, CONFIG, or DISK parameter.

PATCH PROCEDURE

The PATCH procedure enables IBM service personnel to modify (patch) a disk or diskette sector. The sector to be modified is displayed, 40 characters at a time, on the display screen. Then, the keyboard is used to enter patch data.

CAUTION
PATCH can alter any sector of disk storage with the exception of tracks 0, 1, 2, 4, and 5, but it does not test whether the disk area is the library area, user area, or fixed area. Therefore, an **error during this procedure could cause unpredictable results.**

When the PATCH command statement is entered, a prompt for the sector number is displayed. The sector number must be entered as a hexadecimal number. This sector is then displayed and patch data is entered from the keyboard as the affected portion of the sector is displayed. After all changes are made to a sector, the sector is written back to the disk by pressing the REC ADV key. The next sequential sector is then displayed. Other sectors are displayed by pressing the ENTER key and responding to the prompt. To end the job, enter END in response to the prompt.

Each line of the display screen is as follows:

Line 1 Printable EBCDIC characters
Line 2 and 3 } Hexadecimal representation of the characters in line 1
Line 4 The cursor position and the current sector format and address
Line 6 A warning message to the user

The PATCH procedure evokes the $FEPCH utility program.

PATCH Command Statement Format

PATCH [F1 / I1] [,NOHEX]

PATCH Parameters

F1	A disk sector is to be patched. (F1 is the default.)
I1	A diskette sector is to be patched.
NOHEX	The hexadecimal representations of only unprintable characters are to be displayed. If this parameter is not specified the hexadecimal representations of all characters are displayed.

TRACE PROCEDURE

The TRACE procedure provides the ability to compile a history of, or trace, important SCP events occurring in the system. Whenever a request indicator byte (RIB) or other branch to the supervisor is issued, its value, or function, is checked. If the function is one for which a trace was requested, a 12-byte entry describing the function is placed in a trace table in main storage. The table can contain 21 entries. If the table is filled, new entries replace those recorded first in the table; that is, the table is a wraparound table.

If the contents of main storage are saved on the CE cylinder because a processor check interrupt occurred or because the RESET and then the CE START keys were pressed on the CE control panel, the trace table, being contained in main storage, is available on the disk. It is printed or displayed by the DUMP procedure (see index entry: *DUMP procedure*) if DUMP is used to print or display the saved contents of main storage. If the contents of main storage are printed, the trace table is formatted to clearly identify the table and the kinds of information contained in the entries.

The following system functions can be traced:

- Wait
- Disk IOS
- Control storage disk IOS
- Push
- Pull
- Disable
- Enable
- Queue
- Control storage load
- Main storage load
- Transient load
- XFER instruction

Information provided by the trace includes RIB values or supervisor call (SVC) codes, register contents, and selected disk IOB (input/output block) information.

TRACE evokes the $SETCF utility (see index entry: *$SETCF utility program*).

TRACE Command Statement Format

TRACE [ALL / OFF] [,WAIT] [,FDIOS] [,CSFDIOS] [,PUSH] [,PULL] [,DISABLE]
[,ENABLE] [,QUEUE] [,LDCS] [,LOADER] [,XIENT] [,XFER]

Note: If either ALL or OFF is specified, ALL or OFF must be the first parameter. All other parameters specified are ignored. The remaining parameters can be specified in any order. A maximum of 10 parameters can be specified. The entire SCP trace function is disabled if DEBUG-Y is specified in the ALTERBSC command statement, the ALTERSDL command statement, $SETCF SETB and SETP utility control statement (see index entries: *ALTERBSC procedure* and *BSCA environment* ALTERSDL procedure and SDLC environment).

TRACE Parameters

ALL	All traceable system functions are to be traced. ALL is a default value.
OFF	None of the system functions are to be traced.
WAIT	Each evocation of the wait function is to be traced.
FDIOS	Each evocation of disk IOS (input/output supervisor) is to be traced.
CSFDIOS	Each evocation of control storage disk IOS is to be traced.
PUSH	Each evocation of the push function is to be traced.
PULL	Each evocation of the pull function is to be traced.
DISABLE	Each evocation of the disable interrupt function is to be traced.
ENABLE	Each evocation of the enable interrupt function is to be traced.
QUEUE	Each evocation of the queue function is to be traced.
LDCS	Each evocation of the control storage transient loader is to be traced.
LOADER	Each evocation of the main storage relocating loader is to be traced.
XIENT	Each evocation of the main storage transient loader is to be traced.
XFER	Each execution of the XFER instruction is to be traced.

This appendix shows the OCL and utility control statements contained in each IBM procedure. The substitution expressions that determine which statements are generated for a particular procedure are not shown. This appendix is intended as a reference for programmers who want to know what is executed when a procedure is evoked.

ALTERBSC

```
// LOAD $SETCF
// RUN
// SETB [BRATE-{F|H}] [,CLOCK-{Y|N}] [,DEBUG-{Y|N}] [,ERC-{number|Z}]
        [,SLINE-{Y|N}] [,TEST-{Y|N}] [,TONE-{Y|N}]
// END
```

ALTERSDL

```
// LOAD $SETCF
// RUN
// SETB [BRATE-{F|H}] [,CLOCK-{Y|N}] [,DEBUG-{Y|N}]
        [,SLINE-{Y|N}] [,TEST-{Y|N}] [,TONE-{Y|N}]
// END
```

APAR

```
// LOAD $FEAPR
// FILE NAME-APARFILE,RETAIN-999,PACK-vol-id,UNIT-I1
// FILE NAME-FIXDFILE,RETAIN-999,PACK-vol-id,UNIT-I1
// RUN
[// FROMLIBR object program name,LOAD,APARLOAD,,999,vol-id]
[// FROMLIBR source program name,APARSRCE,,999,vol-id]
```

APCHANGE

If the first parameter, blocks, in the command statement is specified:

```
// LOAD $PACK
// RUN
```

If the second parameter, filename, in the command statement is specified:

```
// LOAD $MAINT
// FILE NAME-filename,UNIT-I1
// RUN
[// DELETE NAME-ALL,LIBRARY-ALL]
// COMPRESS
// COPY FROM-DISK,FILE-filename,TO-F1
// END
```

APPLYPTF

```
// LOAD $MAINT
// FILE NAME-{SC1nn|RG1nn|UT1nn|UT2nn|FO1nn|AS1nn},UNIT-I1
// RUN
```

If the second parameter in the command statement is OLD:

```
// COPY FROM-DISK,TO-F1,FILE-{SC1nn|RG1nn|UT1nn|UT2nn|FO1nn|AS1nn},RETAIN-R,OMIT-NEW
```

If the second parameter is ALL:

```
// COPY FROM-DISK,TO-F1,FILE-{SC1nn|RG1nn|UT1nn|UT2nn|FO1nn|AS1nn},RETAIN-R
```

If the second parameter is PTF log number:

```
// COPY FROM-DISK,TO-F1,FILE-{SC1nn|RG1nn|UT1nn|UT2nn|FO1nn|AS1nn},PTF-ptfid,RETAIN-R
// END
```

BACKUP

```
// LOAD $BACK
// FILE NAME-#LIBRARY,LABEL- {filename   } ,RETAIN- {retention-days}
                             {#LIBRARY  }           {1             }
   PACK-vol-id,UNIT-I1
// RUN
```

BUILD

```
// LOAD $BUILD
// RUN
```

BWSUD

```
// LOAD $BWSUD
// RUN
. . CONFIG SLUNAME-name,HOST-name
. . GO
// END
```

BWSUR

```
// LOAD $BWSUR
// RUN
. . CONFIG SLUNAME-name
. . GO
// END
```

CATALOG

```
// LOAD $LABEL
// RUN
// DISPLAY UNIT- {I1} ,LABEL- {filename}
                 {F1}         {ALL     }
// END
```

Set belt image option:

```
// LOAD $SETCF
// IMAGE MEM, { BELT48   }
              { BELT64   }
              { BELT96   }
              { BELT48HN }
// RUN
// SETCF IMAGE-YES
// END
```

Set number of lines per page option:

```
// LOAD $SETCF
// RUN
// SETCF LINES- {1 to 84}
// END
```

Set date format option:

```
// LOAD $SETCF
// RUN
// SETCF FORMAT- { YMD }
                 { MDY }
                 { DMY }
// END
```

Load SCP support for BSC data communications (the nested procedure name is CNFIBSCA):

```
// LOAD $MAINT
// FILE NAME-BSCALOAD,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCALOAD
// END
```

Load SCP support for MRJE data communications (the nested procedure name is CNFIMRJE):

```
// LOAD $MAINT
// FILE NAME-MRJELOAD,UNIT-I1
// FILE NAME-BSCALOAD,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-MRJELOAD
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCALOAD
// END
```

Load SCP support for batch work station data communications (the nested procedure name is CNFITPSD):

```
// LOAD $MAINT
// FILE NAME-BWSLOAD,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BWSLOAD
// END
```

Set line type option for BSC-switched (the nested procedure name is CNFIOVRD):

```
// LOAD $SETCF
// RUN
// SETR LINE- {C|S} ,SWTYP- {AA|MA|MC}
// END
```

Set line type option for BSC-nonswitched (the nested procedure name is CNFILINE):

```
// LOAD $SETCF
// RUN
// SETR LINE- {P|R|T}
// END
```

Set line type option for SDLC—switched (the nested procedure name is CNFIOVSD):

```
// LOAD $SETCF
// RUN
// SETS LINE- {C|S} ,SWTYP- {AA|MA|MC}
// END
```

Set line type option for SDLC—nonswitched (the nested procedure name is CNFILISD):

```
// LOAD $SETCF
// RUN
// SETS LINE- {P|T}
// END
```

World Trade answer tone option for BSC (the nested procedure name is CNFITPBS):

```
// LOAD $SETCF
// RUN
// SETB ERC-7,SLINE-N,BRATE-F,DEBUG-N,TONE- {Y|N}
// END
```

Modem clocking option for BSC (the nested procedure name is CNFITPBS):

```
// LOAD $SETCF
// RUN
// SETB CLOCK- {Y|N}
// END
```

IBM modem option for BSC (the nested procedure name is CNFITPBS):

```
// LOAD $SETCF
// RUN
// SETB TEST- {Y}
              {N}
// END
```

World Trade answer tone option for SDLC (the nested procedure name is CNFITPSD):

```
// LOAD $SETCF
// RUN
// SETP SLINE-N,BRATE-F,DEBUG-N,TONE- {Y}
                                      {N}
// END
```

Modem clocking option for SDLC (the nested procedure name is CNFITPSD):

```
// LOAD $SETCF
// RUN
// SETP CLOCK- {Y}
               {N}
// END
```

IBM modem option for SDLC (the nested procedure name is CNFITPSD):

```
// LOAD $SETCF
// RUN
// SETP TEST- {Y}
              {N}
// END
```

Load SCP support for RPG (the nested procedure name is CNFIRG1):

```
// LOAD $MAINT
// FILE NAME-RPGSUBR,UNIT-I1
// FILE NAME-COMNSUBR,UNIT-I1
// FILE NAME-RPGLINK,UNIT-I1
// FILE NAME-BSCASUBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCASUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-RPGSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-COMNSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-RPGLINK
// END
```

Load SCP support for data recorder attachment (the nested procedure name is CNFICDIO):

```
// LOAD $MAINT
// FILE NAME-CARDIO,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-CARDIO
// END
```

Load SCP support for word processing (the nested procedure name is CNFIGULF):

```
// LOAD $MAINT
// FILE NAME-WPFILE,UNIT-I1
// FILE NAME-WCFILE,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-WPFILE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-WCFILE
// END
```

Set country code options for word processing (the nested procedure name is CNFIGULF):

```
// LOAD $WPSET
// RUN
// CC nn
// END
```

Load OCL support for the 1255 Magnetic Character Reader attachment (the nested procedure name is CNFIMICR):

```
// LOAD $MAINT
// FILE NAME-MICR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-MICR
// END
```

Load SCP support for FORTRAN IV (the nested procedure name is CNFIFORT):

```
// LOAD $MAINT
// FILE NAME-FORTRAN,UNIT-I1
// FILE NAME-OLE,UNIT-I1
// FILE NAME-COMNSUBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-FORTRAN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-COMNSUBR
// END
```

Load SCP support for basic assembler (the nested procedure name is CNFIAMPR):

```
// LOAD $MAINT
// FILE NAME-OLE,UNIT-I1
// FILE NAME-AMMACO,UNIT-I1
// FILE NAME-RPGSUBR,UNIT-I1
// FILE NAME-COMNSUBR,UNIT-I1
// FILE NAME-AMBSCA,UNIT-I1
// FILE NAME-BSCALOAD,UNIT-I1
// FILE NAME-BSCASUBR,UNIT-I1
// FILE NAME-AMSCNT,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMMACO
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-RPGSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-COMNSUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMBSCA
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCALOAD
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-BSCASUBR
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-AMSCNT
// END
```

Load SCP support for the overlay linkage editor (the nested procedure name is CNFIOLED):

```
// LOAD $MAINT
// FILE NAME-OLE,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-OLE
// END
```

Load SCP support for queued job stream (the nested procedure name is CNFIQJOB):

```
// LOAD $MAINT
// FILE NAME-QJOB,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-QJOB
// END
```

Load SCP support for optional messages (the nested procedure name is CNFIMSGS):

```
// LOAD $MAINT
// FILE NAME-MSGMBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-MSGMBR
// END
```

Apply PTFs to SCP and optional programs (the nested procedure name is CNFIPTFS), see index entry: *APPLYPTF procedure.*

Remove the CNFIGSCP procedures from the library:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-P,NAME-CNFI.ALL,RETAIN-S
// END
```

COMPRESS

```
// LOAD $PACK
// RUN
```

CONDENSE

```
// LOAD $MAINT
// RUN
// COMPRESS
// END
```

CONVERT

```
// LOAD $CNVRT
// RUN
```

COPYI1

```
// LOAD $DUPRD
// FILE NAME-COPYI1[,DATE-date],UNIT-I1
// RUN
// COPYI1 NAME-{filename / ALL} ,PACK-vol-id [,DELETE-{YES / NO}]
   [,PRESERVE-{YES / NO}] [,COPIES-{number of copies / 1}]
// END
```

CREATE

```
// LOAD $MGBLD
// RUN
// MGBLD SOURCE-sourcename,REPLACE-{YES / NO}
// END
```

DATE

```
// DATE-date
```

DCPRINT

```
// LOAD $DCSUP
// RUN
[// COPYFILE NAME-filename,OUTPUT-PRINT]
[// GO]
[// END]
```

DELETE

```
// LOAD $DELET
// RUN
// SCRATCH LABEL-filename [,DATE-date] ,UNIT- {F1 / I1}
   and/or
// REMOVE LABEL-filename,DATA- {YES / NO} [,DATE-date] ,UNIT- {F1 / I1}
// END
```

DISPLAY

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-filename [,DATE-date] ,UNIT-F1
// RUN
// COPYFILE OUTPTX-PRINT
[// SELECT RECORD,FROM-number-1 [,TO-number-2]]
// END
```

DUMP

```
// LOAD $FEDMP
// RUN
// DUMP [ LIST- ( MAIN / CONTROL / HISTORY / PTF / CONFIG / DISK / MICR ) ] [,OUTPUT- {PRINTER / CRT}] [,INPUT- {F1 / I1}]
// END
```

FROMLIBR

```
// LOAD $MAINT
// FILE NAME-  [filename-1 / library-name-1 / filename-2 / name-1]  ,RETAIN-  [P / S / T (If F1 is specified.) / retention-days / 1]  [,blocks / ,8]
                                                                                                                  or
                                                                                                                ,PACK-vol-id
   UNIT-  {F1 / I1}

   or, if ADD is specified,
// FILE NAME-  [filename-1 / library-name-1 / filename-2 / name-1]  [,PACK-vol-id] ,UNIT-  {F1 / I1}
                                                                    If UNIT-I1
// RUN
// COPY FROM-F1,LIBRARY-  {S / P / O / R / ALL}  ,NAME-  {library-name-1 / name-1.ALL}  ,
   FILE-  [filename-1 / library-name-1 / filename-2 / name-1]  ,TO-DISK,OMIT-SYSTEM [,ADD-YES]
// END
```

HISTORY

```
// LOAD $HIST
// RUN
  [// DISPLAY  [ALL / NOLIST / VIEWED]  ,  [RESET / NORESET]  [,PRINT-nnn]]
// END
```

INIT

```
// LOAD $INIT
// RUN
// UIN OPTION-  {FORMAT / FORMAT2 / DELETE / RENAME}
[// VOL PACK-  {vol-id / system date}  ,ID-  {owner-id / OWNERID}]
// END
```

INSTALL

(The following are some of the OCL statements for INSTALL.)

Print System Directory:

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,TO-PRINT,LIBRARY-SYSTEM,NAME-DIR
// END
```

Delete INSTALL procedures from tailored system:

```
// LOAD $MAINT
// RUN
// DELETE LIBRARY-P,NAME-INST.ALL,RETAIN-S
// END
```

JOBSTR

For card input

```
// LOAD $QJOB
// RUN
```

For diskette input:

```
// LOAD $BICR
// FILE NAME-COPYIN,LABEL-filename,UNIT-I1
// FILE NAME-COPYO,LABEL-filename,UNIT-F1,RECORDS-number
// RUN
// TRANSFER
// END

// LOAD $MAINT
// FILE NAME-filename,UNIT-F1
// RUN
// COPY FROM-DISK,FILE-filename,TO-F1
// END

// LOAD $DELET
// RUN
// REMOVE LABEL-filename,UNIT-F1
// END
```

For executing a procedure if the procedure name is specified:

```
// INCLUDE procedurename
```

For deleting the procedure name if the NOSAVE parameter is specified:

```
// LOAD $MAINT
// RUN
// DELETE NAME-procedurename,LIBRARY-P
// END
```

LINES

```
// FORMS LINES- { number }
                { 66     }
```

LISTLIBR

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,NAME- { DIR          } ,LIBRARY- { S   } ,TO-PRINT
                      { library-name }           { P   }
                      { name.ALL     }           { O   }
                      { ALL          }           { R   }
                                                 { ALL }
                                                 { SYSTEM }
// END
```

LOG

```
// LOG { PRINTER } { ,EJECT   }
       { CRT     } { ,NOEJECT }
```

MRJE

```
 // LOAD $MRJE
[// FILE NAME-TDISKPR1,BLOCKS-number of blocks[,LABEL-filename] ]
[// FILE NAME-PDISKPR1,BLOCKS-number of blocks]
[// FILE NAME-PDISKPU1,BLOCKS-number of blocks]
 // RUN
 // END
```

ORGANIZE

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-filename-1 [,DATE-date] ,UNIT-F1
// FILE NAME-COPYO,LABEL-filename-2,RETAIN- { P / S / T } ,UNIT-F1
   or
// FILE NAME-COPYO,LABEL-filename-1,RETAIN- { retention-days / 1 } ,
   PACK-vol-id,UNIT-I1
// RUN
// COPYFILE OUTPUT-DISK[,DELETE-'position,character'],REORG-YES
// END
```

OVERRIDE

```
// LOAD $SETCF
// RUN
// SETR [ADDR-nn] [ ,LINE- { C / P / R / S / T } ][ ,SWTYP- { AA / MA / MC } ]
// END
```

PATCH

```
// LOAD $FEPCH
// RUN
// PATCH INPUT- { F1 / I1 } ,HEX- { NO / YES }
// END
```

REBUILD

```
// LOAD $REBLD
// RUN
```

RELOAD

```
// LOAD $LOAD
// FILE NAME-#LIBRARY,LABEL- { filename / #LIBRARY } [,DATE-date] [,PACK-vol-id]
   ,UNIT-I1
// RUN
```

REMOVE

```
// LOAD $MAINT
// RUN
// DELETE NAME-{library-name / name.ALL / ALL} ,LIBRARY-{S / P / O / R / ALL}
// END
```

RENAME

```
// LOAD $RENAM
// RUN
// RENAME LABEL-filename-1,NEWLABEL-filename-2 [,DATE-{mmddyy / ddmmyy / yymmdd}]
// END
```

RESTORE

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-{filename-1 / #SAVE / filename-2} [,DATE-date],UNIT-I1
// FILE NAME-COPYO[,LABEL-filename-2] [{,RECORDS-value-1 / ,BLOCKS-value-2}] [,UNIT-F1]
// RUN
// COPYALL TO-F1
   or
// COPYFILE OUTPUT-DISK,REORG-NO
// END
```

SAVE

```
// LOAD $COPY
// FILE NAME-COPYIN[,LABEL-filename-2] [,DATE-date] [,UNIT-F1]
// FILE NAME-COPYO [,RETAIN-{retention-days / 1}][,LABEL-{filename-2 / filename-1 / #SAVE,}],
   PACK-vol-id,UNIT-I1
// RUN
// COPYALL TO-I1
   or
// COPYFILE OUTPUT-DISK,REORG-NO
   or
// COPYADD
// END
```

SET

```
// LOAD $SETCF
[// IMAGE MEM,source-name]
[// DATE date]
// RUN
// SETCF [LINES-number] [,FORMAT- {MDY|DMY|YMD}] [,IMAGE- {YES|NO}]
// END
```

SETMICR

```
// LOAD $SETCF
// RUN
// SETR CYCLE- {Y|N}
// END
```

SPECIFY

```
// LOAD $SETCF
// RUN
// SETS [ADDR-nn] [,LINE- {C|P|S|T}] [,SWTYP- {AA|MA|MC}] [,ID-nnnnn]
// END
```

STATUS

```
// LOAD $STATS
// RUN
```

SYSLIST

```
// SYSLIST [PRINTER|CRT|OFF]
```

TOLIBR

```
// LOAD $MAINT
// FILE NAME-filename[,DATE-date],UNIT- {F1|I1}
// RUN
// COPY FROM-DISK,FILE-filename,RETAIN- {P|R},TO-F1
// END
```

TRACE

```
// LOAD $SETCF
// RUN
         ┌  ALL-Y    ┐
// TRACE │ {WAIT-N}  │ [,FDIOS-Y]  [,CSFDIOS-Y]  [,PUSH-Y]  [,PULL-Y]
         └ {WAIT-Y}  ┘ [,FDIOS-N]  [,CSFDIOS-N]  [,PUSH-N]  [,PULL-N]

   [,DISABLE-Y]  [,ENABLE-Y]  [,QUEUE-Y]  [,LDCS-Y]  [,LOADER-Y]
   [,DISABLE-N]  [,ENABLE-N]  [,QUEUE-N]  [,LDCS-N]  [,LOADER-N]

   [,XIENT-Y] [,XFER-Y]
   [,XIENT-N] [,XFER-N]
// END
```

TRANSFER

```
// LOAD $BICR
// FILE NAME-COPYIN,LABEL-filename-1[,DATE-date],UNIT- { F1 }
                                                      { I1 }
```

Transfer disk to diskette:

```
// FILE NAME-COPYO,LABEL-filename-1,PACK-vol-id [ ,retention-days ]
                                                [ ,1             ]
   UNIT-I1 or
```

Transfer diskette to disk, with ADD:

```
// FILE NAME-COPYO,LABEL- { filename-2 } [,DATE-date] [,UNIT-F1] or
                          { filename-1 }
```

Transfer diskette to disk, without ADD, size specified:

```
// FILE NAME-COPYO,LABEL-filename-1 { ,RECORDS-value-3 } [,UNIT-F1]
                                    { ,BLOCKS-value-4  }
```

Transfer diskette to disk, without ADD, using size of input file:

No COPYO FILE statement is generated.

```
// RUN
```

Diskette basic data exchange file to disk sequential file, or disk sequential, indexed, or direct file to diskette basic data exchange file:

```
[// TRANSFER]
```

Diskette basic data exchange file to disk sequential file with ADD:

```
// TRANSFER ADD-YES
```

Diskette basic data exchange file to disk indexed file, without ADD:

```
// TRANSFER ADD-NO,KEYLEN-value-1,KEYLOC-value-2

// END
```

IPL: See *initial program load.*

job stream: The input to the system. The job stream can contain OCL statements, utility control statements, and input data.

keyword: A group of characters, usually a word, that identifies a parameter in a control statement.

keyword parameter: A parameter that contains a keyword.

level: See *procedure level.*

library: An area on the disk that contains procedure members, source members, load members, and subroutine members, as well as areas required by the system control program.

library directory: The library component that contains information about each member in the library (for example, name and location).

library member: A named collection of records or statements in the library that can contain source statements, format descriptions, OCL statements, or executable instructions.

load member: A collection of instructions, stored in the library, that the system can execute to perform a particular function, whether the function is requested by the operator or specified in an OCL statement.

megabyte: One million bytes.

member: See *library member.*

message control statement: A statement that specifies the name and level of the message load member to be created.

message identification code (MIC): A 4-digit number associated with a specific error or informational message. The MIC is printed following the program identifier to allow the message to be reviewed after the program is signed off.

message load member: A special type of library member from which the SCP retrieves the text associated with a specific message identification code (MIC).

message source member: A special type of library source member containing control and message text statements.

message text statement: Statement in a message source member that specifies the message identification code (MIC) and text associated with that code.

MIC: See *message identification code.*

modem: A device that modulates and demodulates signals transmitted over communication facilities.

MULTI-LEAVING: The fully synchronized, two-directional transmission of a variable number of data streams between two computers using BSC facilities.

multipoint data link: One or more secondary stations on a common transmission line or communications facility where the primary station has controlling responsibilities for maintaining communications integrity and data link control.

multivolume file: A diskette file that resides on more than one diskette, or that can be expanded from one diskette to more than one diskette. See also *offline multivolume file.*

nested procedure: A procedure that is evoked by another procedure. A nested procedure is a procedure within a procedure.

network: A number of communication lines connecting a computer with remote terminals.

nonswitched line: A communication link between a remote station and computer that does not have to be established by dialing.

null entry: An entry that contains no value. For example, if CATALOG, I1 is entered, the first parameter position contains a null entry.

object program: A set of instructions in machine language. The object program is produced by the compiler from the source program.

OCL: See *operation control language.*

offline multivolume file: A multivolume file that is processed in segments by the system. Each segment is processed before the next segment is copied to or from the disk.

operation control language (OCL): The control language used to communicate with the system control program. OCL is composed of statements with which specific system functions are requested.

parameter: A variable that is given a constant value for a specific purpose or process.

point-to-point line: A communications facility connecting a single remote station to the computer.

positional parameters: Parameters in a statement that must appear in a designated sequence.

procedure: A named collection of related OCL statements, and possibly, utility control statements, that describe a specific function or set of functions. A procedure is evoked by a command statement or included OCL statements.

procedure level: Identifies the precedence of a particular procedure in a progression of nested procedures. For example, if procedure A evokes procedure B, which in turn evokes procedure C, procedure C is a third level procedure.

procedure member: A named collection of related OCL statements, and possible, utility control statements stored in the library.

PTAM: Pseudo tape access method.

pseudo tape access method (PTAM): An access method for processing simulated tape files on disk.

record mode: The mode of system operation in which data is transferred by the system one record at a time. The record mode of operation is used by the library maintenance utility ($MAINT) when placing user-generated source or procedure members into the library or a file.

relocation dictionary (RLD): The part of a load member used for adjusting main storage addresses when the member is moved to main storage.

rollout area: An area on disk that is allocated if inquiry support or offline multivolume support is selected. Programs interrupted by an inquiry request (INQ key pressed and the 1 option selected) are stored in the rollout area while the interrupting program is processed.

scheduler work area (SWA): An area on disk reserved for use by the scheduler program. The scheduler is part of the SCP.

scientific instruction set (SIS): The object program language, processed by the interpreter resident in the control storage increment, used to execute System/32 scientific programs.

SDLC: See synchronous data link control.

sector: A unit of data recorded on disk. A sector of data is the smallest amount of data that can be read from disk or the smallest amount of data that can be transferred by a single data transfer operation.

sector mode: The mode of system operation in which data is transferred by the system either one sector at a time or several sectors at a time. (Only whole sectors are transferred.) The sector mode of operation is used by the library maintenance utility ($MAINT) when placing user-generated members into the library or a file.

segment: See *file segment.*

sequential file: A file in which the order of records is determined by the order that they are put in the file. For example, the tenth record entered occupies the tenth record position. Sequential files can be processed using the consecutive, random by relative record number, and ADD-ROUT file processing methods.

SNA: See systems network architecture.

source member: A collection of records (such as RPG II specifications or sort sequence specifications) that are used as input for a program. Source members are stored in the library.

source program: A set of instructions that represents a particular job as defined by the programmer. These instructions are written in a programming language such as RPG II, and are translated by a compiler into an object program.

statement parameter: The portion of an IF expression that defines the action to be taken if the condition exists as specified. The statement parameter can be an OCL statement (except comment or end of data) or a utility control statement. The initial // of the statement is not entered as part of the expression. CANCEL and RETURN are also valid entries in the statement parameter.

Index

Technical Newsletter

This Newsletter No.	GN21-7993
Date	22 November 1978
Base Publication No.	GC21-7593-3
File No.	S32-36
Previous Newsletters	GN21-7939

IBM System/32
System Control Programming
Reference Manual

This technical newsletter, a part of version 8, modification 0 of IBM System/32 (Program Product 5725-SC1), provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions unless specifically altered. Pages to be inserted and/or removed are:

Title Page, Edition Notice	105, 106	187, 188
iii through xii	111, 112	207, 208
xiii, xiv (added)	119, 120	211, 212
17, 18	135 through 138	212.1, 212.2 (added)
21, 22	141, 142	225, 226
37, 38	159, 160	251, 252
45, 46	160.1, 160.2 (added)	265, 266
46.1, 46.2 (added)	161, 162	267, 268
55 through 58	162.1, 162.2 (added)	272.1, 272.2
69 through 72	165 through 168	295 through 298
93, 94	168.1, 168.2	301, 302
94.1, 94.2 (added)	179, 180	302.1, 302.2 (added)
95, 96		331 through 340

Changes to text and illustrations are indicated by a vertical line at the left of the change.

Summary of Amendments

- Miscellaneous editorial and technical changes
- $FREE and $RENAM utility programs

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

Technical Newsletter

This Newsletter No. GN21-7939
Date 25 November 1977

Base Publication No. GC21-7593-3
File No. S32-36

Previous Newsletters

IBM System/32
System Control Programming
Reference Manual

This technical newsletter, a part of version 7, modification 00 of the IBM System/32 (Program Product 5725-SC1), provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions unless specifically altered. Pages to be inserted and/or removed are:

ix, x
15, 16
55 through 60
69, 70
77 through 80
80.1, 80.2 (added)
93, 94
103 through 106
139, 140
175, 176
195, 196
205, 206
225 through 230
230.1, 230.2 (added)
241, 242
249 through 252
255, 256
261, 262
264.1 through 264.6 (added)
265, 266
266.1, 266.2 (added)
267, 268
268.1, 268.2 (added)
272.1 through 272.4 (added)
281, 282
293, 294
297, 298
301, 302
327, 328

Changes to text and illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by the symbol • at the left of the caption.

Summary of Amendments

- Add a version update instruction summary.
- Miscellaneous changes.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM S/32 System Control Programming Reference Manual (File No. S32-36) Printed in U.S.A. GC21-7593-4