

**SYSTEM** / **32**

**IBM System/32  
System Control Programming  
Reference Manual**

**32**

*IBM System/32  
System Control Programming  
Programming Information*

GC21-7593-1  
File No. S32-36

**Program Number  
5725-SC1**

## **Second Edition (May 1975)**

This is a major revision of GC21-7593-0 and obsoletes GC21-7593-0 and Technical Newsletters GN21-5314 and GN21-7808.

This edition applies to version 02, modification 00 of IBM System/32 (Program Number 5725-SC1); to version 01, modification 00 of the IBM System/32 Utilities Program (Program Number 5725-UT1); to version 01, modification 00 of IBM System/32 RPG II (Program Number 5725-RG1); and to all subsequent versions and modifications unless otherwise indicated in new editions or technical newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the branch office serving your locality.

A Reader's Comment Form is at the back of this publication. If the form is gone, address your comments to Publications, Department 245, Rochester, Minnesota 55901.

© Copyright International Business Machines Corporation 1975

This manual provides system programmers the information needed to run application programs on IBM System/32 and use the system procedures and utility programs provided with IBM System/32.

This manual provides:

- 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, as well as 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, and an example of using OCL statements and procedures to perform applications
- A description of each system utility program provided with IBM System/32, including a description of associated utility control statements
- A description of how to configure, 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 system procedure contents
- Standard characters for IBM System/32 printers

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

#### **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.

→ *The IBM Diskette for Standard Data Interchange*, GA21-9182, describes the diskette data format for standard data interchange.

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

# Contents

<b>LIST OF ABBREVIATIONS . . . . .</b>	<b>viii</b>	<b>IBM SCP PROCEDURE DESCRIPTIONS . . . . .</b>	<b>55</b>
<b>HOW TO USE THIS MANUAL . . . . .</b>	<b>ix</b>	ALTERBSC Procedure . . . . .	56
<b>PART 1. OCL STATEMENTS . . . . .</b>	<b>1</b>	ALTERBSC Command Statement Format . . . . .	56
<b>INTRODUCTION TO OCL STATEMENTS . . . . .</b>	<b>3</b>	ALTERBSC Parameters . . . . .	56
What is OCL . . . . .	3	BACKUP Procedure . . . . .	57
OCL Statements and the Job . . . . .	3	BACKUP Command Statement Format . . . . .	57
System Configuration . . . . .	4	BACKUP Parameters . . . . .	58
<b>CODING OCL STATEMENTS . . . . .</b>	<b>5</b>	CATALOG Procedure . . . . .	58
Types of Information Conveyed in OCL Statements . . . . .	5	CATALOG Command Statement Format . . . . .	58
Identifiers . . . . .	5	CATALOG Parameters . . . . .	58
Parameters . . . . .	6	COMPRESS Procedure . . . . .	59
General OCL Coding Rules . . . . .	6	COMPRESS Command Statement Format . . . . .	59
Comments . . . . .	7	COMPRESS Parameters . . . . .	59
Continuation . . . . .	7	COPY11 Procedure . . . . .	59
<b>OCL STATEMENT TABLES . . . . .</b>	<b>9</b>	COPY11 Command Statement Format . . . . .	59
<b>OCL STATEMENT DESCRIPTIONS . . . . .</b>	<b>15</b>	COPY11 Parameters . . . . .	60
COMPILE Statement . . . . .	15	COPY11 Example . . . . .	60
DATE Statement . . . . .	16	CREATE Procedure . . . . .	61
FILE Statement . . . . .	17	CREATE Command Statement Format . . . . .	61
FORMS Statement . . . . .	23	CREATE Parameters . . . . .	61
IMAGE Statement . . . . .	24	CREATE Example . . . . .	61
INCLUDE Statement . . . . .	26	DATE Procedure . . . . .	62
LOAD Statement . . . . .	27	DATE Command Statement Format . . . . .	62
LOG Statement . . . . .	28	DATE Parameters . . . . .	62
MEMBER Statement . . . . .	29	DELETE Procedure . . . . .	63
PAUSE Statement . . . . .	30	DELETE Command Statement Format . . . . .	63
RUN Statement . . . . .	31	DELETE Parameters . . . . .	63
SWITCH Statement . . . . .	32	DELETE Example . . . . .	63
SYSLIST Statement . . . . .	33	DISPLAY Procedure . . . . .	64
* (Comment) Statement . . . . .	33	DISPLAY Command Statement Format . . . . .	64
/* (End of Data) Statement . . . . .	34	DISPLAY Parameters . . . . .	64
/** (Message) Statement . . . . .	34	DISPLAY Example . . . . .	64
<b>PART 2. PROCEDURES . . . . .</b>	<b>35</b>	FROMLIBR Procedure . . . . .	65
<b>INTRODUCTION TO PROCEDURES . . . . .</b>	<b>37</b>	FROMLIBR Command Statement Format . . . . .	65
IBM SCP Procedures . . . . .	38	FROMLIBR Parameters . . . . .	66
Creating a Procedure . . . . .	38	FROMLIBR Examples . . . . .	67
Evoking a Procedure . . . . .	39	HISTORY Procedure . . . . .	68
Keyboard Entry of the INCLUDE Statement . . . . .	39	HISTORY Command Statement Format . . . . .	68
Command Key Request . . . . .	39	HISTORY Parameters . . . . .	68
Evoking a Procedure from Another Procedure . . . . .	41	INIT Procedure . . . . .	69
Procedure Execution . . . . .	41	INIT Command Statement Format . . . . .	69
Procedure Parameters . . . . .	42	INIT Parameters . . . . .	69
Modifying a Procedure Job Stream . . . . .	42	INIT Examples . . . . .	70
Substitution . . . . .	42	LINES Procedure . . . . .	71
Conditional Expressions: IF and ELSE . . . . .	44	LINES Command Statement Format . . . . .	71
Example of Procedure Coding . . . . .	47	LINES Parameters . . . . .	71
FILEBKUP Procedure . . . . .	47	LISTLIBR Procedure . . . . .	72
FILEBKUP Parameters . . . . .	47	LISTLIBR Command Statement Format . . . . .	72
<b>IBM SCP COMMAND STATEMENTS . . . . .</b>	<b>51</b>	LISTLIBR Parameters . . . . .	72
		LISTLIBR Examples . . . . .	73
		LOG Procedure . . . . .	74
		LOG Command Statement Format . . . . .	74
		LOG Parameters . . . . .	74
		ORGANIZE Procedure . . . . .	75
		ORGANIZE Command Statement Format . . . . .	75
		ORGANIZE Parameters . . . . .	75
		ORGANIZE Examples . . . . .	76
		OVERRIDE Procedure . . . . .	77

OVERRIDE Command Statement Format . . . . .	77
OVERRIDE Parameters . . . . .	77
REBUILD Procedure . . . . .	78
REBUILD Command Statement Format . . . . .	78
REBUILD Parameters . . . . .	78
RELOAD Procedure . . . . .	79
RELOAD Command Statement Format . . . . .	79
RELOAD Parameters . . . . .	79
REMOVE Procedure . . . . .	80
REMOVE Command Statement Format . . . . .	80
REMOVE Parameters . . . . .	80
REMOVE Examples . . . . .	80
RESTORE Procedure . . . . .	81
RESTORE Command Statement Format . . . . .	81
RESTORE Parameters . . . . .	81
RESTORE Examples . . . . .	82
SAVE Procedure . . . . .	83
SAVE Command Statement Format . . . . .	83
SAVE Parameters . . . . .	83
SAVE Examples . . . . .	84
SET Procedure . . . . .	85
SET Command Statement Format . . . . .	85
SET Parameters . . . . .	85
STATUS Procedure . . . . .	86
STATUS Command Statement Format . . . . .	87
STATUS Parameters . . . . .	87
SYSLIST Procedure . . . . .	88
SYSLIST Command Statement Format . . . . .	88
SYSLIST Parameters . . . . .	88
TOLIBR Procedure . . . . .	89
TOLIBR Command Statement Format . . . . .	89
TOLIBR Parameters . . . . .	89
TRANSFER Procedure . . . . .	91
TRANSFER Command Statement Format . . . . .	91
TRANSFER Parameters . . . . .	92
TRANSFER Examples . . . . .	93
<b>PART 3. USING OCL STATEMENT AND PROCEDURES . . . . .</b>	<b>95</b>
<b>CREATING DISK AND DISKETTE FILES . . . . .</b>	<b>97</b>
Disk File . . . . .	97
Obtaining Space for a File . . . . .	97
Describing a File . . . . .	97
Diskette File . . . . .	98
Offline Multivolume File . . . . .	99
Purpose of Offline Multivolume Files . . . . .	99
Creating an Offline Multivolume File . . . . .	100
Reading an Offline Multivolume File . . . . .	101
Offline Multivolume File Restrictions and Considerations . . . . .	101
<b>LOADING AND RUNNING PROGRAMS . . . . .</b>	<b>105</b>
IBM Programs . . . . .	105
Object Programs Using One Disk File . . . . .	105
Object Programs Using More Than One Disk File . . . . .	105
Object Programs Using One Disk File and External Indicators . . . . .	106
<b>OCL AND PROCEDURE EXAMPLE . . . . .</b>	<b>107</b>

<b>PART 4. SYSTEM UTILITY PROGRAMS . . . . .</b>	<b>111</b>
<b>INTRODUCTION TO THE SYSTEM UTILITY PROGRAMS . . . . .</b>	<b>113</b>
WRITING UTILITY CONTROL STATEMENTS . . . . .	113
Rules for Coding Utility Control Statements . . . . .	113
Conventions for Describing Utility Control Statement Formats . . . . .	114
<b>UTILITY PROGRAM DESCRIPTIONS . . . . .</b>	<b>115</b>
\$BACK—Backup Library Utility Program . . . . .	116
\$BACK Utility Control Statement Format . . . . .	116
\$BACK OCL Sequence . . . . .	116
\$BICR—Standard Interchange Utility Program . . . . .	117
\$BICR Utility Control Statement Formats . . . . .	117
\$BICR Parameters . . . . .	117
\$BICR OCL and Utility Control Statement Sequence . . . . .	118
\$BICR Example . . . . .	118
\$BUILD—Alternate Sector Rebuild Utility Program . . . . .	119
Bypass Unreadable Data . . . . .	121
Correct Unreadable Data . . . . .	121
\$BUILD Utility Control Statement Format . . . . .	121
\$BUILD OCL Sequence . . . . .	121
\$COPY—Disk Copy/Display Utility Program . . . . .	122
\$COPY Utility Control Statement Formats . . . . .	122
\$COPY Parameters . . . . .	124
\$COPY Parameter Summary . . . . .	126
\$COPY OCL and Utility Control Statement Sequence . . . . .	129
\$COPY Examples . . . . .	131
\$DELET—File Delete Utility Program . . . . .	132
\$DELET Utility Control Statement Formats . . . . .	132
\$DELET Parameters . . . . .	133
\$DELET Parameter Summary . . . . .	133
\$DELET OCL and Utility Control Statement Sequence . . . . .	134
\$DELET Examples . . . . .	134
\$DUPRD—Diskette Copy Utility Program . . . . .	136
\$DUPRD Utility Control Statement Formats . . . . .	136
\$DUPRD Parameters . . . . .	136
\$DUPRD Parameter Summary . . . . .	136
\$DUPRD OCL And Utility Control Statement Sequence . . . . .	137
\$DUPRD Examples . . . . .	138
\$HIST—History File Display Utility Program . . . . .	139
\$HIST Utility Control Statement Formats . . . . .	139
\$HIST Parameters . . . . .	139
\$HIST OCL and Utility Control Statement Sequence . . . . .	139
\$HIST Examples . . . . .	140
\$INIT—Diskette Labeling and Initialization Utility Program . . . . .	141
Initialize (FORMAT and FORMAT2) . . . . .	141
Delete (DELETE) . . . . .	141
Rename (RENAME) . . . . .	142
Diskette Defects Encountered During Processing . . . . .	142
\$INIT Utility Control Statement Formats . . . . .	142
\$INIT Parameters . . . . .	143
\$INIT Parameter Summary . . . . .	143
\$INIT OCL and Utility Control Statement Sequence . . . . .	145
\$INIT Examples . . . . .	145

\$LABEL--VTOC Display Utility Program . . . . .	146
Sample VTOC Displays . . . . .	146
\$LABEL Utility Control Statement Formats . . . . .	149
\$LABEL Parameters . . . . .	149
\$LABEL OCL and Utility Control Statement Sequence . . . . .	149
\$LOAD--Reload Library Utility Program . . . . .	150
Inquiry Option . . . . .	151
Offline Option . . . . .	152
\$LOAD Utility Control Statement Format . . . . .	152
\$LOAD OCL Sequence . . . . .	152
\$MAINT--Library Maintenance Utility Program . . . . .	153
System Library File (#LIBRARY) . . . . .	153
Allocate Function . . . . .	155
Copy Function . . . . .	156
Delete Function . . . . .	172
\$MGBLD--Create Message Member Utility Program . . . . .	176
\$MGBLD Utility Control Statement Format . . . . .	176
\$MGBLD Parameters . . . . .	176
\$MGBLD OCL and Utility Control Statement Sequence . . . . .	176
Message Source Member . . . . .	177
An Example of Creating a Message Source and Load Member . . . . .	178
\$PACK--Disk Reorganization Utility Program . . . . .	179
\$PACK Utility Control Statement Format . . . . .	179
\$PACK OCL Sequence . . . . .	179
\$REBLD--Rebuild Data File Utility Program . . . . .	180
\$REBLD Utility Control Statement Format . . . . .	181
\$REBLD OCL Sequence . . . . .	181
\$SETCF--Set Utility Program . . . . .	181
Set the System Environment . . . . .	181
Set the BSCA Environment . . . . .	183
Override RPG BSCA Specifications . . . . .	184
Set Functions to be Traced . . . . .	185
\$STATS--Status Display Utility Program . . . . .	187
\$STATS Utility Control Statement Format . . . . .	188
\$STATS OCL Sequence . . . . .	188
<b>PART 5. SYSTEM CONFIGURATION, INSTALLATION, AND MODIFICATION . . . . .</b>	<b>189</b>
<b>INTRODUCTION TO SYSTEM CONFIGURATION, INSTALLATION, AND MODIFICATION . . . . .</b>	<b>191</b>
<b>SYSTEM CONFIGURATION AND INSTALLATION . . . . .</b>	<b>193</b>
Diskettes Required . . . . .	193
System Configuration Steps . . . . .	194
System Installation Steps . . . . .	196
Calculating the Number of Backup Diskettes Required for the System . . . . .	198
APPLYPTF Procedure . . . . .	199
APPLYPTF Command Statement Format . . . . .	199
APPLYPTF Parameters That are Not Prompted . . . . .	199
Prompted Parameters for APPLYPTF . . . . .	200
CNFIGSCP Procedure . . . . .	200
CNFIGSCP Command Statement Format . . . . .	201
Prompted Parameters for CNFIGSCP . . . . .	201
INSTALL Procedure . . . . .	202
INSTALL Command Statement Format . . . . .	202
INSTALL Parameters that are Not Prompted . . . . .	202
Prompted Parameters for INSTALL . . . . .	202

<b>PROGRAM PRODUCT INSTALLATION AND VERIFICATION . . . . .</b>	<b>203</b>
Program Product Installation . . . . .	203
To Install a Program Product . . . . .	203
To Create a Backup Copy of a Program Product . . . . .	203
Program Product Installation Verification . . . . .	204
SEU Installation Verification . . . . .	204
RPG II Installation Verification . . . . .	206

<b>SYSTEM MODIFICATION . . . . .</b>	<b>209</b>
Library Requirements . . . . .	209
Deleting From the Library . . . . .	209
Determining Space Available in the Library . . . . .	209
Determining Space Available on the Disk . . . . .	210
Selecting Members to Delete . . . . .	210
Deleting Members . . . . .	211
RELOAD Display . . . . .	211
If Values in the RELOAD Display are Correct . . . . .	212
If Values in the RELOAD Display are to be Changed . . . . .	213

<b>APPENDIX A. RECORDS, BLOCKS, AND SECTOR CONVERSION . . . . .</b>	<b>215</b>
Records to Blocks Conversion for Disk . . . . .	215
Determining the Number of Sequential or Direct File Blocks . . . . .	215
Determining the Number of Indexed File Blocks . . . . .	215
Disk Sector Number to Block Number Conversion . . . . .	216
Disk Block Number to First Sector in Block Conversion . . . . .	216

<b>APPENDIX B. HEX AND DECIMAL CONVERSION . . . . .</b>	<b>217</b>
Hex and Decimal Chart . . . . .	217
Hex to Decimal Example . . . . .	218
Decimal to Hex Example . . . . .	218

<b>APPENDIX C. DISKETTE FORMATS AND DISKETTE DATA FILES . . . . .</b>	<b>219</b>
Diskette Formats . . . . .	219
Diskette Data Files . . . . .	219
Standard Interchange Files . . . . .	219
System Files . . . . .	220

<b>APPENDIX D. IBM SCP SERVICE PROCEDURES . . . . .</b>	<b>223</b>
APAR Procedure . . . . .	224
APAR Command Statement Format . . . . .	224
APAR Parameters . . . . .	224
BUILD Procedure . . . . .	225
BUILD Command Statement Format . . . . .	225
BUILD Parameters . . . . .	225
DUMP Procedure . . . . .	225
DUMP Command Statement Format . . . . .	225
DUMP Parameters . . . . .	226
PATCH Procedure . . . . .	226
PATCH Command Statement Format . . . . .	226
PATCH Parameters . . . . .	227
TRACE Procedure . . . . .	227
TRACE Command Statement Format . . . . .	228
TRACE Parameters . . . . .	229

<b>APPENDIX E. IBM SCP PROCEDURE CONTENTS . . . . .</b>	<b>231</b>
ALTERBSC . . . . .	231
APAR . . . . .	231
APPLYPTF . . . . .	231

BACKUP . . . . .	232
BUILD . . . . .	232
CATALOG . . . . .	232
CNFIGSCP . . . . .	232
COMPRESS . . . . .	234
COPY11 . . . . .	234
CREATE . . . . .	234
DATE . . . . .	234
DELETE . . . . .	234
DISPLAY . . . . .	235
DUMP . . . . .	235
FROMLIBR . . . . .	235
HISTORY . . . . .	236
INIT . . . . .	236
INSTALL . . . . .	236
LINES . . . . .	236
LISTLIBR . . . . .	236
LOG . . . . .	237
ORGANIZE . . . . .	237
OVERRIDE . . . . .	237
PATCH . . . . .	237
REBUILD . . . . .	237
RELOAD . . . . .	237
REMOVE . . . . .	238
RESTORE . . . . .	238
SAVE . . . . .	238
SET . . . . .	238
STATUS . . . . .	239
SYSLIST . . . . .	239
TOLIBR . . . . .	239
TRACE . . . . .	239
TRANSFER . . . . .	239
<b>APPENDIX F. IBM SYSTEM/32 CHARACTERS . . . . .</b>	<b>241</b>
<b>APPENDIX G. POLLING AND ADDRESSING CHARACTERS FOR SYSTEM/32 TRIBUTARY STATIONS. . . . .</b>	<b>245</b>
EBCDIC . . . . .	245
ASCII . . . . .	246
<b>APPENDIX H. ASCII AND EBCDIC CODES . . . . .</b>	<b>247</b>
EBCDIC Codes . . . . .	248
ASCII Codes . . . . .	249
<b>APPENDIX I. DATA LINK CONTROL CHARACTERS . . . . .</b>	<b>251</b>
<b>GLOSSARY . . . . .</b>	<b>253</b>
<b>INDEX . . . . .</b>	<b>257</b>

## List of Abbreviations

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

CE	IBM customer engineer
DTF	define the file
IOB	input/output block
IOS	input/output supervisor
IPL	initial program load
K	1024 bytes
MIC	message identification code
OCL	operation control language
PID	program information department
PLCA	program level communication area
PTF	program temporary fix
RIB	request indicator byte
SCA	system communication area
SCP	system control programming
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 use of OCL and procedures to perform applications. Part 4 describes system utility programs. Part 5 describes system configuration, installation, and modification. Part 5 also describes 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
- Where each OCL statement is placed in relation to others and when it is needed
- 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
- 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 load and run programs
- How OCL and procedures are used to perform applications

**Part 4.**

Refer to Part 4 if you want to know:

- What system utility programs are supplied with IBM System/32
- What the function of each utility program is
- What OCL statements and utility control statements are necessary to request 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 if you want verification that they are installed correctly)
- Modifying an installed system by deleting certain system control programming components or program product functions from the library

**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 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 do not, however, contain data to be processed.
- *Subroutine members.* Subroutine members contain subroutines that can be combined with other 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 directly to the system as 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 (filename, etc) to utility programs. Utility control statements can be included with OCL statements in procedures.

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

## **SYSTEM CONFIGURATION**

IBM System/32 System Control Programming runs on all models of System/32 and supports all available System/32 features.

## TYPES OF INFORMATION CONVEYED IN OCL STATEMENTS

OCL statements contain, at the most, two types of information: an *identifier* and *parameters*. An identifier is information that distinguishes one OCL statement from another. A parameter is information supplied 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

*Note:* Most OCL statements begin with //. However, // is not used if the identifier is \* or /\*. Also, // is not required with a command statement (a simplified form of the INCLUDE OCL statement).

### 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.

OCL statement identifiers that follow // 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 follow // are:

```
* (comment)
```

```
/* (end of data)
```

For example, in the statement

```
* END OF JOB
```

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

## Parameters

Some statements need parameters, others do not. Parameters are either *symbolic* or *keyword* parameters. In the following statement, \$COPY is a *symbolic parameter*—the symbolic name of a system utility program:

```
// LOAD $COPY
```

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

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

A keyword parameter contains a *keyword* (NAME, UNIT, and LABEL are 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, F1, and PAYROLL are values in the preceding sample OCL statement). If a value does follow a keyword, it must be separated from the keyword by a hyphen.

## 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 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 and number, represent information that you must supply.

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 blanks must be left between the // and the statement identifier (LOAD, RUN, etc).
- One or more blanks must be left between the end of the statement identifier and the first parameter.
- 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 (parameter name) from a value.



## Comments

Comments can contain any character except the question mark (?). Any combination of valid characters can be included in the following places:

- Following the \* on the OCL comment statement.
- After the last parameter. Leave one or more blanks between the last parameter and your comment.
- After the identifier on statements without parameters. Leave one or more blanks between the identifier and your comments.

*Note:* If you want a comment but no parameters 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.

Examples of the preceding three ways to include comments in OCL statements are:

1. \*THIS IS AN EXAMPLE OF A COMMENT STATEMENT  
In the example above the comment is THIS IS AN EXAMPLE OF A COMMENT STATEMENT.
2. // LOAD \$COPY LOAD THE DISK COPY UTILITY  
In this example the comment is LOAD THE DISK COPY UTILITY.
3. // RUN RUN THE DISK COPY UTILITY  
The comment here is RUN THE DISK COPY UTILITY.
4. // INCLUDE PROC, MAIN PROCEDURE  
The comment here is MAIN PROCEDURE.

## Continuation

No OCL statement except a FILE statement (see index entry: *// FILE statement* for a description of FILE statements) can exceed 120 characters, including blanks and commas. Because of the large number of parameters possible in FILE statements, FILE statements can exceed 120 characters if composed of two or more OCL records to express a single FILE statement. However, no individual OCL record, including those used to express FILE statements, can exceed 120 characters.

Expressing a single statement in two or more records is called *continuation*. Only FILE statements can use continuation. 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 OCL records are used to express a single FILE statement. In the second example, two OCL records express one FILE statement.

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

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

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 on use of 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 a choice on your part, depending on the functions you want the statement to perform. Refer to Figure 3 to see which parameters are available. Those parameters that are capitalized must be coded along with the keyword parameter.

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 <i>except</i> 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 has been changed
// INCLUDE	Identifies the procedure member to be merged into job stream	Can be placed anywhere among the OCL statements	Can be no more than 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	Changes the message member from which messages are to 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	Required in the job stream for the program to be run
// SWITCH	Used to set 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	Used to explain the job; does not affect the program in operation	Can be placed anywhere among the OCL statements	The * must be in the first position.
/* (End of Data)	Indicates the end of a data file read from the keyboard	Last record of an input data file	Not recognized in a procedure
// * (Message)	Displays message to 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
// FILE (Disk)	NAME-filename or NAME-COPYIN or NAME-COPYO	Name the program uses to refer to the file.  For certain utility program, names the input file when used in conjunction with the LABEL parameter.  For certain utility programs, names the output file when used in conjunction 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 by which your file is identified on disk
	RECORDS-number or BLOCKS-number	Amount of space needed on the disk for a file
	LOCATION-blocknumber	Number of block on which 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 by which your file is identified on the diskette
	RETAIN-retention-days	The number of days a file will be retained before it expires. Maximum is 998. If 999, the expiration date is set to a value that cannot be met.

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

Statement	Parameter	Meaning of Parameter
// FILE (Diskette) (continued)	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 or	Characters which follow are in hexadecimal form.
	CHAR or	Characters which follow are in EBCDIC form.
	MEM or MEMBER	Characters are identified as to 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 data.
// INCLUDE	procedure-name	Name that identifies the procedure member in the library.
	procedure parameters	Parameters (as many as ten) 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 member used for program product level 1 messages. If name is 0, the member name is cleared.
	PROGRAM2-name	Name of member used for program product level 2 messages. If name is 0, the member name is cleared.
	USER1-name	Name of member used for user program's level 1 and OCL message statements. If name is 0, the member name is cleared.

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

Statement	Parameter	Meaning of Parameter
// MEMBER (continued)	USER2-name	Name of member used for user program's level 2 messages. If name is 0, the member name is cleared.
// PAUSE	none	
// RUN	none	
// SWITCH	nnnnnnn 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. (During IPL the printer is assigned.)
	OFF	Ignore request for SYSLIST output.
*(Comment)	none	
/* (End of data)	none	
// * (Message)	msg-id	The identification of a message in the assigned USER1 message member.
	'message'	A character string which 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 <i>source program</i> to be compiled. A source program is a collection of statements, such as RPG II specifications, that can be translated into an <i>object program</i> . An object program is a program that can be loaded into main storage and run. Source programs are stored in library 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	<code>// COMPILE SOURCE-name</code>
Contents	<b>SOURCE:</b> This parameter specifies the name of the source member to be compiled as a source program.
Example	<p>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.)</p> <pre>// LOAD #RPG // COMPILE SOURCE-PROG3 // RUN</pre>

## 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 date is also used for the creation date of the disk or diskette.

**Placement** A DATE statement can be given during IPL. 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 mmddy or yymmdd or ddmmyy

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

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	mmddy
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

## FILE Statement

Function	The FILE statement supplies the system with information about disk files. The system uses this information to read records from and write records on the disk and diskettes.
Placement	A FILE statement is required 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.
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, which are 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-yyymmdd

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, periods, single quotes ('), and blanks. Because the question mark (?) has a special meaning in procedures (see index entry: *procedure parameters*) and certain control statements, the question mark should not be used in filenames. 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 will be 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 name 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 name from the NAME parameter is used. If the file is an existing file, a LABEL parameter is required when the name a program uses to refer to the file differs from the name by which the file is identified on the disk. The name can be any combination of characters (numeric, alphabetic, #, \$, or @). The first character must be alphabetic, #, \$, or @. The number of characters must not exceed eight.

**RECORDS or BLOCKS:** The RECORDS or BLOCKS parameter is needed for files that are being created. The parameter tells the system the amount of space needed on the disk for the file.

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 or a diskette in one read/write operation. Disk blocks available to the user vary with disk size (K = 1024 bytes, one *megabyte* = one million bytes):

5.0 Megabytes Disk	9.1 Megabytes Disk
1968 blocks	3576 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) is allocated.

If RECORDS is used, the number can be up to six 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 disk size:

5.0 Megabyte Disk	9.1 Megabyte Disk
1985	3593

LOCATION is required in only two cases:

- You are creating another version of an existing output 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 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. A different creation date, taken from the current job date, will exist for the new file.

*Note:* Use LOCATION with caution. 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 two procedures do not display a message to notify the operator of the new locations. (For more information on the COMPRESS and RESTORE procedures, see index entries: *COMPRESS procedure* and *RESTORE procedure*.) To determine the current location of a file, use the CATALOG procedure (see index entry: *CATALOG procedure*).

*RETAIN:* The RETAIN parameter is used to classify files according to their use: 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 delete it.
- Another file with the same LABEL name is loaded into the area occupied by the temporary file, changing only the data; that is, space (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 has deleted the permanent file (see index entry: *DELETE procedure*).

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. 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.

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).

*DATE:* The DATE parameter identifies the creation date of the file, though it 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 (mmddy), day-month-year (ddmmy) or year-month-day (yymmdd). However, the form chosen must conform to that of the previous date statement.

#### *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 that will 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 and are as follows (keywords are in capital letters):

```
NAME-filename (in program)  
UNIT-I1  
LABEL-filename (on diskette)  
RETAIN-retention-days  
DATE-mmddy or DATE-ddmmy 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 associated 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, periods, single quotes ('), and blanks. Because the question mark (?) has a special meaning in procedures (see index entry: *procedure parameters*) and certain control statements, the question mark should not be used in filenames. 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 will be on the disk or on a diskette.

The code for the UNIT parameter on a FILE statement for the diskette is 11. This keyword and code must be specified on a diskette FILE statement.

**LABEL:** The keyword for the parameter is LABEL. It must be followed by the name of the file on the diskette. The name can be any combination of characters (numeric, 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.

Only one version of a file can be created under a given name on any one diskette.

**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. If RETAIN is omitted, 1 is assumed. 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's 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 have been 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 its 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 existing unexpired 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, name, expiration date, and creation date (job date) are written on the disk 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 (mmddy), day-month-year (ddmmy), 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 alphameric characters. The number of characters must not exceed six.

| The PACK parameter vol-id will be compared to the vol-id of the inserted diskette. If they are unequal, a message will be displayed to the operator and he will have 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 an input diskette file; however, it is recommended that you ensure 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 named 666666.
- The system will choose the file's location on the diskette.

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: <i>LINES procedure</i> and <i>SET procedure</i> ) 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	<p><i>LINES</i>: 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 LINES parameter remains in effect until a SET procedure (see index entry: <i>SET procedure</i>) is used to change the variable, another FORMS statement is received, an IPL is performed, or a 3 option 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 the program.</p> <p>The printer will skip (overflow) to a new page when there are six lines remaining on the page. For example, if LINES-84 is specified, the printer skips to a new page after printing line 78. If LINES-7 is specified, there would be one printed line per page. When six or less lines are specified there is printing on every line (no overflow).</p> <p><i>Note</i>: RPG II programs can specify their own overflow rules to override the values specified in a // FORMS statement. However, if an RPG II program does not specify overflow rules, those specified in a // FORMS statement are used for the program.</p>
Example	<p>The following statement tells the system that the forms length is 50 lines per page.</p> <pre>// FORMS LINES-50</pre>

## 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 the print belt is replaced with one having 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 when IPL occurs. 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 of two things:

- The new print belt characters are to be read from the keyboard (HEX or CHAR and number parameters); or
- The new print belt characters are to be read from a source member in the library (MEM,name or MEMBER,name parameters).

### *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 hex form. (See Appendix F for the hex form of standard characters.)

**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 will be prompted for). This number must not exceed 384 when the characters are hex, 192 when 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.
- The characters must begin in position 1 to continue on another line.

## Examples

The IMAGE statements in Examples A and B following tell the system that the new characters are to be entered from the keyboard. The format parameter in example A indicates that the new characters are in hex 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
```

### *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 member. Name identifies the library source member containing the characters.

In the following example, the name parameter indicates that the characters are to be found in a library source member named BELT: // IMAGE MEM,BELT

A // IMAGE statement specifying the format as either hex (HEX) or alphameric (CHAR) and specifying the number of characters in the source member is required as the first record within the source member. Assume that the source member BELT was to contain 13 characters (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \*, -, and +) to be read. The member BELT would contain the following two statements:

```
// IMAGE CHAR,13  
0123456789* -+
```

## 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 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 may be any combination of characters except question marks, commas, quotes, 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 has been omitted. JOE and SAM are two parameters which 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 { // LOAD PAYROLL
in place of the INCLUDE  { // FILE LABEL-WEEKLY,...
statement                 { // FILE...
                          // RUN
```

### LOAD Statement

- Function** The LOAD statement identifies the load program to be executed.
- Placement** The LOAD statement must be the first statement in a set of statements for a LOAD 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	<p>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.</p> <p><i>Note:</i> 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. During IPL, only the display screen is assigned for displaying messages and OCL statements.</p>										
Placement	The LOG statement can be used anywhere within the set of OCL statements for a program.										
Format	<pre>// LOG CRT      ,EJECT            or      or            PRINTER NOEJECT</pre>										
Contents	<table><thead><tr><th>Parameter</th><th>Meaning</th></tr></thead><tbody><tr><td>CRT</td><td>Use only the display screen.</td></tr><tr><td>PRINTER</td><td>Use the printer and the display screen.</td></tr><tr><td>EJECT</td><td>Skip to line 1 of the next page at end of job. EJECT is assumed if only CRT or PRINTER is specified.</td></tr><tr><td>NOEJECT</td><td>Do not skip to line 1 of the next page at end of job.</td></tr></tbody></table>	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 only CRT or PRINTER is specified.	NOEJECT	Do not skip to line 1 of the next page at end of job.
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 only CRT or PRINTER is specified.										
NOEJECT	Do not skip to line 1 of the next page at end of job.										
Example	<p>The following example specifies that messages and OCL statements are to be displayed on both the display screen and the printer.</p> <pre>// LOG PRINTER</pre>										

## MEMBER Statement

Function	<p>The MEMBER statement allows the user to change the message member from which messages are to come.</p> <p>There are four types of message members: PROGRAM1, PROGRAM2, USER1, and USER2.</p> <p>PROGRAM is used by IBM program products to assign names to associated message members.</p> <p>USER means that the messages are for user-generated programs and OCL statements.</p> <p>Each message member has a level 1 and a level 2. 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.</p>								
Placement	<p>The MEMBER statement can be placed anywhere among OCL statements.</p>								
Format	<p>// MEMBER parameters</p>								
Contents	<p>All the parameters are keyword parameters and are as follows (keywords are in capital letters):</p> <table><tr><td>PROGRAM1-name</td><td><p>The name of the member used for IBM program product level 1 messages. Each IBM program product has its own set of names for related message members.</p><p>If 0 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.</p></td></tr><tr><td>PROGRAM2-name</td><td><p>The name of the member used for IBM program product level 2 messages. Each IBM program product has its own set of names for related message members.</p><p>If 0 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.</p></td></tr><tr><td>USER1-name</td><td><p>The name of the member used for level 1 and OCL statement messages for a program supplied by the user.</p><p>If 0 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.</p></td></tr><tr><td>USER2-name</td><td><p>The name of the member used for level 2 messages for a program supplied by the user.</p><p>If 0 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.</p></td></tr></table> <p><i>Note:</i> A level 2 message can be displayed only after the level 1 message of the same MIC (message identification code) has been issued.</p>	PROGRAM1-name	<p>The name of the member used for IBM program product level 1 messages. Each IBM program product has its own set of names for related message members.</p> <p>If 0 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.</p>	PROGRAM2-name	<p>The name of the member used for IBM program product level 2 messages. Each IBM program product has its own set of names for related message members.</p> <p>If 0 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.</p>	USER1-name	<p>The name of the member used for level 1 and OCL statement messages for a program supplied by the user.</p> <p>If 0 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.</p>	USER2-name	<p>The name of the member used for level 2 messages for a program supplied by the user.</p> <p>If 0 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.</p>
PROGRAM1-name	<p>The name of the member used for IBM program product level 1 messages. Each IBM program product has its own set of names for related message members.</p> <p>If 0 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.</p>								
PROGRAM2-name	<p>The name of the member used for IBM program product level 2 messages. Each IBM program product has its own set of names for related message members.</p> <p>If 0 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.</p>								
USER1-name	<p>The name of the member used for level 1 and OCL statement messages for a program supplied by the user.</p> <p>If 0 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.</p>								
USER2-name	<p>The name of the member used for level 2 messages for a program supplied by the user.</p> <p>If 0 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.</p>								

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 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, message 6666 comes from the message member named JOE. Message 7777 in procedure B comes from the message 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 prepare for inserting a diskette. A message telling the operator which diskette to insert usually precedes a PAUSE statement.

When the operator is ready, he can restart the SCP by taking the *continue* response to the pause message. 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**

<b>Function</b>	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.
<b>Placement</b>	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.
<b>Format</b>	// RUN
<b>Contents</b>	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 is a code that 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 code 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 of listing output, which can be listed on the printer or on the display screen, or specifies that output is 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, only the first 40 characters of each SYSLIST output line are displayed.

PRINTER	Print output on the printer.
---------	------------------------------

OFF	Do not list output.
-----	---------------------

**Example** The following is an example of assigning the printer as the SYSLIST device:  
// 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.

**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. However, 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 a data file entered from the keyboard.

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

**Format**                   /\*

*Note:* An end of data statement will not be 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).

**'message'**                A character string enclosed by single quotes is the actual message. Any character can be used in the character string except a single quote.

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.

Any combination of OCL statements and utility control statements can be contained in a procedure. One procedure may cause more than one job to be run. That is, a single procedure may contain more than one LOAD statement (see index entry: *// LOAD 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 interface to often-used system functions.

	ALTERBSC	DELETE	LOG	SAVE
	BACKUP	DISPLAY	ORGANIZE	SET
	CATALOG	FROMLIBR	OVERRIDE	STATUS
	COMPRESS	HISTORY	REBUILD	SYSLIST
	COPY11	INIT	RELOAD	TOLIBR
	CREATE	LINES	REMOVE	TRANSFER
	DATE	LISTLIBR	RESTORE	

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, 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 set of related OCL statements and utility control statements 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. To store a set of related OCL statements and utility control statements in the library as a procedure member you must use **\$MAINT**.



## EVOKING A PROCEDURE

Procedures can be evoked in three ways:

- By keying an INCLUDE OCL statement (command statement)
- By issuing a command key request
- 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.

### Command Key Request

The command key request is another way of evoking a procedure. By pressing the CMD key in response to READY and then an upper or lower case command key (the first command key is to the right of the CMD key, the second command key is to the right of the first, etc) you can request one of a possible twenty-four procedures (see *Creating Command Key Messages* 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 issuing a command key request, 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, 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 would evoke the PAYROLL procedure. If parameters are to be entered, they are keyed before ENTER is pressed.

### *Creating Command Key Messages*

If you wish to request a command statement not currently accessible via a command key, you must create a level 2 message load member named **##MSG3** using the CREATE command statement (see index entry: *CREATE procedure*) or the \$MGBLD utility (see index entry: *\$MGBLD utility program*), and then perform an initial program load (IPL).

### *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 (Lower Case)	MIC	Command Key (Upper Case)
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 COPY11 command statement as text, COPY11 is executed after the CMD key and the 0 data key are pressed.

## Evoking a Procedure from Another Procedure

A procedure requested by an INCLUDE OCL statement (command 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.

	PAYROLL Procedure	TAXES Procedure	DEDUCT Procedure
PAYROLL calls	// ...		
	// ...		
	TAXES calls	//...	
		//...	
		//...	
		DEDUCT calls	//...
			//...
			.
			.
			.

A procedure evoked by another procedure is called a *nested procedure*. In the preceding example TAXES and DEDUCT are nested procedures. Also, 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. 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 since it was the last parameter.

Some parameters have *defaults*. A default is a parameter which 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.

A maximum of 10 parameters can be passed when evoking a procedure. The 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:** You should not put sequence numbers on procedure statements.

### Modifying a Procedure Job Stream

Parameters, substitution, and conditional expressions allow you to modify a procedure job stream without changing OCL statements in the library.

### Substitution

Substitution allows you to omit certain information in OCL statements within procedures and allows this information to be passed by parameters when evoking the procedure. Information can also be passed in responses to prompts from OCL statements within the procedure. Defaults may be used when you do not supply the parameter.

There are six substitution formats that can be used within a statement contained in a procedure.

1. ?n?—where n is the number of the positional parameter on the statement that evokes the procedure. If no parameter is passed, the ?n? is removed. For example, suppose this OCL message statement were in a procedure:  

```
// * '3? HAS BEEN DELETED'
```

If no third parameter is passed, the statement would be:  
// \* 'HAS BEEN DELETED' (the ?3? has been removed)  
If the third parameter is FILEX, the statement would be:  
// \* 'FILEX HAS BEEN DELETED'

2. ?n'default'?—where n is the number of the positional parameter on the statement which evokes the procedure. If no parameter is passed, the 'default' value is substituted and treated as a positional parameter for all OCL statements in the procedure that require that parameter. For example:

```
// FILE NAME-?2'FILEIN'?
```

If no parameter is passed, the statement would be executed as  
// FILE NAME-FILEIN

FILEIN would then be treated as the second positional parameter for the rest of the procedure.

If FILEOUT were passed as the second positional parameter, the statement would be executed as

```
// FILE NAME-FILEOUT
```

and FILEOUT would become the second positional parameter for the rest of the procedure.

3. ?nT'default'?—same as ?n'default'? except T means temporary. That is, the default is substituted only for a particular statement. It is not treated as a positional parameter for any other statement in the procedure.

4. ?nR'msg-id'?—where n is the number of the positional parameter on the statement that evokes the procedure. R indicates that the parameter is required, and msg-id is a 4-digit decimal number identifying a message in the USER1 message member in the library. If no positional parameter is passed, the message corresponding to the number (msg-id) is displayed on the display screen for the operator. A reply must be entered and is substituted in the procedure statement for the substitution variable. For example:

```
// FILE NAME-?2R'6666'?
```

would cause the message corresponding to number 6666 in the USER1 message member to be displayed if the second positional parameter was not passed. Some value would then have to be entered from the keyboard in order for execution of the job stream to continue.

5. ?nR?—where n is the number of the positional parameter on the statement that evoked the procedure and R indicates that the parameter is required. If no parameter is passed, a reply must be entered on the keyboard. The reply is then substituted for the substitution variable. For example:

```
// FILE NAME-?2R?
```

with no second parameter entered would cause a message (ENTER MISSING PARAMETER) to be displayed to the operator. The SCP would then wait for a reply to be entered on the keyboard.

6. ?R?—no positional parameter is given on the statement that evokes the procedure. R indicates that a parameter is required. A reply must be entered on the keyboard. The reply is substituted for the substitution variable. For example:

```
// FILE NAME-?R?
```

would always require the operator to reply.

*Note:* In the preceding forms of substitution, the 'n' can be any number between 01 and 11 (the leading 0 can be dropped). The first ten parameters on the statement that evokes the procedure are positional. Only ten parameters can be passed; the eleventh must be supplied by default (for example, ?11'FILEIN?') or prompting (for example, ?11R?).

### **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 OCL statement 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:

<b>Keyword</b>	<b>Meaning</b>
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:

<b>Format</b>	<b>Meaning</b>
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) or a utility control statement. 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	Return to the previous procedure. If in the first level, then return to the keyboard for the next OCL statement.

*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 ?1?/REPORT IF ?2?/EOM MONTHLY  
    This expression says that if the first parameter on the statement that evokes the procedure is REPORT and the second parameter on that statement is EOM, then evoke the MONTHLY procedure. If the first parameter was not REPORT or the second was not EOM, the next statement or expression is read and MONTHLY is not executed.

### *ELSE Expression*

The ELSE expression is used in conjunction with the IF 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. For example:

```
// IF ?1?/ PAYROLL
// ELSE DELETE ?1?
// IF ?2?/ PAYROLL
// ELSE DELETE ?2?
```

In this example, if both parameters 1 and 2 were passed by the statement that evoked the procedure, the files specified by these parameters would be deleted by the DELETE procedure.

An IF expression can follow an ELSE expression in a conditional statement. For example:

```
// IF ?3?/ RETURN
// ELSE IF ?3?/YEAREND PAYROLL YEAREND
// ELSE PAYROLL WEEKLY
```

In the preceding example, if the third parameter is a blank, the PAYROLL procedure is not evoked. If the third parameter is YEAREND, YEAREND is passed to PAYROLL and the PAYROLL procedure is run. If the third 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.

### FILEBKUP Parameters

Filename-1	This is a required parameter. It is the name of the disk file to be backed up.
Vol-id	This is a required parameter. It is the volume identification of the diskette.
Filename-2 Filename-1	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 following procedural comment informs the operator of the system activity:

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

The following procedural statements, (2) through (6), will 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 will stop for key entry.

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

Parameter number 11 cannot be passed with a command statement; 10 is the maximum number of parameters, but parameter number 11 is available for this type of internal communication. The IF statement 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 entry key, the message ++ PARAMETER OMITTED-PROCEDURE CANCELED is displayed and statement (6) cancels the procedure.

The next procedural 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 coded prompt, 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 NAME'  
8. // IF ?1R?/ * '++ PARAMETER OMITTED-PROCEDURE CANCELED'  
9. // IF ?1?/ CANCEL  
10. // IF ?2?/ * 'ENTER THE DISKETTE VOLUME-ID'  
11. // IF ?2R?/ * '++ PARAMETER OMITTED-PROCEDURE CANCELED'  
12. // IF ?2?/ CANCEL
```

The next steps, (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, which follows, 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*, which follows Figure 4.

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

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

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

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

COMPRESS

COPY11 [ ALL ] ,,vol-id [,DELETE]

or

COPY11 filename, [ mddy ] [ ,vol-id ]  
 [ ddmy ] [ , ]  
 [ yymm ] [ , ]

CREATE sourcename [,REPLACE]

DATE { mddy }  
 { ddmy }  
 { yymm }

DELETE filename, [ F1 ] [ ,mddy ]  
 [ I1 ] [ ,ddmy ]  
 [ SCRATCH ] [ ,yymm ]  
 [ REMOVE ] [ , ]  
 [ ERASE ] [ , ]

DISPLAY filename [ ,mddy ]  
 [ ,ddmy ]  
 [ ,yymm ]

or

DISPLAY filename, [ mddy ] [ ,RECORD,value-1 [,value-2] ]  
 [ ddmy ] [ , ]  
 [ yymm ] [ , ]

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

FROMLIBR library-name-1, [SOURCE  
PROC  
LOAD  
SUBR  
LIBRARY], [filename-1  
library-name-1], [11  
F1], [ADD  
retention-days  
1], ,vol-id  
[P  
T  
S], [,blocks  
.8]  
or  
[ADD]

or

FROMLIBR {name-1,ALL  
ALL}, [SOURCE  
PROC  
LOAD  
SUBR  
LIBRARY], [filename-2  
name-1], [11  
F1], [ADD  
retention-days  
1], ,vol-id  
[P  
T  
S], [,blocks  
.8]  
or  
[ADD]

HISTORY [ALL] [,RESET]

INIT [vol-id  
system-date], [owner-id  
OWNER-ID], [RENAME  
DELETE  
FORMAT  
FORMAT2]

LINES [number  
66]

LISTLIBR DIR [SOURCE  
PROC  
LOAD  
SUBR  
LIBRARY]

or

LISTLIBR DIR,SYSTEM

or

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

LOG [CRT  
PRINTER], [EJECT  
NOEJECT]

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

ORGANIZE filename-1,  $\begin{bmatrix} \text{mmddy} \\ \text{ddmmy} \\ \text{yymmdd} \end{bmatrix}$ , F1,filename-2,  $\begin{bmatrix} \text{I} \\ \text{S} \\ \text{P} \end{bmatrix}$  [,position,char-literal]

or

ORGANIZE filename-1,  $\begin{bmatrix} \text{mmddy} \\ \text{ddmmy} \\ \text{yymmdd} \end{bmatrix}$ , [I1],vol-id,  $\begin{bmatrix} \text{retention-days} \\ \underline{1} \end{bmatrix}$  [,position,char-literal]

OVERRIDE [ADDR-nn]  $\left[ \text{,LINE-} \begin{Bmatrix} \text{C} \\ \text{P} \\ \text{R} \\ \text{S} \\ \text{T} \end{Bmatrix} \right]$

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

REBUILD

RELOAD [vol-id] ,  $\begin{bmatrix} \text{mmddy} \\ \text{ddmmy} \\ \text{yymmdd} \end{bmatrix}$   $\begin{bmatrix} \text{,filename} \\ \underline{\#LIBRARY} \end{bmatrix}$

REMOVE  $\left\{ \begin{array}{l} \text{library-name} \\ \text{name,ALL} \\ \text{ALL} \end{array} \right\}$   $\begin{bmatrix} \underline{\text{SOURCE}} \\ \text{,PROC} \\ \text{,LOAD} \\ \text{,SUBR} \\ \underline{\text{LIBRARY}} \end{bmatrix}$

RESTORE [ALL],  $\begin{bmatrix} \text{filename-1} \\ \underline{\#SAVE} \end{bmatrix}$   $\begin{bmatrix} \text{,mmddy} \\ \text{,ddmmy} \\ \text{,yymmdd} \end{bmatrix}$

or

RESTORE filename-2,  $\begin{bmatrix} \text{mmddy} \\ \text{ddmmy} \\ \text{yymmdd} \end{bmatrix}$   $\begin{bmatrix} \text{,RECORDS,value-1} \\ \text{,BLOCKS,value-2} \end{bmatrix}$

SAVE [ALL],  $\begin{bmatrix} \text{retention-days} \\ \underline{1} \end{bmatrix}$ ,  $\begin{bmatrix} \text{filename-1} \\ \underline{\#SAVE} \end{bmatrix}$ ,vol-id

SAVE filename-2,  $\begin{bmatrix} \text{retention-days} \\ \underline{1} \\ \text{ADD} \end{bmatrix}$ ,  $\begin{bmatrix} \text{mmddy} \\ \text{ddmmy} \\ \text{yymmdd} \end{bmatrix}$ ,vol-id

SET [value], [source-name],  $\begin{bmatrix} \text{MDY} \\ \text{DMY} \\ \text{YMD} \end{bmatrix}$   $\begin{bmatrix} \text{,mmddy} \\ \text{,ddmmy} \\ \text{,yymmdd} \end{bmatrix}$

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

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

STATUS

SYSLIST 

PRINTER
CRT
OFF

TOLIBR filename, 

F1
11

, 

mmddy
ddmmy
yymmdd

 [,REPLACE]

TRANSFER filename-1, 

11
----

, 

mmddy
ddmmy
yymmdd

, ADD, 

filename-2
filename-1

 [,date]

or

TRANSFER filename-1, 

11
----

, 

mmddy
ddmmy
yymmdd

, [NOADD], { value-1,value-2 } 

.RECORDS,value-3
.BLOCKS,value-4

or

TRANSFER filename-1,F1, 

mmddy
ddmmy
yymmdd

, vol-id 

.retention-days
1

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



This section describes all the IBM SCP procedures except the service procedures, which are described in Appendix D and three procedures in Part 5. The following information is given for each 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,

```
[
mmddy
ddmmy
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: mmddy, ddmmy, 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, [11] 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} ], 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. 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 is used to alter the following BSCA (binary synchronous communications adapter) environment items:

Item	Parameter
BPS (bits per second) rate	BRATE
Modem clocking	CLOCK
Debug facility	DEBUG
Error retry count	ERC
Standby line	SLINE
Modem test	TEST
Non-U.S.A. $\downarrow$	TONE

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

*Note:* The ALTERBSC procedure is intended for use only with telecommunications programming that uses the BSCA. For background information on binary synchronous communications, see *General Information – Binary Synchronous Communications, GA27-3004*.

### ALTERBSC Command Statement Format

$$\text{ALTERBSC } \left[ \text{BRATE-} \left\{ \begin{array}{c} \text{F} \\ \text{H} \end{array} \right\} \right] \left[ \text{,CLOCK-} \left\{ \begin{array}{c} \text{Y} \\ \text{N} \end{array} \right\} \right] \left[ \text{,DEBUG-} \left\{ \begin{array}{c} \text{Y} \\ \text{N} \end{array} \right\} \right] \left[ \text{,ERC-} \left\{ \begin{array}{c} \text{number} \\ \underline{\quad} \end{array} \right\} \right] \\ \left[ \text{,SLINE-} \left\{ \begin{array}{c} \text{Y} \\ \text{N} \end{array} \right\} \right] \left[ \text{,TEST-} \left\{ \begin{array}{c} \text{Y} \\ \text{N} \end{array} \right\} \right] \left[ \text{,TONE-} \left\{ \begin{array}{c} \text{Y} \\ \text{N} \end{array} \right\} \right]$$

*Note:* Though each particular 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 BSCA 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 N	Built-in debug facility is required, BSCA trace requested. Built-in debug facility is not required, BSCA trace not requested.
ERC-number <u>7</u>	Error retry count. The standard number of retries provided is seven (the default number); if more than seven are desired, enter a number up to 255. Valid numbers are 7 through 255.
SLINE-Y N	Switched line will be used as backup (standby) line for a point-to-point line. The normal line is to be used.
TEST-Y N	IBM data modem is being used. Automatic wrap test includes testing when a permanent error occurs unless the RPG II program specified a permanent error indicator for the BSCA file. Non-IBM data modem is being used. Automatic wrap test does not include modem testing.
TONE-Y N	Non-U.S.A. special TONE is required. Non-U.S.A. special TONE is not required.

## 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 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 ] [ ,filename ]  
                [ 1 ] [ #LIBRARY ]
```

## BACKUP Parameters

vol-id	Volume identification of the diskette(s). One to six <i>alphanumeric</i> (alphabetic or numeric) characters.
retention-days <u>1</u>	Length of the retention period (0 to 999 days) for the diskette file. The default is one day.  <i>Note:</i> 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).
<u>#LIBRARY</u>	#LIBRARY is the name assigned to the created diskette file.

## CATALOG PROCEDURE

A CATALOG request causes the disk or a diskette *VTOC (volume table of contents)* or a VTOC entry to be listed 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  $\left[ \begin{array}{l} \underline{ALL} \\ \text{filename} \end{array} \right] \left[ \begin{array}{l} ,I1 \\ ,\underline{F1} \end{array} \right]$

## CATALOG Parameters

<u>ALL</u>	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.
<u>F1</u>	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 will not be 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 if you want to determine the new locations of the files.

### COMPRESS Command Statement Format

COMPRESS

### COMPRESS Parameters

none

## COPY11 PROCEDURE

The COPY11 procedure causes all files on a single diskette or an individual file on a single diskette to be logically copied to another diskette. 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.

COPY11 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.

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.

COPY11 evokes the \$DUPRD utility (see index entry: *\$DUPRD utility program*).

### COPY11 Command Statement Format

Use	Format			
Copy all diskette files to another diskette	COPY11 [ALL] ,vol-id [,DELETE]			
Copy specific diskette file to another diskette	COPY11 filename, <table border="1"><tr><td>mmddyy</td></tr><tr><td>ddmmyy</td></tr><tr><td>yymmdd</td></tr></table> ,vol-id	mmddyy	ddmmyy	yymmdd
mmddyy				
ddmmyy				
yymmdd				

### **COPYI1 Parameters**

<b><u>ALL</u></b>	Indicates that all files on the diskette are to be copied to another diskette.
<b>filename</b>	Name of the single file to be copied to another diskette.
<b>mmdyy or ddmmyy or yymmdd</b>	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.)
<b>vol-id</b>	Volume label of output diskette; one to six alphanumeric characters.
<b>DELETE</b>	Any expired file will not be copied. (DELETE can be specified only when ALL is specified.)

### **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 with the same name

### CREATE Example

Assume a message source member contains the following statements:

```
USERMSG,1    (This is a control statement; USERMSG is to be the message
              load member name, and 1 is the message level.)

1234 ENTER YESTERDAY'S DATE.    (These are message text statements.
1235 ENTER TODAY'S DATE.       1234, 1235, and 1236 are MICs.
1236 ENTER TOMORROW'S DATE.    The message text follows the MICs.)

* THE ABOVE MESSAGES ARE FOR PROGX.    (This is a comment statement.)
```

If the above source member was named USERMI, the CREATE command statement would appear as follows:

```
CREATE USERMI
```

This would cause the MICs and their associated text to be formatted into a message load member named USERMSG. The comment statement (\* THE ABOVE MESSAGES ARE FOR PROGX) does not become part of USERMSG (message load member).

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

### DATE Command Statement Format

DATE { mmddyy }  
          { ddmmyy }  
          { yymmdd }

### DATE Parameters

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

*Note:* You must use the current system date format. You can use the STATUS procedure (see index entry: *STATUS procedure*) to see what the current format is.



## 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 \$DELETE utility (see index entry: *\$DELETE utility program*).

### DELETE Command Statement Format

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

### DELETE Parameters

filename	Name of the file to be deleted from the disk or a diskette. ALL cannot be used as a filename.
F1	The file to be deleted is on the disk.
<u>I1</u>	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.
<u>SCRATCH</u>	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.
mmdyy 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: <i>CATALOG procedure</i> and <i>STATUS procedure</i> ).

**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

In order 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 <table border="1"><tr><td>,mddy</td></tr><tr><td>,ddmmy</td></tr><tr><td>,ymmdd</td></tr></table>	,mddy	,ddmmy	,ymmdd
,mddy				
,ddmmy				
,ymmdd				
Display records by relative record number	DISPLAY filename, <table border="1"><tr><td>mddy</td></tr><tr><td>ddmmy</td></tr><tr><td>ymmdd</td></tr></table> ,RECORD,value-1 [ ,value-2 ]	mddy	ddmmy	ymmdd
mddy				
ddmmy				
ymmdd				

### DISPLAY Parameters

filename	Name of the file to be displayed or printed.
mddy   ddmmy   yymmdd	Creation date of file to be displayed or printed.
RECORD	The records from the file are to be displayed or printed based on their relative record number.
value-1	Starting record number to be displayed or printed. Valid for sequential, indexed, and direct files.
value-2	Last record number to be displayed or printed. Valid for sequential, indexed, and direct files. If value-2 is omitted, the display continues until end of file is reached.

### DISPLAY Example

- | In order to display or print the first one hundred records of the last 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-1, 

SOURCE
PROC
LOAD
SUBR
LIBRARY

, 

filename-1
library-name-1

,

I1
F1

ADD
retention-days
1

 ,vol-id

P
I
S

,blocks
.8

or

[ADD]

**Use**

Copies or adds all non-SCP members having names beginning with name-1.

**Format**

FROMLIBR { name-1,ALL } ,

SOURCE
PROC
LOAD
SUBR
LIBRARY

filename-2
name-1

[ I1 / F1 ] [ ADD retention-days 1 ] ,vol-id

[ P / T / S ] [ ,blocks 8 ]

or [ ADD ]

**FROMLIBR Parameters**

	library-name-1	Name of the non-SCP library member that is being copied from the library.
	name-1,ALL	All non-SCP members with names beginning with the indicated characters are to be copied. Up to seven characters may be used. Example: PAYR,ALL refers to non-SCP members having names that begin with PAYR.
	ALL	All designated non-SCP members are copied from the library.
	<u>SOURCE</u>	Source members are to be copied.
	PROC	Procedure members are to be copied.
	LOAD	Load members are to be copied.
	SUBR	Subroutine members are to be copied.
	LIBRARY	All types of members (SOURCE,PROC,LOAD and SUBR) are to be copied.
	filename-1	Name of the file that is created. If the filename is not specified, library-name-1 is assumed.
	filename-2	Name of the file that is created. If not specified, name-1 is assumed.

<u>I1</u>	Output file to be created on the diskette.
F1	Output file to be created on the disk.
retention-days <u>1</u>	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.  <i>Note:</i> A retention value of 999 makes a diskette file a permanent file. Retention cannot be specified if ADD is specified. ADD cannot be specified if retention is specified.
P	Permanent retention on disk.
<u>T</u>	Temporary retention on disk.
S	Scratch retention on disk.
ADD	Add library member(s) to an existing file that contains library members.  <i>Note:</i> 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. Retention cannot be specified if ADD is specified. ADD cannot be specified if retention is specified.
vol-id	Diskette volume label. One to six alphanumeric characters.
blocks <u>8</u>	Number of blocks to allocate for the output file. Ignored if ADD specified (see preceding description of ADD).

#### 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, noninterchange data 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 displaying 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's responses to messages and prompts. The entire file can be displayed (ALL parameter) or just the items previously displayed to the operator. Items previously displayed to the operator consist of items such as OCL statements and messages which were displayed—logged—as they were entered or issued (see index entry: *LOG procedure*).

| Any message issued when BSCA is active will not be recorded in the HISTORY file.

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

### HISTORY Command Statement Format

```
HISTORY [ALL] [,RESET]
```

### HISTORY Parameters

ALL        The entire contents of the HISTORY file will be displayed. This includes OCL statements in procedures. If ALL is *not* specified, only items previously displayed to the operator are displayed.

RESET     The HISTORY file will be cleared after it is displayed.

## 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 [vol-id  
      system-date] [owner-id  
                     OWNER-ID] [RENAME  
                                ,DELETE  
                                ,FORMAT  
                                ,FORMAT2]
```

### INIT Parameters

<u>vol-id</u> <u>system-date</u>	If specified, it consists of one to six alphanumeric characters. The vol-id will be 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.
<u>owner-id</u> <u>OWNER-ID</u>	Up to eight alphanumeric characters may be specified. These are placed (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 or FORMAT is specified, owner-id is written as OWNER-ID.
<u>RENAME</u>	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 all space on the diskette (initializes track 0 on the diskette).
FORMAT	Formats the entire surface of the diskette in the standard interchange format (see Appendix C). Tracks are renumbered for each track with a surface defect. If track 0 or more than 2 tracks have defects, the diskette is not initialized and no label of any kind is written.

*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 in 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 for a diskette 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 for Standard Data Interchange*, GA21-9182—see also Appendix C. However, diskettes formatted in 512-byte data sectors cannot be used for standard data interchange.)

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

*Note:* If FORMAT2 is specified for one diskette in a multivolume file, it must be specified for all diskettes in the file.

### INIT Examples

In order 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
```



## LINES PROCEDURE

The LINES procedure provides a means of modifying the printer lines per page variable. 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 is used to determine 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 your 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 <span style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">,SOURCE ,PROC ,LOAD ,SUBR ,LIBRARY</span>
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 } <span style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">,SOURCE ,PROC ,LOAD ,SUBR ,LIBRARY</span>

### 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	Library system directory. Valid with DIR only.
<u>SOURCE</u>	Source directory entries, if DIR is specified; otherwise, indicates source member(s).

<b>PROC</b>	Procedure directory entries, if DIR is specified; otherwise, indicates procedure member(s).
<b>LOAD</b>	Load directory entries, if DIR is specified; otherwise, indicates load member(s).
<b>SUBR</b>	Subroutine directory entries, if DIR is specified; otherwise, indicates subroutine member(s).
<b>LIBRARY</b>	All directory entry types (SYSTEM, SOURCE, PROC, LOAD, and SUBR), if DIR is specified; otherwise, indicates all member types (SOURCE, PROC, LOAD, and SUBR).

#### **LISTLIBR Examples**

In order 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 [CRT  
    PRINTER] [EJECT  
             ,NOEJECT]
```

### 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, deleting specified record types
- Copies a disk file to a diskette
- Copies a disk file to a diskette, deleting specified record types

If the input file is sequential, the output file is sequential. 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.

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, $\begin{bmatrix} \text{mmddyy} \\ \text{ddmmyy} \\ \text{yymmdd} \end{bmatrix}$ , F1, filename-2, $\begin{bmatrix} \text{I} \\ \text{S} \\ \text{P} \end{bmatrix}$ [,position,char-literal]
Reorganize a disk file as a diskette file.	ORGANIZE filename-1, $\begin{bmatrix} \text{mmddyy} \\ \text{ddmmyy} \\ \text{yymmdd} \end{bmatrix}$ , [I1], vol-id, $\begin{bmatrix} \text{retention-days} \\ \underline{1} \end{bmatrix}$ [,position,char-literal]

### ORGANIZE Parameters

filename-1	Name of the 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 of the name specified in filename-1 is the one that is reorganized.
F1	The disk will contain the organized copy.
I1	The diskette will contain the organized copy. Default is I1.
filename-2	Name of the disk file to contain the organized copy.
vol-id	Identifies the diskette by volume label. One to six alphanumeric characters. Valid only if I1 is specified.

<u>T</u>	Temporary retention on the disk.
S	Scratch retention on the disk.
P	Permanent retention on the disk.
retention-days <u>1</u>	Number of days (0 to 999) in the retention period for the diskette file. Default is 1.
	<i>Note:</i> A retention value of 999 makes a diskette file a permanent file.
position	Requests deletion of records having a certain character ( <i>char-literal</i> ) in the position specified. These records will not be copied to the reorganized file.
char-literal	Char (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 this character in the position specified by the position parameter are not copied to the reorganized file.

#### ORGANIZE Examples

In order to reorganize PAYROLL file into a permanent disk file called PAYR, you could enter:

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

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

*Note:* A date is not specified in either of the preceding two examples. Consequently, if more than one file named JOE or PAYROLL exist 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 the BSCA parameters specified in RPG source statements without recompiling the program:

Function	Parameter
Tributary Station Address	ADDR
Line Type	LINE

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

*Note:* The **OVERRIDE** procedure is intended for use only with telecommunications programming that uses the BSCA. For background information on binary synchronous communications, see *General Information – Binary Synchronous Communications, GA27-3004*.

### OVERRIDE Command Statement Format

OVERRIDE [ADDR-*nn*] [ ,LINE- $\left. \begin{array}{c} \text{C} \\ \text{P} \\ \text{R} \\ \text{S} \\ \text{T} \end{array} \right\}$  ]

#### 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 RPG specifications, reenter a valid **OVERRIDE** command omitting the ADDR parameter. The addressing characters will default to the RPG specifications.

### OVERRIDE Parameters

ADDR- <i>nn</i>	Hex 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 RPG specifications.
LINE-C	CDSTL (connect data set to line) switched line (World Trade only)
P	Point-to-point leased line.
R	Line type specified in RPG source statements.
S	Point-to-point switched line.
T	Tributary station line on multipoint.

## **REBUILD PROCEDURE**

The REBUILD procedure allows you to restore certain 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 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



## RELOAD PROCEDURE

The RELOAD procedure initiates an IPL from a diskette file containing the system library. That is, RELOAD initiates an IPL from a diskette file produced as output from the BACKUP procedure (see index entry: *BACKUP procedure*).

Space allocations for the library file (#LIBRARY) and the *library directory* within the library file are displayed on the display screen during execution of RELOAD. The library directory contains an entry for each member in the library. Each entry describes and identifies the location of the corresponding library member.

The allocations can be altered at the time they are displayed. Unaltered allocations remain what they were when the diskette file was created by the BACKUP procedure.

At the time that space allocations are displayed you can also change the inquiry program support and offline multivolume file support option. Inquiry programs are described with the \$LOAD utility program—see index entry: *\$LOAD utility program*. See index entry: *offline multivolume file* for a description of how offline multivolume files are processed. If the INQUIRY/OFFLINE option is not changed at the time it is displayed, it remains at the value in effect when the diskette file was created by BACKUP.

The RELOAD procedure evokes the \$LOAD utility.

*Note:* The \$LOAD utility provides the only method whereby you can change the size of the library directory (alter the space allocated to it). When using \$LOAD (RELOAD), 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 library members on the disk that you want to save, 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 may 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 entry: *FROMLIBR procedure* and *TOLIBR procedure*.

### RELOAD Command Statement Format

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

### RELOAD Parameters

vol-id	Volume identification of the first (or only) diskette. One to six alphanumeric characters. Vol-id becomes the disk volume identification.
mmddyy ddmmyy yymmdd	Creation date of the diskette file.
filename <u>#LIBRARY</u>	Name of the diskette file containing the system library. Default is #LIBRARY.

## REMOVE PROCEDURE

The REMOVE procedure causes the named library member(s), unless they are SCP members, to be deleted. The space that was occupied by members deleted by the REMOVE procedure can be used for new members, provided there are no active members physically located after the deleted ones in the library.

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

### REMOVE Command Statement Format

REMOVE	{	library-name	}	[	<u>SOURCE</u>	]
		name,ALL			,PROC	
		ALL			,LOAD	
					,SUBR	
					,LIBRARY	

### REMOVE Parameters

library-name	Full 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.
<u>SOURCE</u>	Remove non-SCP source statement 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

An example of a library REMOVE is:

```
REMOVE JOE,PROC
```

This deletes the procedure non-SCP library member named JOE from the library.

```
REMOVE SAM,LIBRARY
```

deletes the non-SCP members in the library that are named SAM.

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

deletes all non-SCP members in the library beginning with the letters PAY.

## RESTORE PROCEDURE

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

- the ORGANIZE procedure (see index entry: *ORGANIZE procedure*),
- the SAVE procedure (see index entry: *SAVE procedure*), or
- 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.

A RESTORE request causes a file to be reconstructed 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.

Insert diskette messages 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*).

### RESTORE Command Statement Format

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

### RESTORE Parameters

<u>ALL</u>	All data files previously saved are to be restored to the disk.
filename-1 <u>#SAVE</u>	Name associated with the entire set of files previously saved on the diskette by the SAVE (SAVE ALL) procedure. #SAVE is the default value.

filename-2	Name of the single diskette file that is to be restored to the disk.
mmddy ddmmy 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.

### **RESTORE Examples**

In order to restore all files previously saved by a SAVE procedure, you would enter:

**RESTORE**

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

**RESTORE JOE,,RECORDS,300**

In the above 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 file. Sequential, indexed, and direct disk files can also be added to files saved previously and can reside as single volume or multivolume files. Appropriate insert diskette messages 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 [ <u>ALL</u> ], [ <u>1</u> retention-days ], [ filename-1 #SAVE ],vol-id
Save one disk file on diskette, or add a disk file to a file saved previously	SAVE filename-2, [ <u>1</u> retention-days ], [ mmdyy ddmmyy yymmdd ],vol-id ADD

### SAVE Parameters

<u>ALL</u>	Requests that all data files on the disk be copied to diskette.
filename-2	Name of one file on the disk to be saved. The diskette file will have the same name.
retention-days <u>1</u>	Number of days (0 to 999) the diskette file is to be retained. Default is 1.  <i>Note:</i> 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 <u>#SAVE</u>	Name associated with the entire set of saved files. #SAVE is the default value.
mmdyy 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**

In order 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**

In order 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 is used to specify 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 will remain 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 | ,mddyy  |
| DMY | ,ddmmyy |
| YMD | ,yymmdd |


```

*Note:* Though each particular 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.
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: <i>IMAGE statement</i> ).
	<i>Note:</i> Belt 48 and belt 64 are the two IBM-supplied procedures if you are using standard characters.
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, MDY format.
ddmmyy	Specifies the system date, DMY format.
yymmdd	Specifies the system date, YMD format.

## STATUS PROCEDURE

The STATUS procedure causes the system status information to be displayed on the display screen (and printed on the printer if the printer is assigned for logging—see index entry: *LOG procedure*). The first six lines displayed (four lines of status information and two lines of messages) include:

- System date—see index entry: *DATE procedure* for information about setting the date.
- Date format—see index entry: *SET procedure* for information about setting the date format.
- SYSLIST assignment—see index entry: *SYSLIST procedure* for information about changing the assignment.
- Number of lines printed per page—see index entries: *LINE procedure* and *SET procedure* for information about specifying the number of lines to be printed per page.
- SWITCH indicators—see index entry: *//SWITCH statement* for information about setting and changing the indicators.
- Print belt image format, permanent or temporary. The permanent image is set at IPL. A temporary image is any image that has replaced the image established at IPL—see index entry: *//IMAGE statement* for information about changing the print belt image.
- Print belt length in number of characters—see index entry: *SET procedure* for information about the print belt.

After the first six lines of status information are displayed, you have four options:

- 0 option—next page (display more status information)
- 1 option—display the print belt image
- 2 option—end STATUS
- 3 option—cancel the job

After the initial six lines of information are displayed, the 0 option causes the release level of the system, inquiry/offline multivolume availability, and disk capacity to be displayed. If the 0 option is chosen after this information is displayed, the initial six lines are displayed again. This cycle is continued each time the 0 option is chosen. If your system has BSC (binary synchronous communications) support, BSC status information is displayed if you continue to choose the 0 option. The first display of BSC information shows transmission rate, clocking, modem, line, answer tone, terminal number, debug, and error retry count. The second BSC display shows line type, line code, and tributary address. (For more information on changing variables on BSC, see index entries: *ALTERBSC procedure* and *OVERRIDE procedure*.) After information is displayed by the 0 option, STATUS again gives you the four options listed.

The 1 option displays the print belt image (see index entry: *SET procedure* for information about changing the image). After the print belt image is displayed, STATUS again gives you the four options listed.



If you choose the 2 option, the STATUS procedure is terminated and the job stream continues. If STATUS was not part of a job stream, the keyboard is put in the ready status.

If you choose the 3 option, the job stream is terminated and the keyboard is put in the ready status.

The STATUS procedure evokes the \$STATS utility (see index entry: *\$STATS utility program*).

#### **STATUS Command Statement Format**

STATUS

#### **STATUS Parameters**

None

## **SYSLIST PROCEDURE**

The SYSLIST procedure can be used to change the method of listing output. The possible changes allowed are:

- Subsequent SYSLIST output going to the printer
- Subsequent SYSLIST output going to the display screen
- Subsequent SYSLIST output requests being ignored

### **SYSLIST Command Statement Format**

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

### **SYSLIST Parameters**

<u>PRINTER</u>	Selects the printer for SYSLIST output
CRT	Selects the display screen for SYSLIST output
OFF	Suppresses SYSLIST output

## TOLIBR PROCEDURE

The TOLIBR procedure copies into the library 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 record and end with a CEND record. The format of the COPY record, 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 records 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 the standard interchange 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.

The TOLIBR procedure evokes the \$MAINT utility.

### TOLIBR Command Statement Format

```
TOLIBR filename, [ F1 ] , [ mmdyy
                  [ I1 ] , [ ddmyy
                        yymmdd ] [REPLACE]
```

### TOLIBR Parameters

filename	Name of the file containing the member(s) to be placed in the library.
F1	The file is on the disk.
<u>I1</u>	The file is on a diskette.
mmdyy ddmyy 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 it 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 duplicate member replaces the existing member in the library. If more duplicates are found during the job, they automatically replace existing members, and no messages are displayed regarding the duplicates.

If **REPLACE** is specified, all duplicates are placed in the library, and no messages regarding them are displayed.

## TRANSFER PROCEDURE

The TRANSFER procedure moves files between the disk and diskettes on which data is recorded in the standard interchange format. (See Appendix C for information on the standard interchange format.) TRANSFER can:

- Add a diskette file that is in the standard interchange format to an existing sequential disk file.
- Convert a standard interchange diskette file to a disk sequential or indexed file.
- Convert a disk file to a standard interchange diskette file. (Standard interchange files are sequential files.)

*Note:* Because TRANSFER only moves files between the disk and standard interchange diskettes, TRANSFER cannot be used to move files between the disk and diskettes on which data is recorded in 512-byte sectors. Data is recorded in 512-byte sectors on a diskette if, when the diskette was initialized, FORMAT2 was specified in the INIT command statement—or—OPTION-FORMAT2 was specified in the UIN utility control statement. See index entries: *\$INIT utility program* and *INIT procedure*.

When a standard interchange diskette file is added to an existing sequential disk file, the record length of the disk file is used for all records added to the file. When a standard interchange diskette file is converted to a sequential or indexed disk 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 standard interchange diskette—always sequential—can be a sequential, indexed, or direct file. If the record length of the disk file is greater than 128, all records are truncated to 128.

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

### TRANSFER Command Statement Format

Use	Format
Transfer file from I1 to an existing F1 file	TRANSFER filename-1, [I1], [mmddy ddmmy yymmdd], ADD, [filename-2 filename-1], [date]
Transfer a file from I1 to a new F1 file	TRANSFER filename-1, [I1], [mmddy ddmmy yymmdd], [NOADD], [value-1,value-2] [RECORDS,value-3 BLOCKS,value-4]
Transfer a file from F1 to I1	TRANSFER filename-1,F1, [mmddy ddmmy 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 for filename-1.
<u>I1</u>	A standard interchange diskette file is being transferred to a sequential or indexed disk file.
F1	A fixed disk file is being transferred to a standard interchange diskette file.
mmddy ddmmy yymmdd	Creation date of the file being transferred.
ADD	Records in a standard interchange diskette file are to be added to the records in an existing disk sequential file; the first record from the diskette file is to be placed after the last record existing in the disk file.
filename-2	Identifies the existing disk file to which a standard interchange diskette file is to be added. Filename-2 is valid only if ADD is specified. If 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: mmddy, ddmmy, or yymmdd.
<u>NOADD</u>	The standard interchange diskette file being transferred will not be added to an existing disk file, but will become a new file with filename-1 as the filename. NOADD is assumed whenever a file is transferred from I1 to F1.
value-1	Key length for an indexed disk 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 relative displacement of 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 and value-3	Specifies that the disk file being created be large enough to contain the number of records specified by value-3.  <i>Note:</i> 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 and value-4	Specifies that the disk file being created be large enough to contain the number of blocks specified by value-4.  <i>Note:</i> 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 standard interchange diskette file. One to six alphanumeric characters.
retention-days <u>1</u>	Number of days (0 to 999) the created standard interchange diskette file is to be retained. Default is 1.  <i>Note:</i> A retention value of 999 makes a diskette file a permanent file.

### TRANSFER Examples

In order to add a diskette standard interchange file named JOE to an existing disk file named JOE, you could enter:

TRANSFER JOE,,,ADD

In order to create a sequential disk file named JIM from diskette standard interchange file named JIM, you could enter:

TRANSFER JIM

In order to create a diskette standard interchange 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

You can use the CATALOG procedure (see index entry: *CATALOG procedure*) 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 in one area.

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 freeing up disk space 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 will not be needed in the next job. Then, to free up space for new files, use the DELETE procedure to delete the original versions of the copied 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 COPY11 procedure (see index entry: *COPY11 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 be used to identify the file to the program creating the file. The LABEL parameter can be used to assign a name for 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 is used to identify the file on the disk. Names that are related to file contents and to applications using the files should be assigned so that files can easily be identified 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 for the file 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 helps protect a file from being deleted accidentally. Temporary files (RETAIN-T) are usually used more than once, but you can delete a temporary file at any time by changing its classification to RETAIN-S, which identifies the file as a scratch file. Scratch files are deleted when the job in which they are created ends.

*Note:* Two disk work files, \$SOURCE and \$WORK, are needed to load and run a program that requires source input (bit 4 of the first attribute byte in the load member's library directory entry is on—see index entry: *library directory entry* for a description of the information contained in library directory entries). You can create the two files by entering two FILE OCL statements in the program's job stream. If you do not enter FILE statements to create \$SOURCE and \$WORK for a program that requires the two work files, the IBM System/32 SCP will create the files automatically as scratch files (RETAIN-S) with a file size of 24 blocks. (The space must be available.)

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 index entry: *offline multivolume file*.

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 the 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 will request another diskette to continue the file. (For more information, see index entry: *multivolume file*).
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*.)

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 listings created by the CATALOG procedure, to help identify which files are on which diskettes.

## 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 room for. 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 had been filled. A solution would be to use the DELETE and COMPRESS procedures. (See index entries: *COMPRESS procedure* and *DELETE procedure* to free up disk space.)

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 has been inserted, the system copies the records from the disk extent to the diskette. The disk extent will then be 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 dates (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 command 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.

*Note:* To process an offline multivolume file (for input, update, or add), you must allocate a disk space equal to the segment size defined (by the BLOCKS parameter) when the offline multivolume file was created. You may find it helpful, therefore, to keep a record of the segment size of each offline multivolume file you create to remind you of the segment sizes. The CATALOG ALL, I1 procedure will display the number of blocks (OLMV block size) if you do not know it.

## 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 (standard 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 the standard or extended formats, you cannot create an offline multivolume file using these formats interchangeably. Either format can be used to create an offline MVF, but all diskettes for a file must have the same format.

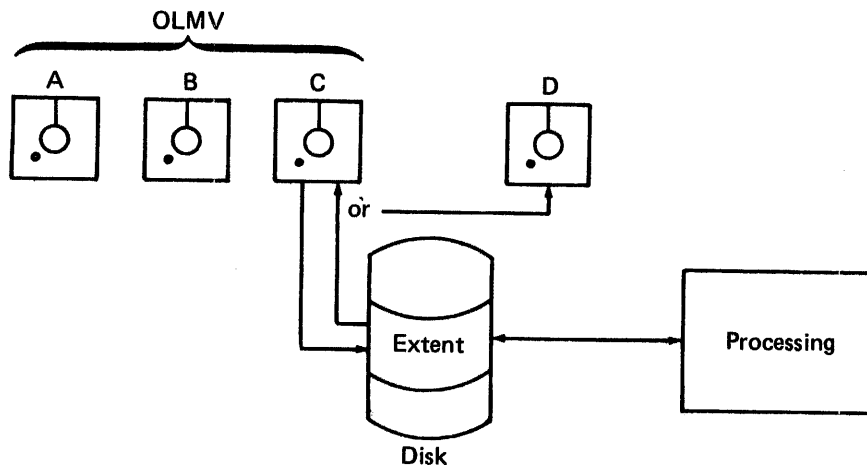
- To process an offline multivolume file after it is created, you must allocate a disk extent at least equal in size to the extent defined when the file was created. If you do not remember or do not have a record of the number of blocks allocated originally, you can run the CATALOG ALL, I1 procedure (see index entry: *CATALOG procedure*) using the offline multivolume diskette. The disk extent is indicated in the column titled OLMV BLOCK SIZE 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 file support cannot be used on a system having BSC (binary synchronous communications) 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 multivolume files) have been initialized before you begin the job. You can use the INIT procedure (see index entry: *INIT procedure*) to initialize the diskettes.
- 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 has been inserted, the system transfers the records to the disk extent; the system 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.

You can write the file records back onto diskette C, or you can scratch diskette C and write the file records on diskette D. Diskette D now contains the records from diskette C plus the records just added:





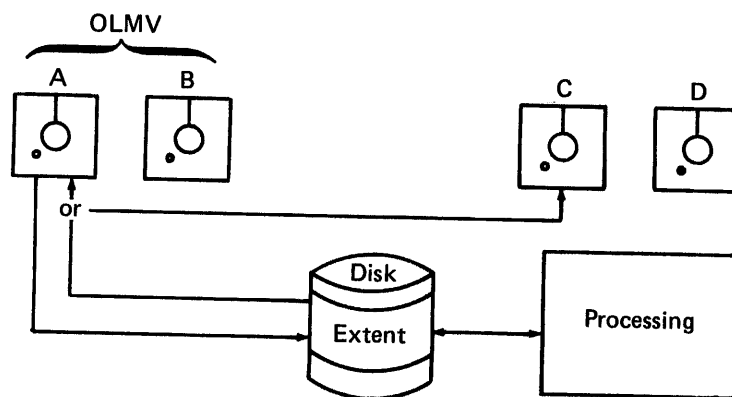
- Except for adding records to an offline multivolume file (which involves only the last segment of the file), the segments of an offline multivolume file must be processed sequentially beginning with the first segment.

### Considerations

- When an offline multivolume file is created, each segment, except possibly the last, contains the maximum number of complete records that can fit into the disk extent allocated for a file segment.
- When updating the contents of an offline multivolume file, only one diskette can be updated at a time. Suppose, for example, you have two diskettes to be updated: diskettes A and B.

To update diskette A, the system displays the message: CONTINUE WHEN PROPER DISKETTE INSERTED for the input operation. The same message is displayed again for the output operation, after the diskette has been updated.

You can then write the update back onto diskette A, or you can write the update onto diskette C. You will now have two copies, one of the old version and one of the updated version:



Repeat the same procedure for updating diskette B, using diskette D if you want to retain the previous version of diskette B.

- When additions are made to an offline multivolume file, the current job date becomes the creation date of all new file segments plus the last one of the old file segments.
- For the operator's convenience, indicate on the envelope of each diskette in an offline multivolume file the volume number sequence of the diskette. For example, the first diskette in a file would be volume #1, the second would be volume #2, and so on.
- Whenever an offline multivolume file is updated, the current job date becomes the creation date of the file segments that are updated.
- When writing file statements, be sure to write the file name in the I1 file statement correctly. If you do not, and still have the old extent on the disk, you will process erroneous data.



**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 fixed 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 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.

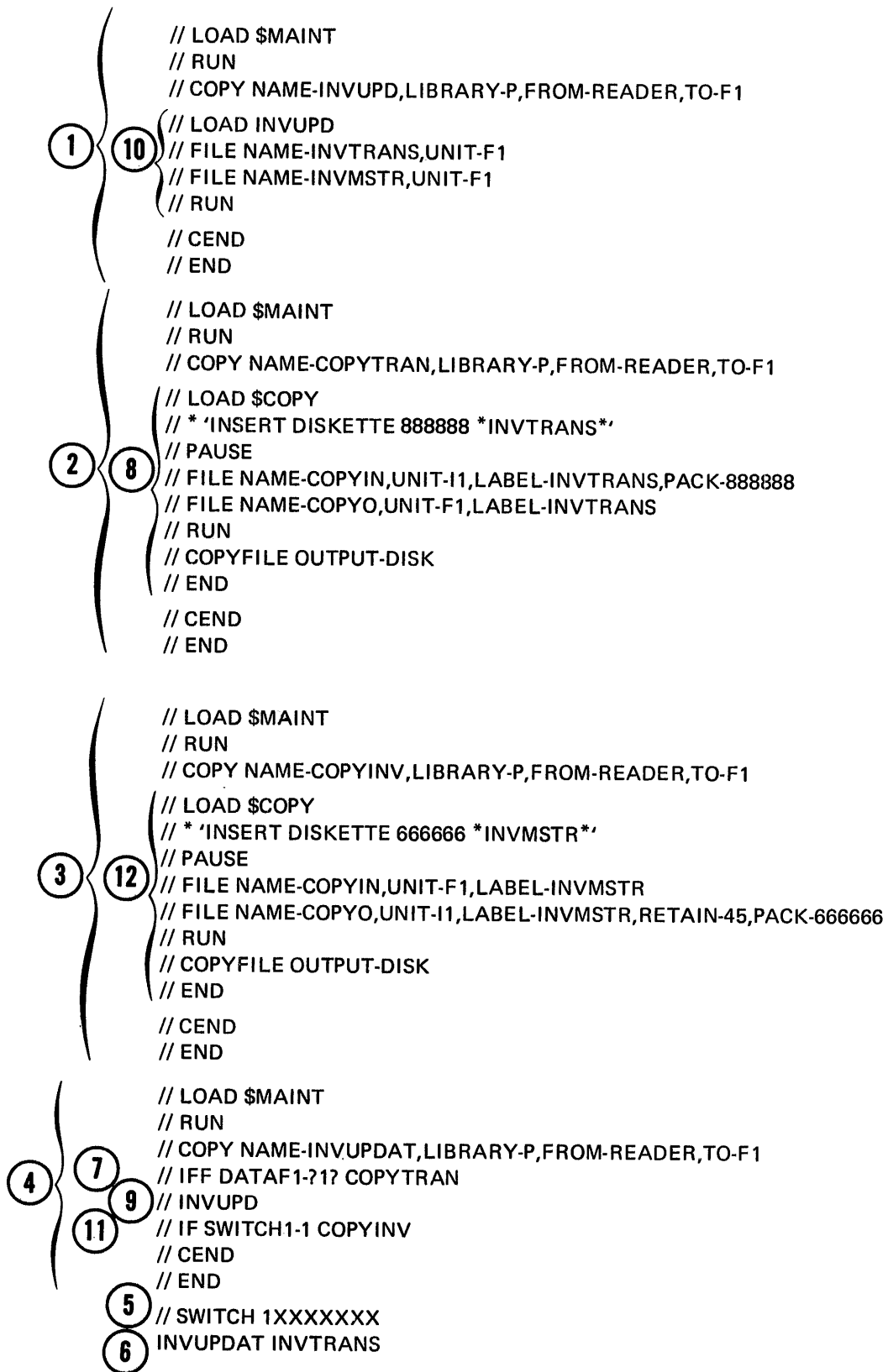


Figure 5. OCL and Command Statement Example

1. The procedure INVUPD (10) is cataloged in the library as a procedure member.
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 below) 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).

Underlining identifies default values. A default value is a value that is automatically substituted for an optional parameter that is omitted. For example,  $\left[ \frac{\text{YES}}{\text{NO}} \right]$

means that if neither YES or NO is specified, YES is used.

Braces ( { } ) indicate that you must choose one of the values enclosed by the braces.

For example, in the expression  $\left[ \text{PARAM-} \begin{Bmatrix} \text{A} \\ \text{B} \end{Bmatrix} \right]$ , the braces indicate that if you

choose to enter the parameter, you must specify either A or B.



**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.

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 PROCEDURE 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 PROCEDURE 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:

```
// COPY11 NAME-JOE,PACK-123456
```

The statement identifier is COPY11. The parameter keywords are NAME and PACK. The information supplied by the parameters is JOE and 123456.

*Notes:*

1. An end control statement (// END) must be the last control statement entered for a utility if control statements are used. The end control statement indicates the end of the control statements for a utility.
2. Do not put sequence numbers on utility control statements.

### CONVENTIONS FOR DESCRIBING UTILITY CONTROL STATEMENT FORMATS

In the descriptions of utility control statement formats given in this part of the manual, capitalized words and letters, numbers, special characters, brackets, and braces have special meanings. Capitalized expressions must be entered as they appear in the formats. When numbers or nonalphabetic characters appear in a capitalized expression, they 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. However, if the noncapitalized expression is shown enclosed in single quotes ('), the quotes must be entered if a value is entered.

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

```
[OUTPUT-PRINT  
OUTPTX-PRINT]
```

means that neither expression has to be entered (they are optional), and that only one of them can be entered for one utility control statement.

Underlining identifies default values. A default value is a value that is automatically substituted for an optional parameter that is omitted. For example,

```
YES  
NO
```

 means that if neither YES nor NO is specified, YES 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.

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 inquiry mode is interrupted by inquiry, 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 program or erroneous results when the interrupted program regains control.

The system and disk oriented functions listed below have the potential for such abnormal termination and erroneous results when executed in an inquiry mode:

Utilities — all functions of which are always prohibited:

Utility	Function(s)
\$BACK	(Back up library)
\$LOAD	(Reload library)
\$PACK	(Compress file space)
\$REBLD	(Rebuild VTOC)
\$SETCF	(Reconfigure system)
\$BUILD	(Alternate sector)

Utilities — some functions of which are prohibited:

Utility	Function(s)
\$COPY	(Restore all/save all)
\$DELET	(Delete all)

Utilities — processing of active files prohibited:

Utility	Function(s)
\$BICR	(TRANSFER active file)
\$COPY	(SAVE/ORGANIZE active file)
\$DELET	(DELETE active file)

Utilities — processing permitted with warning message:

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 reconstruct on the disk a library that has been backed up on—copied to—diskettes, you can use the RELOAD procedure (see index entry: *RELOAD procedure*) or perform an IPL from the diskettes 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*).

*Note:* 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*.

### **\$BACK Utility Control Statement Format**

Utility control statements are not used.

### **\$BACK OCL Sequence**

```
// LOAD $BACK
// FILE NAME-#LIBRARY,UNIT-I1,...
// RUN
```

## \$BICR—STANDARD INTERCHANGE UTILITY PROGRAM

This utility provides a means of converting a disk file to a standard interchange file on diskette, of converting a diskette standard interchange file to a sequential or indexed disk file, and of adding a standard interchange file to a sequential disk file. All diskette files that are input for \$BICR must be in the standard interchange format (see Appendix C); all diskette files created by \$BICR are in the standard interchange format.

In adding a standard interchange 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 standard interchange diskette file, the record length of the disk file is set to that of the diskette file. In creating a new standard interchange 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 (see index entry: *TRANSFER procedure*).

### \$BICR Utility Control Statement Formats

Use	Control Statements
To create a diskette standard interchange file from a disk file or convert a diskette standard interchange file to a disk sequential file	[// TRANSFER] // END
To add the data in a standard interchange diskette file to a disk sequential file	// TRANSFER ADD-YES // END
To create an indexed file on the disk from a diskette standard interchange file	// TRANSFER ADD-NO,KEYLEN-value,KEYLOC-value // END

### \$BICR Parameters

ADD-YES	Specifies that when converting a standard interchange diskette file to a disk file, the records in the diskette file are to be added to an existing disk sequential 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 available in the disk file.

If ADD-YES is not specified, ADD-NO is assumed.

- ADD-NO** Indicates that when copying a diskette standard interchange 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 relative displacement of 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- $\left\{ \begin{array}{l} I1 \\ F1 \end{array} \right\}$ ,LABEL-from-filename,...
[// FILE NAME-COPYO,UNIT- $\left\{ \begin{array}{l} F1 \\ I1 \end{array} \right\}$ ,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 standard interchange 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 has occurred.

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 or unusable disk sectors. The \$BUILD utility program searches the alternate sectors of the disk for data that is 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 character and hex 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 hex 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.



### **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, this 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 hex 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-03740 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, after keying corrections, you press the ENTER key, 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 on a diskette, advance the cursor into 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
```

## **\$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 ordering 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; you have the option of deleting selected records if the entire file is displayed.

\$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', 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 reorganization.

**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.

*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 hex 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 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 what 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).

**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.

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      This parameter 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-  $\left. \begin{array}{l} \{F1\} \\ \{I1\} \end{array} \right\}$       Specifies that the disk (F1) or a set of diskettes (I1) is to contain the copy.

### *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.

The parameter OUTPUT-DISK means the file is to be copied; OUTPUT-PRINT means it is to be displayed and OUTPTX-PRINT means the file is to be displayed in hex values.

*Copying a File:* \$COPY can copy a file from one disk to another or from one area on the disk to another on the disk. Data files copied to and from diskette are not standard interchange 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 and file organization are retained.

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:** Records from the 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.

The 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). Characters that have no graphic display symbol are displayed as 2-digit hex numbers in over-and-under format.

The following is an example of the way the program displays hex numbers:

```
ABCDEF J12345
      B
      6
```

The hex number B6 represents a character that has no graphic symbol.

After displaying the last record, the program prints a message stating the number of records displayed.

#### ***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,XE7' and DELETE-'100,X' would yield the same results.

#### ***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 apply to copying or displaying part of an indexed file. The SELECT PKY parameter applies to an indexed file that contains packed keys. Related parameters are FROM and TO.

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, A1990, and A1955. The parameters FROM-'A10' and TO-'A15' refer to record keys A1000, A1010, A1040, and A1500.

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 } ] ,LABEL-filename
// FILE NAME-COPYO [ ,UNIT- { F1 } ] ,LABEL-filename,
```

For F1:

```
{ RECORDS-number } [ ,RETAIN- { T } ]
{ BLOCKS-number } [ ,RETAIN- { P } ]
[ ,RETAIN- { S } ]
```

For I1:

```
[ RETAIN- { retention-days } ] ,PACK-vol-id
[ 1 ]
```

```
// RUN
// COPYALL...
or
// COPYADD
or
// COPYFILE
[ // SELECT... ]
[ // KEY... ]
// END
```

Statement Entry	Meaning
<code>// LOAD</code>	
<code>\$COPY</code>	Name of disk copy/display program.
<code>// FILE</code>	
<code>NAME-COPYIN</code>	Name disk copy/display program uses to refer to the file to be copied, reorganized, or displayed.
<code>UNIT- { F1 / I1 }</code>	Identifies either the disk (F1) or a diskette (I1) as containing the file to be copied.
<code>LABEL-filename</code>	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.
<code>// FILE</code>	
<code>NAME-COPYO</code>	Name disk copy/display program uses to refer to output file being created. (This OCL statement is not needed for displaying a file.)
<code>UNIT- { F1 / I1 }</code>	Specifies location of output file: disk (F1) or diskette (I1).
<code>LABEL-filename</code>	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.
<code>{ RECORDS-number / BLOCKS-number }</code>	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.
<code>RETAIN- { T / P / S }</code>	Retention designation of the disk output file: T is temporary, P is permanent, S is scratch.
or <code>[ retention-days / 1 ]</code>	Retention designation of diskette output file expressed in number of days. Default is one day.
<code>PACK-vol-id</code>	The diskette volume label. Meaningful only if the unit designation is I1.

## **\$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
```

## \$DELETE—FILE DELETE UTILITY PROGRAM

The \$DELETE 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 \$DELETE program. The system file #LIBRARY cannot be deleted.

\$DELETE is evoked by the DELETE procedure (see index entry: *DELETE procedure*).

### \$DELETE Utility Control Statement Formats

Use	Control Statements
Scratch the VTOC entry for the named file	// SCRATCH UNIT- $\left\{ \begin{matrix} F1 \\ I1 \end{matrix} \right\}$ , LABEL-filename [,PACK-vol-id] // END
Scratch the VTOC entry for the named file identified by the specified creation date	// SCRATCH UNIT- $\left\{ \begin{matrix} F1 \\ I1 \end{matrix} \right\}$ , LABEL-filename, DATE- $\left\{ \begin{matrix} mmddyy \\ ddmmyy \\ yymmdd \end{matrix} \right\}$ [,PACK-vol-id] // END
Remove the VTOC entry of the named file	// REMOVE UNIT- $\left\{ \begin{matrix} F1 \\ I1 \end{matrix} \right\}$ , LABEL-filename [,PACK-vol-id] // END
Remove the VTOC entry for the named file identified by the specified creation date	// REMOVE UNIT- $\left\{ \begin{matrix} F1 \\ I1 \end{matrix} \right\}$ , LABEL-filename, DATE- $\left\{ \begin{matrix} mmddyy \\ ddmmyy \\ yymmdd \end{matrix} \right\}$ [,PACK-vol-id] // END
Remove the VTOC entries for all the files on the specified disk	// REMOVE UNIT- $\left\{ \begin{matrix} F1 \\ I1 \end{matrix} \right\}$ , LABEL-ALL $\left[ \begin{matrix} \text{,PACK-vol-id} \\ \text{required if} \\ \text{UNIT-I1} \end{matrix} \right]$ // END

Use	Control Statements
Remove the VTOC entry for the named file and erase the contents of the file	<pre>// REMOVE UNIT- {F1}                   {I1} ,LABEL-filename,DATA-YES [PACK-vol-id] // END</pre>

### **\$DELET Parameters**

UNIT- {F1} {I1}	Specifies the location of the volume containing the file(s) being deleted. F1 specifies the disk, I1 specifies diskettes.
LABEL-filename	Identifies by name the file to be deleted. If the file is a multi-volume file on diskette, the entire logical file will be deleted.
LABEL-ALL	Specifies deletion of all files on the specified volume. For diskettes, the files on more than one diskette can be deleted with one \$DELET request if the diskettes have the same vol-id.
DATE- {mmddyy} {ddmmyy} {yymmdd}	Identifies a file by its creation date.
DATA-YES	Specifies that the contents of a file be erased.
DATA-NO	Specifies that the contents of a file not be erased. If the DATA parameter is not specified, DATA-NO is the default.
PACK-vol-id	Identifies diskette by vol-id. Required only when both LABEL-ALL and UNIT-I1 are specified in the REMOVE statement.

### **\$DELET Parameter Summary**

#### *UNIT Parameter*

The UNIT parameter is either UNIT-F1 (disk) or UNIT-I1 (diskette). It tells the program the location of the volume containing the file(s) being deleted.

#### *LABEL Parameter*

Specifies the name of the file to be deleted. For disk requests, LABEL-ALL specifies that all files on the disk except #LIBRARY be deleted. For diskette requests, LABEL-ALL specifies that all files on the inserted diskette be deleted. The files on more than one diskette can be deleted by a single LABEL-ALL request if the diskettes have the same vol-id.

### *DATE Parameter*

Each file can be further identified by its creation date. For the disk, the optional DATE parameter can also be used to distinguish a particular file when there is more than one file with the same name. If date is specified, then only the file with the specified name and date is deleted; if date is not specified, then all files with the specified name are deleted. For disk requests, the date must be in the same format as that of the system date. For diskette requests, the date must be in the same format as creation date for the diskette file to be deleted. The date parameter is not valid when LABEL-ALL is specified.

### *DATA Parameter*

The DATA parameter lets you delete the specified file directly from the disk as well as from the VTOC.

If YES is coded in this parameter, then the specified file will be removed from the disk and any reference to it in the VTOC will be removed. In addition, a message will be logged on the display screen or printer for each file removed from the disk.

If NO is coded in this parameter, then the specified file will not be removed from the disk. However, any reference to it in the VTOC will be removed. If this parameter is not used, DATA-NO is assumed.

The DATA parameter is valid only with the REMOVE control statement.

### **\$DELET OCL and Utility Control Statement Sequence**

To initiate the \$DELET program through OCL, the following is required:

```
// LOAD $DELET
// RUN
// SCRATCH UNIT- { F1 } ,LABEL- { filename } [ ,DATE- { mmdyy } ] [ ,PACK-vol-id ]
                { I1 }      { ALL }      { ddmmyy }
                { I1 }      { ALL }      { yymmdd }
```

and/or

```
// REMOVE UNIT- { F1 } ,LABEL- { filename } [ ,DATE- { mmdyy } ] [ ,DATA- { NO } ] [ ,PACK-vol-id ]
                { I1 }      { ALL }      { ddmmyy }
                { I1 }      { ALL }      { yymmdd }
                { I1 }      { ALL }      { YES }
```

```
// 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
```



The NAME-ALL parameter indicates that all files on the inserted diskette are to be copied to another diskette. The NAME-filename parameter specifies the name of the single file that is to be copied from one diskette to another.

When all files on a diskette are copied, the contents of the input and output diskettes are the same, except, possibly, for the volume identification and alternate track information. Space between files is eliminated; the files are physically contiguous on the new diskette.

The diskette that is being copied can contain standard interchange data files or non-interchange files (see Appendix C). The diskette 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 space on the disk, then displays a message telling the operator to mount the diskette that is to contain the copy. After transferring the copy from the disk to a diskette, \$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 FILENAME-ALL is specified. DELETE-YES specifies that expired files on the input diskette are to be deleted. DELETE-NO specifies that expired files on the input diskette are to be copied. DELETE-NO 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-COPY11,UNIT-I1,...
// RUN
// COPY11...
// END
```

## **\$DUPRD Examples**

Copy all files on a diskette to the diskette with a vol-id of 123456.

```
// LOAD $DUPRD
// FILE NAME-COPY11,UNIT-11
// RUN
// COPY11 NAME-ALL,PACK-123456
// END
```

Copy diskette file (with filename of JIM and creation date of 01-02-75) to another diskette (with vol-id of 345678).

```
// LOAD $DUPRD
// FILE NAME-COPY11,UNIT-11,DATE-010275
// RUN
// COPY11 NAME-JIM,PACK-345678
// 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 statement 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**

<b>Use</b>	<b>Control Statement</b>
Display only previously displayed HISTORY file data	<code>[// DISPLAY] // END</code>
List complete contents of HISTORY file (including items not previously displayed)	<code>// DISPLAY ALL // END</code>

### **\$HIST Parameters**

The DISPLAY utility control statement requests that part or all of the HISTORY file be displayed or printed. If the DISPLAY statement is not entered, the only information listed will be that information displayed or printed at the time it was logged.

**ALL** ALL specifies that every entry in the HISTORY file is to be displayed or printed. If ALL is not specified, only those entries previously viewed by the operator (information displayed at the time it was logged) are shown.

### **\$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
// END
// LOAD $HINT
// RUN
```

**Note:** \$HINT is the utility program that clears the HISTORY file if RESET is specified in the HISTORY command statement. (See index entry: *HISTORY command statement*.)

## \$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 an IBM standard interchange 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* and Appendix C).

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 of the fact via the display screen and can choose to cancel the job or continue. If the operator continues the job, active files are deleted. 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 of the fact via the display screen and initialization is terminated. Otherwise, if one or two bad tracks are found, their addresses are preserved in the ERMAP field on track 0 (see *The IBM Diskette for Standard Data Interchange*, GA21-9182).

To complete initialization, all data sectors (tracks 1–76) 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, OWNER-ID is used.

The VTOC is initialized to indicate that one file, DATA, occupies tracks 1–73, and DATA is empty.

*Note:* All diskettes for a multivolume file must be initialized in the same format; all diskettes in the file must be in the standard interchange format or have 512-byte data sectors.

### 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, OWNER-ID 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 process 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 beginning 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.

The diskette containing the defect can be used by the system only for input, unless the file that was being created when the defect was detected is expired or deleted. 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	<pre>// UIN OPTION- {FORMAT }                   {FORMAT2 } [// VOL [PACK-vol-id] [,ID-owner-id]   [      system date] [,OWNER-ID] ] // END</pre>
Delete files on a diskette	<pre>// UIN OPTION-DELETE [// VOL [PACK-vol-id] [,ID-owner-id]   [      system date] [,OWNER-ID] ] // END</pre>
Rename a diskette	<pre>[// UIN OPTION-RENAME] [// VOL [PACK-vol-id] [,ID-owner-id]   [      system date] [,OWNER-ID] ] // END</pre>



## \$INIT Parameters

### UIN Statement

The UIN statement specifies which \$INIT option is selected.

OPTION- { FORMAT } FORMAT2 }	Initializes a diskette as a standard interchange 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: <i>INIT command statement</i> .
OPTION-DELETE	Deletes files on a diskette.
OPTION-RENAME	Renames a diskette. RENAME is the option selected if no option is specified.

### VOL Statement

The VOL statement provides information to be inserted in the volume label.

<u>PACK-vol-id</u> <u>system date</u>	The PACK parameter specifies the vol-id. If the PACK parameter is not used, the system date is the default.
<u>ID-owner-id</u> <u>OWNER-ID</u>	The ID parameter specifies information for the owner-id field of the volume label. If the ID parameter is not used, OWNER-ID is the default.

### \$INIT Parameter Summary

OPTION- { FORMAT } FORMAT2 }	<p>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.</p> <p>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 flaw is found within a track, the entire track is marked defective (IDs all X'FF') and an alternate is assigned. If more than two bad tracks are found, the job is terminated and the diskette is not usable.</p> <p>Tracks are initialized by writing sectors of blanks on all data tracks (1-76). 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 OWNER-ID are written in the vol-id and owner-id fields, respectively.</p>
------------------------------------	--

For more details on **FORMAT** and **FORMAT2**, see index entry: *INIT command statement*

**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, **OWNER-ID** is placed in the owner identification field.

**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 alphanumeric 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 mounted diskette.

**ID-owner-id**  
OWNER-ID

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, **OWNER-ID** 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
    FORMAT2
    DELETE
    RENAME
  }
  [
    // VOL [PACK-vol-id] [ID-owner-id]
    [system date] [OWNER-ID]
  ]
// END
```

### \$INIT Examples

In order to name a diskette JIM, you could enter:

```
// LOAD $INIT
// RUN
// VOL PACK-JIM
// END
```

In order to initialize to 128-byte data sectors, test, name (APRO), and insert an owner identification of FRANT, you would enter:

```
// LOAD $INIT
// RUN
// UIN OPTION-FORMAT
// VOL PACK-APRO,ID-FRANT
// END
```

## \$LABEL-VTOC DISPLAY UTILITY PROGRAM

The \$LABEL utility program displays on the display screen or prints on the printer, depending on the current SYSLIST assignment (see index entry: *SYSLIST procedure*), either an entire disk or diskette VTOC (volume table of contents), or a specific VTOC entry.

The displayed information is an up-to-date record of the contents of the disk or diskette. Some common reasons for wanting such information are:

- To check the contents of a diskette before reinitializing it to ensure that it does not contain active files
- To find out how much disk space is available for new files
- To obtain specific file information, such as the file name, designation (permanent, temporary, scratch), or the space reserved for the file

\$LABEL is evoked by the CATALOG procedure (see index entry: *CATALOG procedure*).

### Sample VTOC Displays

In the sample VTOC displays that follow, each item shown is explained after the display.

#### Disk VTOC

DEVICE CAPACITY = 5.0 MEGABYTES  
AVAILABLE SPACE ON PACK- LOCATION BLOCKS

PACK-	SYSTEM	UNIT-	F1	DATE	RECORD COUNT	RECORDS AVAILABLE	RETAIN TYPE	FILE ORG	RECORD LENGTH	FILE LOCATION	KEY LENGTH	KEY LOCATION	CREATION FORMAT	UNITS ALLOCATED
				729 5 937 1049 (date)										
#LIBRARY	99/99/99						P	S	00	18			BLOCKS	700
PRMSTR	01/03/75				30	60	T	I	256	718	08	08	BLOCKS	10
BCKORD	01/10/75				01	39	T	S	64	728			BLOCKS	01
KRCINV	01/17/75				100	38	P	I	37	734	05	05	RECORDS	100
TEMPRM	01/24/75				200	16184	T	I	20	737	08	08	BLOCKS	200

\*\*\*\*\*END OF VTOC DISPLAY\*\*\*\*\*

#### DEVICE CAPACITY

The capacity in megabytes (one megabyte is one million bytes) of the disk used by the system.

#### AVAILABLE SPACE

The LOCATION (block number, in decimal) of a block of available space, and the number, in decimal, of BLOCKS of space available at that location. LOCATION and BLOCKS are given for all free space on the disk.

#### PACK

The volume identification given in the disk volume label. The disk volume label (VOL1) is located on track 0.

#### 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, D for direct.
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 index file.
KEY LOCATION	The position, in decimal, of the rightmost byte of the key in the records in an index 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 used to allocate a file.

*Note:* If the RECORDS parameter was used to allocate space, the number shown may 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 NAME	FILE DATE	FILE LENGTH	FILE TYPE	RECORD LENGTH	FILE LOCATION	EXPIRATION DATE	MVF FILE	SEQUENCE NUMBER	OLMV BLOCK SIZE
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 standard interchange diskettes is described in *The IBM Diskette for Standard Data Interchange*, GA21-9182. See also Appendix C.
- ID** The owner identification given in the diskette volume label. The volume label for standard interchange diskettes is described in *The IBM Diskette for Standard Data Interchange*, GA21-9182. See also Appendix C.
- SPACE AVAILABLE** The number, in decimal, of sectors, 128 bytes or 512 bytes each, available on the diskette. The number represents space following the last active (unexpired) file on the diskette.
- 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 the 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 will increase 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.
- FILE TYPE** Indicates a file's interchange type; STANDARD, SYSTEM, or UNDEFINED. See Appendix C for a description of the standard interchange and system files.
- RECORD LENGTH** The length, in decimal number of bytes, of the records in a file.

FILE LOCATION	The decimal sector number of the beginning of data in a file.
EXPIRATION DATE	The expiration date of a file. PROTECT indicates that a diskette file is permanent (RETAIN-999).
MVF FILE	Indicator to specify whether a file is a multivolume file: blank for a nonmultivolume file, CONTINUED for any volume but the last volume of a multivolume file, LAST for the last volume of a multivolume file.
SEQUENCE NUMBER	A diskette's sequence number within a multivolume file if the diskette is part of a multivolume file.
OLMV BLOCK SIZE	The number of disk blocks used when creating the offline MVF.

### \$LABEL Utility Control Statement Formats

Use	Control Statement
Display the entire VTOC	// DISPLAY UNIT- $\left. \begin{matrix} \{I1\} \\ \{F1\} \end{matrix} \right\}$ ,LABEL-ALL // END
Display a VTOC entry for a particular data file	// DISPLAY UNIT- $\left. \begin{matrix} \{I1\} \\ \{F1\} \end{matrix} \right\}$ ,LABEL-filename // END

### \$LABEL Parameters

UNIT-F1	Indicates that the disk contains the VTOC information to be displayed or printed.
UNIT-I1	Indicates that a diskette contains the VTOC information to be displayed or printed.
LABEL-ALL	Specifies that the entire contents of the VTOC are to be displayed or printed.
LABEL-filename	Indicates the VTOC entry for a particular file is to be displayed or printed. If the disk contains more than one file with the same name, the program will display or print all those entries.

### \$LABEL OCL and Utility Control Statement Sequence

To initiate \$LABEL, the following statements are required:

```
// LOAD $LABEL
// RUN
// DISPLAY UNIT-  $\left. \begin{matrix} \{F1\} \\ \{I1\} \end{matrix} \right\}$  ,LABEL-  $\left. \begin{matrix} \{ALL\} \\ \{filename\} \end{matrix} \right\}$ 
// END
```

## \$LOAD—RELOAD LIBRARY UTILITY PROGRAM

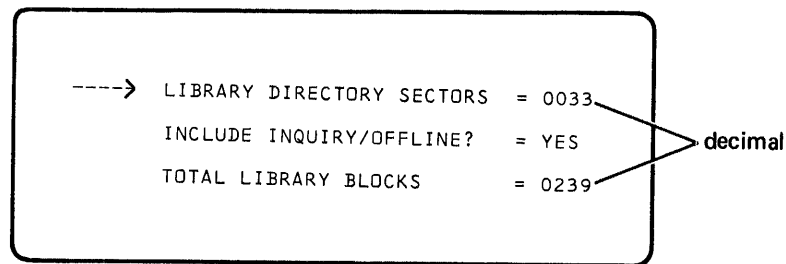
The \$LOAD utility returns to the disk a library backed up on diskette by the \$BACK utility (see index entries: *\$BACK utility program* and *BACKUP procedure*). When the \$LOAD utility is used, the vol-id in the volume label of the first input diskette becomes the vol-id in the volume label of the disk. The \$LOAD utility assigns error recording table areas on the disk.

Though \$LOAD is the name of the reload utility, the utility is composed of two programs: \$LOAD and \$LOADI. \$LOAD loads \$LOADI, a stand-alone program that actually performs the function of returning a library to the disk. \$LOADI can also be loaded by an IPL performed from diskette.

The \$LOAD utility displays space allocations specified for the library file (#LIBRARY) and the library directory within the library file. \$LOAD also displays the setting of the INQUIRY/OFFLINE option. For example,

```
----> LIBRARY DIRECTORY SECTORS = 0033
        INCLUDE INQUIRY/OFFLINE? = YES
        TOTAL LIBRARY BLOCKS    = 0239
```

decimal



Space allocations can be altered when they are displayed (For a detailed description of how to change the allocations, see index entry: *RELOAD display*). The maximum number of 256-byte sectors that can be allocated to the library directory is 256. The number of entries the directory can contain is: number of directory sectors x 11 minus 23. Each entry requires 23 bytes. A description of the contents of each entry is given in the description of the \$MAINT utility copy functions (see index entry: *printing from the library*). Each library block contains 2560 bytes.

Unaltered allocations remain at the values in effect when the diskette file was created by \$BACK. If the INQUIRY/OFFLINE option is not changed during the display, it too remains at the setting in effect when the file was created by \$BACK.

\$LOAD is evoked by the RELOAD procedure (see index entry: *RELOAD procedure*).

*Note:* The \$LOAD utility provides the only method whereby you can change the size of the library directory (alter the space allocated to it). When using \$LOAD (RELOAD), 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 library members on the disk that you want to save, 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 may 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 given back 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 has been pressed and the 1 option chosen), a rollout routine moves the interrupted program from main storage to the disk.
2. The program for which the interrupt was requested must be loaded normally. The interrupting program can be any type. This interrupting program cannot be interrupted, but can be cancelled.

*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, the interrupted program moves back into main storage using a rollin routine. The interrupted program begins execution at the point of interruption and terminates in a normal manner.

If the inquiry option is selected, a rollout area on the disk is allocated 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.

*Note:* If the operator presses the INQ key after pressing the STOP key and the IPL diskette switch is in the on position, 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. (For additional information, refer to *IBM System/32 Data Areas and Diagnostic Aids*, SY21-0532.)

### CAUTION

When a program that allows inquiry mode is interrupted by inquiry, 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 program or erroneous results when the interrupted program regains control.

The system and disk oriented functions listed below have the potential for such abnormal termination and erroneous results when executed in an inquiry mode:

Utilities — all functions of which are always prohibited:

Utility	Function(s)
\$BACK	(Back up library)
\$LOAD	(Reload library)
\$PACK	(Compress file space)
\$REBLD	(Rebuild VTOC)
\$SETCF	(Reconfigure system)
\$BUILD	(Alternate sector)

Utilities — some functions of which are prohibited:

Utility	Function(s)
\$COPY	(Restore all/save all)
\$DELET	(Delete all)

Utilities — processing of active files prohibited:

Utility	Function(s)
\$BICR	(TRANSFER active file)
\$COPY	(SAVE/ORGANIZE active file)
\$DELET	(DELETE active file)

Utilities — processing permitted with warning message:

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-11
// RUN
```

## \$MAINT—LIBRARY MAINTENANCE UTILITY PROGRAM

\$MAINT is evoked by the FROMLIBR, LISTLIBR, REMOVE, and TOLIBR procedures (see index entries: *FROMLIBR procedure*, *LISTLIBR procedure*, *REMOVE procedure*, and *TOLIBR procedure*). \$MAINT has three major functions:

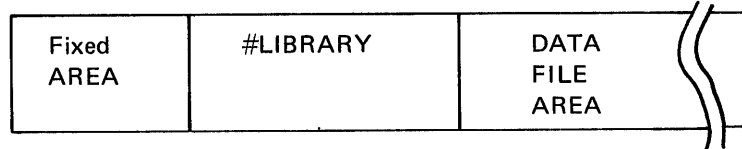
- *Allocate*. Allocate specifies and changes the size of the library file (#LIBRARY).
- *Copy*. Copy
  1. Places library members in the library
  2. Copies library members within the library
  3. Copies library members to a file on the disk or on a diskette
  4. Displays or prints the contents of the library or directory
- *Delete*. Delete removes library members by deleting them.

The functions of \$MAINT and the related utility control statements are described here in detail, following a general description of the library.

## System Library File (#LIBRARY)

### Location

The library is the first file on the disk immediately following the 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

Reserved Area	Disk Volume Label (VOL1)	Error Logging Area	Directory Area	Rollout Area (Optional)	Scheduler Work Area (SWA)	Additional Main Storage Dump Area	Library Members
---------------	--------------------------	--------------------	----------------	-------------------------	---------------------------	-----------------------------------	-----------------

**Disk volume label (VOL 1).** 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 or 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 programming.

**Additional main storage dump area.** This area is set aside for the 24K and 32K 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 ('), and blanks. Because the question mark (?) has a special meaning in procedures (see index entry: *procedure parameters*) and in certain control statements, the question mark should not be used in member names. The first character of a member name must be alphabetic (this also 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, all 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 members before adding more, the space they occupied in the library is freed and can be used 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. Sectors between members are unusable. If the number of unusable sectors becomes high, the library should be reorganized by performing a BACKUP and RELOAD sequence (see index entry: *BACKUP procedure* and *RELOAD procedure*). When a library is reorganized, members are shifted so that gaps do not occur between them—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*

<b>Use</b>	<b>Control Statement</b>
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  | <ul style="list-style-type: none"><li>● Add a procedure or source member to the library, or replace a procedure or source member in the library. Reader refers to the keyboard, by which the member is entered into the system.</li></ul> |
| Library-to-Library | <ul style="list-style-type: none"><li>● Copy a member from the library to the library, changing the name of the member.</li><li>● Copy a member having a certain name or all members having the name.</li></ul>                           |

Library-to-File  
(record mode or  
sector mode)

- 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 (system control programming) members.
- Copy a member from the library to a file.
- 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 all members except SCP members.
- 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 all members that have had a PTF applied to them (sector mode only).
- Add library member(s) to an existing file that contains library members.

File-to-Library

- Copy a member or members from a file to the library.
- Copy members, selecting members that have a specified PTF log number.

*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 of *Library-to-File* and index entry: *TOLIBR procedure*.

Library-to-Printer

- Print a member having a certain name or all members having the name.
- Print all members of a certain type.
- Print members, either of a specified type or of all types, that have names beginning with certain characters.
- 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.

- Print the directory entries for members of a certain type.
- Print all directory entries and system information in the directory area.
- Print the system information in the directory area.
- 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, 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.

#### 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 } NAME-name,TO-F1 [ ,RETAIN-P ] ,RECL-number
                          { S }
```

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 } NAME-name,TO-F1,NEWNAME-name [ ,RETAIN-P ]
                          { P }
                          { O }
                          { R }
```

- Copy a member having a certain name or all members having the name:

```
// COPY FROM-F1,LIBRARY- { S } NAME-name,TO-F1 [ ,RETAIN-P ] ,NEWNAME-name
                          { P }
                          { O }
                          { R }
                          ( ALL )
```



- 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
                          R
                          ALL } ,NAME-characters.ALL,

TO-F1 [ ,RETAIN-P
       ,RETAIN-R ] ,NEWNAME-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:

```
// COPY FROM-F1,LIBRARY- { S
                          P
                          O
                          R
                          ALL } ,NAME- { name
                                          characters.ALL } ,

TO-F1 [ ,RETAIN-P
       ,RETAIN-R ] ,NEWNAME-characters,OMIT- { name
                                                characters.ALL
                                                SYSTEM }
```

*Library-to-File, Record Mode:* Control statements depend on the function required.

- Copy or add a library member to a file:

```
// COPY FROM-F1,TO-DISK,FILE-filename { ,RECL-number
                                         ,ADD-YES } ,NAME-name,

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
                                         ,ADD-YES } ,NAME-name,

LIBRARY- { S
          P
          ALL }
```

- Copy or add members, either of a specified type or of all (both) permitted types, that have names beginning with certain characters:

```
// COPY FROM-F1,TO-DISK,FILE-filename { ,RECL-number
                                         ,ADD-YES } ,NAME-characters.ALL,

LIBRARY- { S
          P
          ALL }
```

- Copy or add all members of both 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 (both) 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 }
           { P } [ ,ADD-YES ]
           { 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 }
           { P } [ ,ADD-YES ]
           { 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 } [ ,ADD-YES ]
           { O }
           { R }
           { ALL }
```

- 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 } [,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 } ] [,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
```

*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-  $\left. \begin{matrix} S \\ P \\ O \\ R \\ ALL \end{matrix} \right\}$  ,NAME-name,TO-PRINT

- Print all members of a certain type:

// COPY FROM-F1,LIBRARY-  $\left. \begin{matrix} S \\ P \\ O \\ R \end{matrix} \right\}$  ,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-  $\left. \begin{matrix} S \\ P \\ O \\ R \\ ALL \end{matrix} \right\}$  ,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-  $\left. \begin{matrix} S \\ P \\ O \\ R \\ ALL \end{matrix} \right\}$  ,NAME-  $\left. \begin{matrix} \text{name} \\ \text{characters.ALL} \end{matrix} \right\}$  ,  
TO-PRINT,OMIT-  $\left. \begin{matrix} \text{name} \\ \text{characters.ALL} \\ \text{SYSTEM} \end{matrix} \right\}$

- Print the directory entries for members of a certain type:

// COPY FROM-F1,LIBRARY-  $\left. \begin{matrix} S \\ P \\ O \\ R \end{matrix} \right\}$  ,NAME-DIR,TO-PRINT

- Print all directory entries and system information in 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 } ,NAME-DIR,TO-PRINT,
                          O
                          R
                          ALL)

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, 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); 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 from (entered on) the keyboard.
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	<ul style="list-style-type: none"> <li>● For copying library-to-file in record mode, specifies source (S) and procedure (P) members.</li> <li>● For printing from the directory area, specifies that system information as well as directory entries are to be printed.</li> <li>● For all uses of copy except the two just listed, specifies that all member types (S, P, O, and R) are involved.</li> </ul>

LIBRARY-SYSTEM	Only system information is to be printed from the directory area.
NAME-name	Name of the member to be added, replaced, copied, or printed.
NAME-characters.ALL	Only those members whose names begin with the indicated characters (maximum of seven) are to be copied or printed. For example, NAME-PAY.ALL copies or prints all members whose names begin with the characters PAY. (Duplicate members with a different name may be created in a library-to-library copy. See NEWNAME-NAME and NEWNAME-characters parameter described later in this list.)
NAME-ALL	All members except SCP members are to be copied, or all the members of a certain type are to be printed.
NAME-DIR	Directory entries for all library members of the type indicated in the LIBRARY parameter are to be printed. If the LIBRARY parameter is LIBRARY-ALL, system information is also to be printed. NAME-DIR is only valid with TO-PRINT.
TO-F1	The library will contain the copied members.
TO-DISK	The copied members will be in a disk or diskette file. A FILE statement identifies the unit (F1 or I1).
TO-PRINT	The specified members or directory information is to be printed.
<u>RETAIN-P</u>	A member is being entered in the library. If a member of the same name as the new member already exists in the library, the system displays a message to that effect on the display screen and requires an operator response before replacing the existing member with the new member. If the RETAIN parameter is omitted, RETAIN-P is the default.
RETAIN-R	A member is being entered in the library. If a member of the same name as the new member already exists in the library, it is replaced by the new member. The operator is not notified that a duplicate entry existed.
RECL-number	This parameter specifies the record length of the statements for a source or procedure member. The record length can be from 40 through 120.  The RECL parameter also specifies that a copy to a file is in record mode and not in sector mode.

*Record Mode:* Record mode is specified by the RECL-number 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 record and followed by a CEND record. (The format of the COPY record is // COPY NAME-name,LIBRARY- $\left\{ \begin{array}{l} P \\ S \end{array} \right\}$  where name is the member name and P or S indicates procedure or source member. The format of the CEND record 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 keyword RECL (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.
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 numerical 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 ...]   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.
// RUN
// COPY ...
// END
```

#### *Using the Copy Function*

*Naming Library Members:* Considerations that apply to naming library members are:

- Member names can be any combination of characters (numeric, alphabetic, and special) except commas, periods, single quotes ('), and blanks. The question mark (?) should not be used because it has special meaning in procedures (see index entry: *procedure parameters*) and certain control statements. 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. The first character of each member name must be alphabetic.
- 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 a different type 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
```

**Figure 7. Sample Printout of System Information**

The first of the two numbers given in the column at the right is a decimal number, the second is hex. (In the example 184/00B8, 184 is the decimal value of hex 00B8.)

## 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	X

Figure 8. Information in Printout of Library Directory Entry

Following is an explanation of the fields shown in Figure 8.

<b>TYPE</b>	Type of library member described by the entry: S source member P procedure member O load member R subroutine member
<b>NAME</b>	Name of the library member.
<b>START ADDR</b>	Sector number of the first sector of the member in both decimal and hex.
<b>TOTAL</b>	Total number of sectors in the member, in decimal and hex.
<b>NUM TEXT/RECORD</b>	For source and procedure members, record length of the member, given in decimal and hex. For load members, the number of text sectors contained in the member, excluding RLDs— <i>relocation directories</i> —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.)
<b>ATTRIBUTES</b>	Two bytes, 16 bits, of <i>attributes</i> giving detailed characteristics of the member.

<b>Bit</b>	<b>Meaning When On (1)</b>
------------	----------------------------

## Byte 0:

0	This member is an SCP member. This bit is used to prevent SCP members from being deleted.
1	Reserved.
2	Reserved.
3	Reserved.
4	This program requires that \$WORK and \$SOURCE be allocated. \$SOURCE must be filled from the keyboard or a source member.
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 *// COMPILER* statement (see index entry: *// COMPILER statement*) and a no-source-required attribute (bit 4 of byte 0 off-0).

3 Reserved.

| 4 This SCP module 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 member contains a where-to-go table. It is used by the transient cross reference resolver program (#OXRF).

<b>LINK ADDR</b>	For load members only. The main storage address, in decimal and hex, assigned to the member when it is linked in main storage with other load members.
<b>RLD DISP</b>	For load members only. Displacement, in decimal and hex, of first RLD (relocation directory) in member in first sector containing RLDs.
<b>ENTRY ADDR</b>	For load members only. Main storage address, of entry point of member in decimal and hex.
<b>PROG SIZE</b>	For load members only. Decimal and hex number of sectors required to run the program contained in the member.
<b>LEVEL</b>	The release level of the member.

### *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 (compressed data, hex format) 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 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,LIBRARY-ALL,OMIT-PAY.ALL
// END
```

Add all members whose names begin with the characters PA, omitting members whose names start with PAY. Add the members to a disk file named PAYR.

```
// LOAD $MAINT
// FILE NAME-PAYR,UNIT-F1
// RUN
// COPY FROM-F1,TO-DISK,FILE-PAYR,NAME-PA,LIBRARY-ALL,
  OMIT-PAY.ALL,ADD-YES
// END
```

*File-to-Library:* The following examples demonstrate copying from a file to the library.

Copy library member(s) from a disk file named SAVED. If the file contains a member whose name and type are the same as a member currently existing in the library, the existing member will not be replaced unless the operator chooses to replace the member after being notified (by a message on the display screen) that a duplicate member exists (that is, RETAIN-P is assumed to be on the COPY control statement).

```
// LOAD $MAINT
// FILE NAME-SAVED,UNIT-F1
// RUN
// COPY FROM-DISK,TO-F1,FILE-SAVED
```

Copy library member(s) from a diskette file named LMT. The operator will not be notified if LMT contains a member whose name and type are the same as a member currently existing in the library.

```
// LOAD $MAINT
// FILE NAME-LMT,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-R,FILE-LMT
```

*Library-to-Printer:* The following is an example of a library-to-printer copy.

Copy (print) the system information in the library directory area and copy (print) all entries in the directory:

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,LIBRARY-ALL,NAME-DIR,TO-PRINT
// END
```

## **Delete Function**

### *Delete Uses*

- Delete a non-SCP member having a certain name or all non-SCP members having the name
- Delete non-SCP members, either of a specified type or of all types, that have names beginning with certain characters

- Delete all non-SCP members of a specified type
- Delete non-SCP members, either of a specified type or of all types, except members that have a certain name or have names beginning with certain characters
- Delete all non-SCP members
- Delete specified members, including SCP members

**Note:** The following restrictions apply to using the delete function:

- The library cannot be totally deleted.
- Except when tailoring your system (see index entry: *system modification*), you should not delete SCP members.

#### **Delete Control Statement Formats**

- Delete a non-SCP member having a certain name or all non-SCP members having the name:

```
// DELETE LIBRARY- { S
                    { P
                    { O },NAME-name
                    { R
                    { ALL
```

- Delete non-SCP members, either of a specified type or of all types, that have names beginning with certain characters:

```
// DELETE LIBRARY- { S
                    { P
                    { O },NAME-characters.ALL
                    { R
                    { ALL
```

- Delete all non-SCP members of a specified type:

```
// DELETE LIBRARY- { S
                    { P
                    { O },NAME-ALL
                    { R
```

- Delete non-SCP members, either of a specified type or of all types, except members that have a certain name or have names beginning with certain characters:

```
// DELETE LIBRARY- { S
                    { P
                    { O },NAME- { name
                    { R           { characters.ALL }
                    { ALL

OMIT- { name
      { characters.ALL }
```

- Delete all non-SCP members:

```
// DELETE LIBRARY-ALL,NAME-ALL
```

- Delete specified members, including SCP members:

```
// DELETE LIBRARY- { S } ,NAME- { name } ,
                   { P } , { characters.ALL } ,
                   { O } , { ALL } ,
                   { R }

[ OMIT- { name } ,RETAIN-S
         { characters.ALL } ]
```

#### *Delete Parameters*

LIBRARY-S	Source members are deleted.
LIBRARY-P	Procedure members are deleted.
LIBRARY-O	Load members are deleted.
LIBRARY-R	Subroutine members are deleted.
LIBRARY-ALL	All types (S, P, O, and R) are deleted.
NAME-name	Name of the member or members being deleted.
NAME-characters.ALL	Only those members whose names begin with the indicated characters (maximum of seven) are deleted. For example, NAME-INV.ALL deletes members whose names begin with the characters INV.
NAME-ALL	All members of a specified type are deleted, or all members except SCP members are deleted. When NAME-ALL and LIBRARY-ALL are specified, only non-SCP members are deleted.
OMIT-name	Members of the name specified are <i>not</i> deleted.
OMIT-characters.ALL	Members whose names begin with the specified characters (maximum of seven) are <i>not</i> deleted.
RETAIN-S	SCP members identified by other parameters in the delete control statement are deleted.

#### *Delete OCL and Utility Control Statement Sequence*

```
// LOAD $MAINT
// RUN
// DELETE ...
// END
```



*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
```

## \$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 } ] [ ,REPLACE- { YES } ]  
                             [ NO ] [ NO ]
```

### \$MGBLD Parameters

**SOURCE-sourcename** Specifies the name of the library message source member that contains the control statement and message text statements (MIC and text) required for creation of a message load member. See index entry: *message source member* for information about message source members.

**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. (Default is NO.)

**REPLACE- YES**  
**NO** If YES is specified, the message load member replaces a member with the same name, if one exists.

If NO is specified, a member having the same name is not replaced—an error message is displayed on the display screen if an attempt is made to replace a member. NO is a default value.

### \$MGBLD OCL and Utility Control Statement Sequence

The statements required to initiate the \$MGBLD program are:

```
// LOAD $MGBLD  
// RUN  
// MGBLD SOURCE-sourcename, ...  
// END
```

## Message Source Member

The message source member is put into the library like any other source member. For example, it might be copied from reader-to-library or library-to-library—see index entry: *\$MAINT utility program*—or entered by a program such as the source entry utility described in *IBM System/32 Utilities Program Product Reference Manual—Source Entry Utility*, SC21-7605.

Two types of statements are required in the message source member: a *control statement* and one or more *message text statements* (comment statements are optional).

### Control Statement

The control statement specifies the name of the message load member to be created and whether a first or second level message load member is being created. The format of the control statement is:

load-name [,level]

load-name	The name to be given to the message load member created.
level	A value of 1 specifies that a first level message load member is to be created (maximum of 40 characters per message, excluding the MIC); a value of 2 specifies that a second level message load member is to be created (maximum of 200 characters per message, excluding the MIC). If the level parameter is not specified, 1 is assumed.

*Note:* Level 2 messages cannot be displayed independently of level 1 messages. A level 2 message can be displayed only after the level 1 message of the same MIC is displayed.

### Message Text Statement

The format of the message text statement is:

MIC Text

- MIC (message identification code). MIC must be specified within the first four characters of the message text statement. It is a 1 to 4 character decimal number from 0 to 9999. It must be in ascending order, relative to the message identification code for the preceding message text statement, unless the same MIC is specified on consecutive message text statements to cause concatenation of 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.
- 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) containing the same MIC. The text area on following statement(s) is appended to the text area for the first statement before trailing blanks for the total text specified are dropped. A message text of one character 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 have an \* as their 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 and Load Member**

Assume you want to enter the following statements into the library as a message source member named USERMI. Assume also that you want to enter them via the keyboard; that is, via the reader-to-library copy function of \$MAINT (see index entry: *\$MAINT utility program*).

```
USERMSG,1      (This is a control statement; USERMSG is the load-name and
                1 is the message level.)
```

```
1234 ENTER YESTERDAY'S DATE.  (These are message text statements.
1235 ENTER TODAY'S DATE.      1234, 1235, and 1236 are MICs. The
1236 ENTER TOMORROW'S DATE.   message text follows the MICs.)
```

```
* THE ABOVE MESSAGES ARE FOR PROGX. (This is 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-40
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
```

## **\$PACK—DISK REORGANIZATION UTILITY PROGRAM**

\$PACK reorganizes the disk so that all free space on the disk is collected in one area. The reorganization is accomplished by successively moving each data file as close to the library as possible.

If a file is being moved to a free 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 (see index entry: *COMPRESS 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.

### **\$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
```

## \$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.

\$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**      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 indexed entry before it is checked, and so on, until an index 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 by normal end-of-job processing.

**Direct file**         The format 1 is updated to a completed status and written to the disk VTOC by 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 is run, messages are issued giving the LABELs (from // FILE statements), creation dates, and organization (sequential, indexed, or direct) 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*).

*Note:* Unless the system failed while the \$PACK utility was being used (see index entry: *\$PACK utility*), \$REBLD must be the first program run after a system failure.

## \$REBLD Utility Control Statement Format

Utility control statements are not used.

## \$REBLD OCL Sequence

```
// LOAD $REBLD
// RUN
```

## \$SETCF—SET UTILITY PROGRAM

\$SETCF is used to define the following items:

- System environment
- BSCA environment
- OVERRIDE RPG BSCA specifications
- TRACE functions

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 altered by \$SETCF. If a value is never altered, it retains its original status. If an item is altered, the new value is reflected in subsequent IPLs until the item is changed again (except for the DEBUG-Y parameter which is reset by IPL or by the TRACE procedure).

\$SETCF is evoked by the SET, ALTERBSC, OVERRIDE, and TRACE procedures (see index entries: *SET procedure*, *ALTERBSC procedure*, *OVERRIDE procedure*, and *TRACE procedure*).

## Set the System Environment

The following system environment items can be defined by \$SETCF:

- BSCA
- 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 } ] [ ,FORMAT- { DMY }
{ YMD } ]
```

**Note:** Though each particular 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.

### *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 outside \$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 BSCA Environment

The following BSCA (binary synchronous communications adapter) items can be set by \$SETCF:

- BPS (bits per seconds) rate
- Modem clocking
- Debug facility
- Error retry count
- Standby line
- Modem test
- Non-U.S.A. tone

*Note:* The items listed are all related to telecommunications programming that uses the BSCA. For background information on binary synchronous communications, see *General Information – Binary Synchronous Communications, GA27-3004*.

### *SETB Utility Control Statement Format for Setting the BSCA Environment*

Use the SETB utility control statement to set the BSCA environment:

```
// SETB [ BRATE- { F } ] [ ,CLOCK- { Y } ] [ ,DEBUG- { Y } ] [ ,ERC- { number } ]  
[ ,SLINE- { Y } ] [ ,TEST- { Y } ] [ ,TONE- { Y } ]
```

For an explanation of the SETB parameters, see *ALTERBSC Command Statement Format*.

*Note:* Though each 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 by IPL or by the TRACE procedure). If DEBUG-Y is specified, the system TRACE procedure (see index entry: *TRACE procedure*) is replaced by the BSCA trace function.

### *Parameters For Setting the BSCA 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, BSCA trace is requested.
N	Built-in debug facility is not required, BSCA 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 a number up to 255. Valid numbers are 7 through 255.
<u>7</u>	
SLINE-Y	Switched line will be used as backup (standby) line for a point-to-point line.
N	The normal line is to be used.
TEST-Y	IBM data modem is being used. Automatic wrap test includes modem testing when a permanent error occurs, unless the RPG II program specified a permanent error indicator for the BSCA file.
N	Non-IBM data modem is being used. Automatic wrap test does not include modem testing.
TONE-Y	Non-U.S.A. special tone is required.
N	Non-U.S.A. special tone is not required.

#### *Example of Setting the BSCA Environment*

Change the current BSCA error retry count to 10:

```
// LOAD $SETCF
// RUN
// SETB ERC-10
// END
```

#### **Override RPG BSCA Specifications**

The following RPG BSCA (binary synchronous communications adapter) specifications can be overridden by \$SETCF:

- Tributary station address
- Line type

*Note:* The items listed are all related to telecommunications programming that uses the BSCA. For background information on binary synchronous communications, see *General Information – Binary Synchronous Communications, GA27-3004*.

#### *Utility Control Statement Format for Overriding RPG II Specifications*

```
// SETR [ADDR-nn] [LINE- { C
                          P
                          R
                          S
                          T } ]
```

#### *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 RPG specifications, reenter a valid // SETR control statement omitting the ADDR parameter. The addressing characters will default to the RPG specifications.

Parameters for overriding RPG II BSCA Specifications:

ADDR-nn      Hex equivalent of one of the pair of tributary station addressing characters. See Appendix G for the S/32 tributary station polling and addressing characters.

Defaults to RPG specifications.

LINE-C      CDSTL (connect data set to line) switched line (World Trade Only)

P            Point-to-point leased line.

R            Line type specified in RPG source statements.

S            Point-to-point switched line.

T            Tributary station line on multipoint.

An example of overriding RPG II BSCA specifications is:

Change an existing tributary station address to 1 (in effect, the addressing characters are changed to 11), EBCDIC line code:

```
// LOAD $SETCF
// RUN
// SETR ADDR-F1
// END
```

**Set Functions to be Traced**

A trace of the following system functions can be requested by using \$SETCF:

- Wait
- Disk IOS
- Control storage disk IOS
- Push
- Pull
- Disable interrupts
- Enable interrupts
- Queue
- Control storage load
- Main storage load
- Transient load

*Note:* Setting functions to be traced is a function of \$SETCF that is intended primarily for IBM service personnel. The function is evoked by the TRACE procedure (see index entry: *TRACE procedure*).

*Utility Control Statement Format for Setting Functions to be Traced*

```
// TRACE [ALL-Y] [WAIT- {Y} / {N}] [FDIOS- {Y} / {N}] [CSFDIOS- {Y} / {N}]
[PUSH- {Y} / {N}] [PULL- {Y} / {N}] [DISABLE- {Y} / {N}] [ENABLE- {Y} / {N}]
[QUEUE- {Y} / {N}] [LDCS- {Y} / {N}] [LOADER- {Y} / {N}] [XIENT- {Y} / {N}]
```

*Note:* The parameters can be specified in any order. (If ALL-Y is specified, all other parameters specified are ignored.) Though each particular parameter is optional, at least one parameter must be specified. A maximum of ten can be specified. However, the entire SCP trace function is disabled if DEBUG-Y is specified in the ALTERBSC command statement or in the // SETB Utility Control Statement of the \$SETCF utility (see index entries: *ALTERBSC procedure* and *BSCA environment*).

*Parameters for Setting Functions to be Traced*

ALL-Y	All traceable system functions are to be traced.
WAIT-Y	Each evocation of the wait function is to be traced.
<u>N</u>	Evocations of the wait function are not to be traced.
FDIOS-Y	Each evocation of disk IOS (input/output supervisor) is to be traced.
<u>N</u>	Evocations of disk IOS are not to be traced.
CSFDIOS-Y	Each evocation of control storage disk IOS is to be traced.
<u>N</u>	Evocations of control storage disk IOS are not to be traced.
PUSH-Y	Each evocation of the push function is to be traced.
<u>N</u>	Evocations of the push function are not to be traced.
PULL-Y	Each evocation of the pull function is to be traced.
<u>N</u>	Evocations of the pull function are not to be traced.
DISABLE-Y	Each evocation of the disable interrupt function is to be traced.
<u>N</u>	Evocations of the disable interrupt function are not to be traced.
ENABLE-Y	Each evocation of the enable interrupt function is to be traced.
<u>N</u>	Evocations of the enable interrupt function are not to be traced.

QUEUE-Y	Each evocation of the queue function is to be traced.
<u>N</u>	Evocations of the queue function are not to be traced.
LDCS-Y	Each evocation of the control storage transient loader is to be traced.
<u>N</u>	Evocations of the control storage transient loader are not to be traced.
LOADER-Y	Each evocation of the main storage relocating loader are not to be traced.
<u>N</u>	Evocations of the main storage relocating loader are not to be traced.
XIENT-Y	Each evocation of the main storage transient loader is to be traced.
<u>N</u>	Evocations of the main storage transient loader 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
```

**\$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
```

**Part 5.**

**System Configuration, Installation, and Modification**





## Introduction To System Configuration, Installation, and Modification

This part describes:

- Configuration and installation of IBM System/32 system control programming and any related PTFs at initial system installation or subsequent system update.
- Individual installation of IBM System/32 program products and any related PTFs (program temporary fix) and verification of correct installment.
- Modifying your system by deleting certain SCP components and/or program products from the library so that you have more disk space for other library members or data files.

Three SCP procedures are described in this part: CNFIGSCP, INSTALL, and APPLYPTF. The formats of the command statements that evoke these procedures are:

```
APPLYPTF [ SC1nn  
          RG1nn  
          UT1nn ] [ ,ALL  
                  ,ptfid ]
```

CNFIGSCP

```
INSTALL [DFU] [,SEU] [,SORT] [,RPG]
```

Other procedures are called by the preceding procedures. The other procedures can be called individually by their associated command statements and are described elsewhere in this manual.



This section describes configuration and installation of both your initial version and subsequent versions of IBM System/32 system control programming and program products.

### DISKETTES REQUIRED

The diskettes required to perform system configuration are:

- Two PID (program information department) distribution diskettes that contain the system control program (SCP).

They are called SCP diskettes.

- PID distribution diskettes containing any program products ordered.
- A diskette containing PTFs (program temporary fix) for the system control program and/or program products.

This diskette is called the PTF diskette. If this diskette is necessary, it will be provided by the customer engineer.

- Two of your diskettes on which a backup of the system control program can be made. They are called backup diskettes.
- Backup diskettes for each of the program products installed, if you want backup copies.
- Backup diskettes for the tailored SCP so that you can quickly reload the library in case of a problem.

### *Other Requirements*

During the building of your SCP, you will be prompted for the following information:

- Print belt image for your system
- The date format you will be using
- Optional SCP support for BSCA, if desired
- Optional SCP support for RPG II, if desired
- Optional BSCA support for RPG, if desired
- Whether or not a PTF diskette is available

Using the information supplied by the prompts, an SCP will be built that contains the support requested. If a PTF diskette is available, the PTFs are applied to the SCP on the disk.

## SYSTEM CONFIGURATION STEPS

The following steps create an SCP on the disk using the PID distribution diskettes. The SCP diskettes received from PID are used only during these system configuration steps, and should then be stored until the next SCP release is available.

### 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 following steps.

The system configuration steps are:

1. Set the IPL switch on the CE control panel to DISKETTE and set the IMPL switch to DISK.
2. Insert the first SCP diskette.
3. Press the LOAD key on the operator panel. The following display appears:

```
---> LIBRARY DIRECTORY SECTORS = 0037
      INCLUDE INQUIRE/OFFLINE  = NO
      TOTAL LIBRARY BLOCKS      = 0281
```

4. The values displayed are those of the PID diskette. If an error message is displayed, read the following procedures:

#### *Error Procedure For INVALID VTOC/LIBRARY*

It will be necessary to 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 have been saved, go back to 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 condition:

### 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 any other error messages. If there are none, go to step 5.

*Error Procedure When the Library is too Large*

You can do (or repeat) any part of the INVALID VTOC/LIBRARY error procedure, or you can decrease the library size by going to step 5.

*Error Procedure For Any Other Error Message*

**Note:** The errors covered by this procedure probably are the result of an error made in step 5.

Go to step 5. If you have already been to step 5, adjust your allocations for directory sectors and library blocks and press the REC ADV key after each entry until the error messages no longer appear; then go to step 7.

5. If you want a larger library or if you plan to apply PTFs to any program products, you should now allocate enough directory sectors and library blocks to contain the program products, the SCP, and any other programs. See index entry: *system modification*, for a description of how to allocate directory sectors and library blocks. See index entry: *library requirements*, to determine the number of directory sectors and library blocks required by the program products. Press the REC ADV key after each entry.
6. If any error messages are displayed, go back to step 4.
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.

```
INSERT DISKETTE WITH FILE LABEL-#LIBRARY
DATE- date, SEQUENCE NUMBER 02
---> PRESS ENTER KEY AFTER INSERTING
WARNING-LIBRARY MAY BECOME UNUSABLE
      IF CORRECT VOLUME NOT INSERTED
```

The second PID SCP diskette is copied to the disk when you press the ENTER key.

9. When the following display appears, set the IPL and IMPL switches to DISK and press the LOAD key.

```
RELOAD COMPLETE - REMOVE LAST
DISKETTE AND IPL FROM DISK
```

10. When the ENTER COMMAND message appears, enter a CNFIGSCP command statement of the format:  
CNFIGSCP
11. The system will prompt you for the variable information needed. Follow the instructions displayed on the display screen. The final message is SYSTEM CONFIGURATION COMPLETE.

The system you will be using has now been configured. If necessary, PTFs have been applied to the SCP. The SCP PID distribution diskettes should now be stored for safekeeping until the next SCP release.

At this time, a backup copy should be made of the system. This backup copy can then be used when building unique systems, as explained under *system installation steps*.

To make the backup copy of the SCP, it is necessary to initialize two diskettes on which a copy of the SCP can be kept. Use the INIT procedure (see index entry: *INIT procedure*) to accomplish this. After the two diskettes have been initialized, you are ready to backup the SCP system onto the diskettes. Use the BACKUP procedure (see index entry: *BACKUP procedure*) to make the backup copy.

Before proceeding with the system installation steps, you should determine if any of the program products you ordered require PTFs. If so, the PTFs should be applied at this time and a backup copy made of each program product. Even if no PTFs are needed, it is recommended that you make a backup copy of the program product PID distribution diskettes and use that copy when installing program products. The PID distribution copy should be stored for safekeeping until the next release from PID is available.

To apply PTFs to the program products, it is necessary to have the program products on the disk. See index entry: *program product installation*, for information on installing program products. When a program product has been copied to the disk, PTFs can be applied, if necessary. Use the APPLYPTF procedure to apply the PTFs (see index entry: *APPLYPTF procedure*). After required PTFs are applied (if any), backup copies of program products can be made. See index entry: *program product installation*, for information on how to create a backup copy of a program product. Backup copies of program products are used during system installation to create the unique systems desired.

## SYSTEM INSTALLATION STEPS

This section describes how to install a unique system using the backup diskettes created during the system configuration steps. The first function performed is setting the library size according to the options you give after entering the RELOAD command statement. Any required application programs should next be installed on the system.

The INSTALL procedure (see index entry: *INSTALL procedure*) is then used to:

- Load the selected program products onto the disk
- Initialize the diskettes needed to back up the system
- Copy the system onto these diskettes

The system installation steps are:

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 index entry: *system modification*, for a description of library requirements to determine the correct values for your system; (also, see index entry: *RELOAD display*, for a description of the display and a description of how to change the values displayed):

```
---> LIBRARY DIRECTORY SECTORS = 0033
      INCLUDE INQUIRY/OFFLINE? = NO
      TOTAL LIBRARY BLOCKS     = 0239
```

*Note:* The values shown in the preceding display are sample values only.

2. When the following display appears, insert the second SCP backup diskette.

```
INSERT DISKETTE WITH FILE LABEL-#LIBRARY
DATE- date, SEQUENCE NUMBER nn
---> PRESS ENTER KEY AFTER INSERTING
WARNING-LIBRARY MAY BECOME UNUSABLE
      IF CORRECT VOLUME NOT INSERTED
```

3. When the following display appears, press the LOAD key.

```
RELOAD COMPLETE - REMOVE LAST
DISKETTE AND IPL FROM DISK
```

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 loaded at this time. The letter accompanying IBM application package describes how to put the programs onto the system. Refer to that letter for installation instructions for IBM application programs.

6. Enter the INSTALL command statement of the format:  
INSTALL [DFU] [,SEU] [,SORT] [,RPG]

The particular command statement you enter depends on the program products you wish to install. See index entry: *INSTALL procedure*, for a description of the command statement parameters.

After the INSTALL command is entered, system directory information is printed—see index entry: *printing from the library*, for a description of the information printed.

7. The system will prompt for the diskettes needed to complete the system installation. You will also be 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 are not formatted, they are only renamed.

As initialized diskettes are inserted, the system is copied on them. The final message will be SYSTEM INSTALLATION COMPLETE. If you want additional unique systems, repeat the system installation steps as often as necessary.

## CALCULATING THE NUMBER OF BACKUP DISKETTES REQUIRED FOR THE SYSTEM

To determine the number of diskettes required to make a diskette backup copy of a system, you need system directory information. If you are installing a system, this information is printed by step 6 of the system installation steps (see *System Installation Steps*, preceding). If you are not installing 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 and round to the next highest number if you have a remainder (this determines the number of active directory sectors).
3. Add the result of step 2 to the decimal number of active library member sectors (this determines the total library sectors referred to in the chart following step 4).



4. Use the result of step 3 and table 1 to determine the number of diskettes needed to hold your system. For extended format, use table 2.

**Table 1. Standard Format Diskette**

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 Diskette**

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

### APPLYPTF PROCEDURE

The APPLYPTF procedure applies PTFs to the library. It is called by the CNFIGSCP procedure, described following, or directly, by the APPLYPTF command statement.

PTFs applied by the APPLYPTF are read from a PTF diskette.

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

### APPLYPTF Command Statement Format

```
APPLYPTF [ SC1nn
           RG1nn
           UT1nn ] [ ,ALL
                   ,PTF log number ]
```

### APPLYPTF Parameters That are Not Prompted

- SC1nn PTFs that change the SCP are applied; nn is the version number (release number) of the system.
- RG1nn PTFs that change the RPG II program product are applied; nn is the version number (release number) of the product.
- UT1nn PTFs that change the IBM System/32 utilities program product (DFU/SEU/Sort) are applied; nn is the version number (release number) of the utility.

ALL Apply all PTFs from the selected PTF file.

PTF Apply only the PTF corresponding to the number given. This number  
log is the PTF log number and is indicated on the cover letter for each  
number PTF. It is also indicated in the PTFXREF source member on each  
PTF diskette.

### Prompted Parameters for APPLYPTF

#### *Translation Required?*

YES PTFs were applied to a translated (non-English) version of a system.  
NO PTFs were applied to a system that has not been translated.

#### *SCP PTFs Applied?*

If the response to the translation required prompt was yes, a prompt inquires whether the PTFs applied were SCP PTFs.

YES The PTFs applied were from the SC1nn file.  
NO The PTFs applied were not from the SC1nn file.

#### *\$MASPC Replaced?*

If the response to the SCP PTFs prompt was yes, a prompt inquires whether the load member \$MASPC was replaced. The PTF cover letter distributed with each PTF diskette lists the members that will be replaced by the PTFs on the diskette.

YES The PTFs applied replaced \$MASPC.  
NO The PTFs applied did not replace \$MASPC.

#### *#RDML Replaced*

If the response to the SCP PTFs prompt was yes, a prompt inquires whether the load member #RDML was replaced.

YES The PTFs applied replaced #RDML.  
NO The PTFs applied did not replace #RDML.

### 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 system once system configuration is complete. The CNFIGSCP procedure prompts the user for the print belt image, the date format, RPG II SCP support required, and the existence of a PTF diskette.

The CNFIGSCP procedure evokes the \$MAINT and \$SETCF utilities (see index entries: *\$MAINT utility program* and *\$SETCF utility program*).

## CNFIGSCP Command Statement Format

### CNFIGSCP

#### Prompted Parameters for CNFIGSCP

##### *Belt Image Option*

- 48 Sets the print belt image and its length in the system configuration record,
- 64 a record in the library directory that defines the system in terms of its components. A length of 48 or 64 can be specified.

##### *Date Format Option*

- YMD Sets the system date format in the system configuration record: year-month-day (YMD), month-day-year (MDY), or day-month-year (DMY).
- MDY
- DMY

##### *BSCA Support for SCP*

- YES Copies the SCP support for the BSCA.
- NO BSCA SCP support is not copied.

##### *RPG II SCP Support Option*

- YES Copies the optional SCP support for RPG II from the SCP PID diskette to the system.
- NO RPG II SCP support is not copied.

##### *BSCA Support Option*

- YES Copies the BSCA RPG support from SCP PID diskette to the system.
- NO BSCA support for RPG is not copied.

*Note:* The CNFIGSCP procedure will be changed to prompt you for BSC options (CLOCKING, TONE, and TEST) if you want SCP support for BSC.

##### *PTF Diskette Available*

- YES Prompts user to insert the PTF diskette and then issues the APPLYPTF command statement to copy SCP PTFs to the system (APPLYPTF SCP01, ALL).
- NO PTFs are not applied.

## INSTALL PROCEDURE

The INSTALL procedure allows the user to apply selected program products to the system and initialize the diskettes needed to back up the system. The INSTALL procedure prompts for the volume ID desired on the backup diskettes and whether the diskettes need to be initialized or not. The procedure then copies the tailored system onto the backup diskettes.

The INSTALL procedure evokes the following utilities: \$MAINT, \$INIT, and \$BACK (see index entries: *\$BACK utility program*, *\$INIT utility program*, and *\$MAINT utility program*).

### INSTALL Command Statement Format

INSTALL [DFU] [,SEU] [,SORT] [,RPG]

*Note:* These parameters are not positional and can be entered in any order desired.

### INSTALL Parameters that are Not Prompted

- DFU    The DFU (data file utility) function of the IBM System/32 utilities program product is installed as part of the system.
- SEU    The SEU (source entry utility) function of the IBM System/32 utilities program product is installed as part of the system.
- SORT    The sort function of the IBM System/32 utilities program product is installed as part of the system.
- RPG    The RPG II program product is installed as part of the system.

### Prompted Parameters for INSTALL

#### *Diskette Volume ID*

- Enter a name that has a maximum of six characters. This name will be placed in the VOLID field on the diskettes if diskettes need to be initialized. It will also be used as the VOLID parameter when the system is copied to the backup diskettes.

#### *Diskettes to be Initialized*

- YES        The diskette inserted will be initialized using the volume ID entered. (After each diskette is initialized, you will be prompted for any more diskettes to be initialized.)
- NO        No diskettes need to be initialized. The procedure will copy the system onto the diskettes that you have already initialized.

**PROGRAM PRODUCT INSTALLATION**

The IBM System/32 utilities program product, consisting of the data file utility, the source entry utility, and sort (DFU/SEU/Sort) is distributed on one diskette. The diskette has a volume identification of PPUTIL. IBM System/32 RPG II is distributed on two diskettes, each with a volume identification of RPGRPG. The method for installing these program products individually and creating a backup copy of each is described here.

**To Install a Program Product**

1. Insert the first (or only) program product diskette.
2. Enter the TOLIBR name command statement where *name* is the three- or four-character identifier of the major function being installed:

DFU	Data File Utility
SEU	Source Entry Utility
SORT	Sort
RPG	RPG II

*Note:* For a description of the TOLIBR procedure, see index entry: *TOLIBR procedure*.

3. Enter nameLOAD (that is, DFULOAD, SEULOAD, SORTLOAD, or RPGLOAD) for each function to be installed.
4. 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.

**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)

*Note:* You can use the INIT procedure to initialize diskettes—see index entry: *INIT procedure*.

2. Enter nameSAVE (that is, DFUSAVE, SEUSAVE, SORTSAVE, or RPGSAVE) for each function (DFU, SEU, Sort, or RPG II) to be saved.

## PROGRAM PRODUCT INSTALLATION VERIFICATION

You can verify the installation of SEU and RPG II.

### SEU Installation Verification

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 and 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) will be printed if SEU is properly installed.

```
THIS WILL VERIFY THAT SEU IS INSTALLED
```

8. Enter the following command statement to remove from the library the member created to verify the SEU installation:  
**REMOVE SEUTEST,SOURCE**

## 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 distribution diskette 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 for the following information:

1. Enter 123 for KEY prompt.
2. Enter DRESS for DESC prompt.
3. Enter 10 for VALUEA prompt.
4. Enter 30 for VALUEB prompt.
5. Enter 20 for VALUEC prompt.
6. Enter 124 for KEY prompt.
7. Enter COAT for DESC prompt.
8. Enter 40 for VALUEA prompt.
9. Enter 50 for VALUEB prompt.
10. Enter 30 for VALUEC prompt.

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, output of the five sample programs is:

1. NO TRANSACTIONS LOADED  
2 MASTERS LOADED

2. DATE IBM SYSTEM/32 SAMPLE UPDATE PROGRAM PAGE 0001  
KEY DESCRIPTION NEW VALUE A NEW VALUE B NEW VALUE C

NO TRANSACTION RECORDS ENTERED

3. DATE IBM SYSTEM/32 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

11243	JONES HARDWARE	27541021175	2375CASH	47	47	2328022175
11352	NU-STYLE CLOTHIERS	27987021475	8707CASH	174		4000022675
11886	MIDI FASHIONS INC	15771020475	10722CASH	214	214	10508021475
12874	ULOOK INTERIORS	25622020975	6795CASH	136		6795022375
18274	STREAMLINE PAPER INC	29703022175	27403	548	238	17055023075
23347	RITE-BEST PENS CO	20842021875	1580	31		1000022075
25521	IMPORTS OF NM	29273022075	79740	1593	1193	58547022775
26723	ALRIGHT CLEANERS	19473020775	46200CASH	924		46200022375
28622	NORTH CENTRAL SUPPLY	17816020575	7597CASH	152		7597022275
29871	FERGUSON DEALERS	27229021075	6191CASH	124		6191022275
30755	FASTWAY AIRLINES	26158020675	74272CASH	1495	1685	72587021975
31275	ENVIRONMENT CONCERNS	20451020675	2943	59		1500023075
32457B	SOLE SILOS	27425021075	11005CASH	220		11005022075
37945H	OFFTA BREAKS INC	18276020675	4723CASH	94		4723022375
42622	EASTLAKE GRAVEL CO	16429020575	2937CASH	58		2937022375

5.

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	2/25/75	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

Some components can be deleted from your system library to free up library space for other members or to free up disk space 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. Inquiry/offline support adds 11 library blocks on a 16K system, 14 library blocks on a 24K system, and 17 library blocks on a 32K system.

In addition to the minimum SCP requirements, the IBM System/32 program products, plus the optional BSCA functions, are shown in the following table:

Library Function	Directory Sectors		Library Blocks	
	Release 1	Release 2	Release 1	Release 2
DFU	3	3	36	36
SEU	4	4	37	37
SORT	5	5	34	34
RPG II				
SCP support	0	2	12	18
Program product	14	16	139	147
BSC				
SCP support <sup>1</sup>	0	1	0	3
RPG support <sup>1</sup>	0	2	0	4

<sup>1</sup> The library functions are broken out in the table since CNFIGSCP prompts the user for these separately.

## 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 \$PACK utility (see index entries: *\$PACK utility program* and *COMPRESS procedure*) to collect free disk space in one area.

### 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	O (load)	Level 1 error messages
##MSG4	O (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 with 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 detected, there will be no text when an error occurs.

In addition to deleting the preceding members you can delete inquiry/offline multi-volume 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:* To use available space after deleting members, do a BACKUP and RELOAD.

## 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 product functions are deleted by entering a **nameDROP** command statement for each function to be deleted—that is, by entering **DFUDROP**, **SEUDROP**, **SORTDROP**, and/or **RPGDROP**. The procedures evoked by these four 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 which you are not deleting.
2. To gather into one usable area the disk space created by deleting from the library, you must copy the library to diskette(s) and then reload the library. That is, **BACKUP** then **RELOAD**. See index entries: *BACKUP procedure* and *RELOAD procedure*.

## RELOAD DISPLAY

The **RELOAD** procedure (described under index entry: *RELOAD procedure*) is used to perform an IPL from diskettes on which the library has been copied using **BACKUP** (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 press the **LOAD** key when the IPL switch on the CE control panel is set to **DISKETTE**.

The **RELOAD** display shows the number of sectors allowed for the library directory, indicates whether or not inquiry or offline multivolume files are supported, and shows the total number of blocks allowed for the library (system file **#LIBRARY**). A sample display follows:

```
----> LIBRARY DIRECTORY SECTORS = 0033
INCLUDE INQUIRY/OFFLINE?       = NO
TOTAL LIBRARY BLOCKS           = 0239
```

### If Values in the RELOAD Display are Correct

If the values shown in the RELOAD display are correct for your system, press the ENTER key. The library is read from the diskette to the disk and the following display appears:

```
INSERT DISKETTE WITH FILE LABEL-#LIBRARY
      DATE- date , SEQUENCE NUMBER nn
----> PRESS ENTER KEY AFTER INSERTING
      WARNING-LIBRARY MAY BECOME UNUSABLE
      IF CORRECT VOLUME NOT INSERTED
```

The INSERT DISKETTE display always appears after a diskette has been read to the disk. When the display appears, remove the diskette and insert the next diskette as indicated. When all the diskettes have been 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 (date)
      LINES/PAGE nn
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:

- If the arrow is pointing at the first line (LIBRARY DIRECTORY SECTORS):

1. If there is no change to the line, press the REC ADV key. The arrow and cursor move to the second line.
2. If a change to this line should be made, 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 x 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. Data is read from the diskette onto the disk, and the insert diskette display appears.

- If the arrow is pointing at the second line (INCLUDE INQUIRY/OFFLINE?):

1. If there is no change to this line, press the REC ADV key. The arrow and cursor move to the third line.
2. If a change to this line should be made, enter the change (YES or NO) over the existing data and then press the REC ADV key. The arrow and cursor move to the third line.

*Note:* The inquiry/offline option requires a disk area in which to roll out an interrupted program or 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. Data is read from the diskette onto the disk, and the INSERT DISKETTE display appears.

- If the arrow is pointing at the third line (TOTAL LIBRARY BLOCKS):
  1. If there is no change to this line, press the REC ADV key. The arrow and cursor move to the first line.

2. If a change to this line should be made, 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 allow some space in the library for new members because of the inconvenience of expanding the library once remaining disk space is allocated to data files.

3. If all lines of the display are now correct, press the ENTER key. Data is read from the diskette onto the disk, and the insert diskette display will appear when the first diskette has been read.



**RECORDS TO BLOCKS CONVERSION FOR DISK – RECORDS GIVEN ON // FILE STATEMENT**

**Determining the Number of Sequential or Direct File Blocks**

Do the following to determine the number of blocks in a sequential or direct file.

1. *Multiply:* number of records x record length = number of characters.
2. *Divide:*  $\frac{\text{number of characters (from step 1)}}{\text{number of characters per block (2560)}} = \text{number of blocks (if there is a remainder, round to the next higher whole number)}$

**Determining the Number of Indexed File Blocks**

Do the following to determine the number of data blocks in an indexed file.

1. *Multiply:* number of records x record length = number of characters.
2. *Divide:*  $\frac{\text{number of characters (from step 1)}}{\text{number of characters per block (2560)}} = \text{number of data blocks (if there is a remainder, round to the next higher whole number)}$

Do the following to determine the number of index blocks in an indexed file.

1. *Add:* key field length + 3 = index entry length
2. *Divide:*  $\frac{\text{number of characters in a sector (256)}}{\text{index entry length (from step 1)}} = \text{number of entries per sector (drop fraction)}$
3. *Divide:*  $\frac{\text{number of records}}{\text{number of entries per sector (from step 2)}} = \text{number of sectors (if there is a remainder, round to the next higher whole number)}$
4. *Divide:*  $\frac{\text{number of sectors (from step 3) + 3}}{\text{number of sectors per block (10)}} = \text{number of index blocks (if there is a remainder, round to the next higher whole number)}$

To determine the total number of blocks required for an indexed file, add the number of data blocks required to the number of index blocks required.

## **DISK SECTOR NUMBER TO BLOCK NUMBER CONVERSION**

To convert sector number to block number, subtract 1 from the sector number, divide the result by 10, and drop the remainder. Examples:

10511 = sector number  
 $(10511 - 1) \div 10 = 1051.0$   
1051 = block number

10520 = sector number  
 $(10520 - 1) \div 10 = 1051.9$   
1051 = block number

## **DISK BLOCK NUMBER TO FIRST SECTOR IN BLOCK CONVERSION**

To find the first sector in a block, multiply the block number by 10 and add 1. Example:

1051 = block number  
 $(1051 \times 10) + 1 = 10511$   
10511 = first sector in block 1051

## Appendix B. Hex and Decimal Conversion

### Hex and Decimal Chart

Byte		Byte		Byte		Byte		Byte		Byte	
0123		4567		0123		4567		0123		4567	
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
6		5		4		3		2		1	
Position											

### Hex to Decimal Example

To find the decimal value of the hex value 1FA, you would find that in position 3 of the preceding hex and decimal chart, hex 1 equals decimal 256; that in position 2 of the chart, hex F equals 240; and that in position 1, hex A equals decimal 10. If you add these decimal values together, you have the decimal value of hex 1FA:  $256 + 240 + 10 = 506$ .

Hex	Dec	Hex	Dec	Hex	Dec
1	256	F	240	A	10
3		2		1	
Position					

### Decimal to Hex Example

To find the hex value of the decimal 534, you would find that the next lower decimal number in the preceding hex and decimal chart is 512 in position 3, equal to hex 2. You then subtract 512 from 534 and use the difference (22) to find the next hex value. The next lower decimal number in the chart is 16 in position 2, equal to hex 1. The difference between 22 and 16 is 6. The remaining 6 is found in position 1 of the chart, equal to hex 6. Therefore, the hex value of decimal 534 is 216.

Hex	Dec	Hex	Dec	Hex	Dec
2	512	1	16	6	6
3		2		1	
Position					

## 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 either in the standard interchange format or in the IBM System/32 extended format. The sectors in track 0 of both formats are 128 bytes long. Data sectors on standard interchange diskettes are also 128 bytes long. Data sectors on extended format diskettes are 512 bytes long. The INIT procedure and \$INIT system utility can initialize diskettes in either format (see index entries: *INIT procedure* and *\$INIT utility program*).

### DISKETTE DATA FILES

IBM System/32 creates and processes two kinds of diskette data files: standard interchange files and IBM System/32 system files.

#### Standard Interchange Files

Standard interchange files are described in *The IBM Diskette for Standard Data Interchange*, GA21-9182. In supporting standard data interchange, IBM System/32 permits the use of the following data set label fields in addition to the data set label fields defined for standard data interchange:

Field Name	Position in Data Set Label	Description
Volume sequence number	46-47	Volume sequence specifies the sequence of volumes in a multivolume file. The sequence must be consecutive, beginning with 01 (to a maximum of 99). Blanks indicate that volume sequence checking is not to be performed.
Creation date	48-53	Date the file was created. Format may be MMDDYY, YYMMDD, or DDMMYY, where MM is month (2 digits), DD is day (2 digits), and YY is year (2 digits).
Expiration date	67-72	Date the file can be purged.

Standard interchange files are created by the TRANSFER procedure and \$BICR system utility. The copy of a diskette file created by the COPY11 procedure or \$DUPRD utility is a standard interchange file if the original diskette file is a standard interchange file. (For a description of the procedures and utilities just mentioned, see index entries: *COPY11 procedure*, *TRANSFER procedure*, *\$BICR utility program*, and *\$DUPRD utility program*.)

Standard interchange files can reside only on diskettes initialized in the standard interchange format.

### System Files

System files differ from standard interchange files in that:

- System files can be recorded on track 74 of a diskette.
- The following fields of the data set labels for system files are defined as:

Field Name	Position	Contents
Offset to next record	18-22	A value indicating the starting position of the next sequential record relative to the EOD (end of data). If the field is blank (hex 40), the next record starts at the EOD address. Any other value in the field is a decimal value to be used as a negative displacement from EOD.
Record attribute	28	blank (hex 40)
		R
		Value                      Meaning
		blank (hex 40)              Records are unblocked and unspanned.
		R                              Records are blocked and spanned.
		Records that are unblocked and unspanned have a physical length of 128 or less. Records whose length is less than 128 are left-justified in sectors and padded to the right with binary zeros.
		Blocked records are recorded with no pad between them and spanned records can logically extend across sectors, tracks, and volumes (diskettes).
Physical record length	34	Contains a value indicating physical record (sector) length: blank (hex 40) indicates sector length of 128 bytes; 2 indicates sector length of 512 bytes.
Interchange type indicator	44	E—indicates system (noninterchange) file.

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 **COPY11** 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*, *COPY11 procedure*, *FROMLIBR procedure*, *ORGANIZE procedure*, *SAVE procedure*, *\$BACK utility program*, *\$COPY utility program*, *\$DUPRD utility program*, and *\$MAINT utility program*.)

System files can reside on both standard interchange and extended format diskettes. Position 76 of the volume label (VOL1) of a diskette contains a 2 if the diskette is in the extended format.





The procedures described here are IBM-provided service procedures designed to help you and IBM service personnel diagnose and correct system problems that may occur.

The following table, Figure 9, shows the formats of the command statements that evoke the service procedures. The table is intended for quick reference. Complete descriptions of the procedures and command statements follow the table.

The descriptions of the service procedures and related command statement formats provide the same kind of information as is provided by the procedure descriptions in Part 2. The use of brackets, capital letters, and so on to describe command statement formats is the same here as in Part 2.

For additional information, see the *IBM Systems Data Areas and Diagnostic Aids*, SY21-0532.

APAR vol-id, [object program name] [,source program name]

BUILD

DUMP 

MAIN
CONTROL
HISTORY
PTF
CONFIG
DISK

 , 

PRINTER
CRT

,F1
,I1

PATCH 

F1
I1

 [,NOHEX]

TRACE 

ALL
OFF

 [,WAIT] [,FDIOS] [,CSFDIOS] [,PUSH] [,PULL] [,DISABLE]  
[,ENABLE] [,QUEUE] [,LDCS] [,LOADER] [,XIENT]

Figure 9. Command Statement Formats for IBM Service Procedures

## APAR PROCEDURE

The APAR procedure collects diagnostic information that can help IBM service personnel isolate and correct programming problems that may occur in the system. The APAR procedure collects the information in two files, APARFILE and FIXDFILE. The APAR procedure creates the two files on a diskette.

Recorded in APARFILE is the information saved on the CE cylinder at the time of a program check interrupt (programming problem). The CE cylinder is two tracks reserved on the fixed disk for diagnostic information. The information that exists on the CE cylinder after a program check consists of:

*Note:* Beware of customer security requirements.

- Contents of control storage and the first 16K of main storage at the time of the program check. (Main storage beyond 16K is stored in APARFILE, but is not on the CE cylinder – it is written within the library.)
- Last 20 sectors of the HISTORY file.

The APAR procedure also records:

- The PTF (program temporary fix) log module (in APARFILE)
- The system configuration record (in APARFILE)
- The scheduler work area (in FIXDFILE)
- The disk VTOC (in FIXDFILE)
- The volume label error logging tables and library control sector from the disk (in FIXDFILE)
- The rollin/rollout area (in FIXDFILE)

The APAR procedure evokes the \$FEAPR utility program.

### APAR Command Statement Format

APAR vol-id, [object program name] [,source program name]

### 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 after an error has occurred 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 has occurred.

### 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 as a result of a program check interrupt or may have been saved because the RESET and CE START keys 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) may be displayed or printed (see index entry: *APAR procedure*).

The DUMP procedure evokes the \$FEDMP utility program.

### DUMP Command Statement Format

DUMP [MAIN  
CONTROL  
HISTORY  
PTF  
CONFIG  
DISK] , [PRINTER  
CRT] [,F1  
,I1]

If MAIN, CONTROL, HISTORY, PTF, or CONFIG are specified with I1, the specified items are printed or displayed from a diskette file created by the APAR command.

## DUMP Parameters

<u>MAIN</u>	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 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 a default value.
CONTROL	The 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	Area of F1 or I1 can be dumped. Prompts will be issued for the sector number and number of sectors to be dumped. At the completion of that dump, prompts are issued for the next group of sectors. You can respond with the limits or END.
<u>PRINTER</u>	Output is on the printer. PRINTER is a default value.
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.
<u>F1</u>	The disk contains the information requested by the MAIN, CONTROL, HISTORY, PTF, CONFIG, or DISK parameter. F1 is a default value.
I1	If disk is specified, specified sectors from the diskette may be displayed or printed.

## PATCH PROCEDURE

The PATCH procedure enables IBM service personnel to patch, or modify, a disk or diskette sector by keying modifications from the keyboard.

The PATCH procedure evokes the \$FEPCH utility program.

## PATCH Command Statement Format

PATCH  $\left[ \begin{array}{c} \text{F1} \\ \text{I1} \end{array} \right]$  [,NOHEX]

If MAIN, CONTROL, HISTORY, PTF, or CONFIG are specified, the specified items are printed or displayed from a diskette created by the APAR command.

## PATCH Parameters

- F1** A disk sector is to be patched. (F1 is a default value.) A prompt requesting the sector number is displayed. That sector is displayed on the display screen, 40 characters at a time. The keyboard function keys can be used to display the entire sector. The first line of the display contains printable EBCDIC characters. If the NOHEX parameter is specified, the hex representation of only unprintable characters is displayed on lines 2 and 3. If NOHEX is not specified, the hex representation of all characters in the sector is displayed on lines 2 and 3. Line 4 is used to display the displacement into the record (COL=nnnnn) and the sector number (SS=nnnn).
- The patch data is entered from the keyboard as the affected portion(s) of the sector is displayed. After all changes are made to a sector, the sector is written back to the disk by pressing REC ADV. The next sequential sector is then displayed. Another sector can be displayed by pressing ENTER. Then a prompt will be given. To end job, enter END in response to prompt.
- I1** A diskette sector is to be patched. The method for patching a diskette sector is the same as for patching a disk sector. See the preceding description of the F1 parameter.
- NOHEX** The hex representations of only unprintable characters are to be 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—part of the SCP—is issued, its value, or function, is checked. If the function is one of those for which a trace has been 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 program check interrupt occurred or because the RESET and CE START keys were pressed, 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

Information provided by the trace includes request indicator byte 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 ] [WAIT,] [FDIOS,] [CSFDIOS,] [PUSH,] [PULL,] [DISABLE,]
      [ENABLE,] [QUEUE,] [LDCS,] [LOADER,] [XIENT]
```

*Note:* If OFF is specified, OFF must be the first parameter. If either ALL or OFF is specified, all other parameters specified are ignored. The remaining parameters can be specified in any order. A maximum of ten parameters can be specified. However, the entire SCP trace function is disabled if DEBUG-Y is specified in an ALTERBSC command statement or \$SETCF SETB utility control statement (see index entries: *ALTERBSC procedure* and *BSCA environment*).

## TRACE Parameters

<u>ALL</u>	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.





This appendix shows the OCL and utility control statements contained in each IBM procedure. 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 } ] [ ,CLOCK- { Y } ] [ ,DEBUG- { Y } ] [ ,ERC- { number } ]
           [ ,SLINE- { Y } ] [ ,TEST- { Y } ] [ ,TONE- { Y } ]
// 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]
```

**APPLYPTF**

```
// LOAD $MAINT
// FILE NAME- { SC1nn }
                { RG1nn } ,UNIT-I1
                { UT1nn }
// RUN
```

If the 2nd parameter is ALL:

```
// COPY FROM-DISK,TO-F1,FILE- { SC1nn }
                                { RG1nn } ,RETAIN-R
                                { UT1nn }
```

If the 2nd parameter is "ptfid":

```
// COPY FROM-DISK,TO-F1,FILE- { SC1nn }
                                { RG1nn } ,PTF-ptfid,RETAIN-R
                                { UT1nn }
// END
```

If the language translation of SCP PTFs is requested, these additional OCL statements are generated:

```
// LOAD $MAINT
// FILE NAME-TRANS,UNIT-I1
// FILE NAME-SCRTAB,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,FILE-TRANS,RETAIN-R
// COPY FROM-DISK,TO-F1,FILE-SRCTAB,RETAIN-R
// END
// LOAD $TRANS
// RUN
// TRANSLATE SOURCE-SRCTAB
// END
// LOAD $MAINT
// RUN
// DELETE LIBRARY-O,NAME-$TRANS
// DELETE LIBRARY-O,NAME-$STRATAB
// DELETE LIBRARY-O,NAME-TRNSMSG1
// END
```

#### BACKUP

```
// LOAD $BACK
// FILE NAME-#LIBRARY,LABEL- $\left\{ \begin{array}{l} \text{filename} \\ \underline{\#LIBRARY} \end{array} \right\}$ ,RETAIN- $\left\{ \begin{array}{l} \text{retention-days} \\ \underline{1} \end{array} \right\}$ ,
PACK-vol-id,UNIT-I1
// RUN
```

#### BUILD

```
// LOAD $BUILD
// RUN
```

#### CATALOG

```
// LOAD $LABEL
// RUN
// DISPLAY UNIT- $\left\{ \begin{array}{l} \underline{I1} \\ \underline{F1} \end{array} \right\}$ ,LABEL- $\left\{ \begin{array}{l} \text{filename} \\ \underline{ALL} \end{array} \right\}$ 
// END
```

#### CNFIGSCP

Set belt option:

```
// LOAD $SETCF
// IMAGE MEM, $\left\{ \begin{array}{l} \underline{BELT48} \\ \underline{BELT64} \end{array} \right\}$ 
// RUN
// SETCF IMAGE-YES
// END
```

Set date format option:

```
// LOAD $SETCF
// RUN
// SETCF FORMAT- { YMD }
                  { MDY }
                  { DMY }
// END
```

Load SCP support for BSCA (the procedure name is CNFIBSCA):

```
// LOAD $MAINT
// FILE NAME-BSCALDAD,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-BSCALOAD
// END
```

Load SCP support for RPGII (the procedure name is CNFIRG1):

If you want BSCA support for RPG, use the following procedure:

```
// LOAD $MAINT
// FILE NAME-RPGSUBR,UNIT-I1
// FILE NAME-RPGLINK,UNIT-I1
// FILE NAME-BSCASUBR,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-BSCASUBR
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-RPGSUBR
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-RPGLINK
// END
```

If you do not want BSCA support for RPG, use the following procedure:

```
// LOAD
// FILE NAME-RPGSUBR,UNIT-I1
// FILE NAME-RPGLINK,UNIT-I1
// RUN
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-RPGSUBR
// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-RPGLINK
// END
```

Apply PTFs to SCP and optional programs (the 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
```

## COPY11

```
// LOAD $DUPRD  
// FILE NAME-COPY11 [,DATE-date],UNIT-11  
// RUN  
// COPY11 NAME- {filename} ,PACK-vol-id [ ,DELETE- {YES} ]  
                { ALL }  
// END
```

## CREATE

```
// LOAD $MGBLD  
// RUN  
// MGBLD SOURCE-sourcename,REPLACE- {YES}  
                                     { NO }  
// END
```

## DATE

```
// DATE-date
```

## DELETE

```
// LOAD $DELET  
// RUN  
// SCRATCH LABEL-filename [,DATE-date] ,UNIT- { F1 }  
                                                { 11 }  
and/or  
// REMOVE LABEL-filename,DATA- { YES }  
                                { NO } [,DATE-date] ,UNIT- { F1 }  
// END                                                                    { 11 }
```

## 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 } ] [ ,OUTPUT- { PRINTER
                                      CRT } ] [ ,INPUT- { F1
                                                         I1 } ]
// END
```

## FROMLIBR

```
// LOAD $MAINT
// FILE NAME- [ filename-1
               library-name-1
               filename-2
               name-1 ] ,RETAIN- [ P
                                   S
                                   I
                                   retention-days ] [ If F1 is
                                                         specified. ] [ ,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]]
// END
// LOAD $HINT
// RUN
```

## INIT

```
// LOAD $INIT
// RUN
// UIN OPTION- { FORMAT
                { FORMAT2
                { DELETE
                { RENAME
[ // VOL PACK- { vol-id
               { system date } ,ID- { owner-id
                                     { OWNER-ID }
// END
```

## 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
```

## LINES

```
// FORMS LINES- { number }
                { 66 }
```

## LISTLIBR

```
// LOAD $MAINT
// RUN
// COPY FROM-F1,NAME- { DIR
                     { library-name
                     { name.ALL
                     { ALL } ,LIBRARY- { S
                                         { P
                                         { O
                                         { R
                                         { ALL
                                         { SYSTEM } ,TO-PRINT
// END
```

## LOG

```
// LOG { PRINTER } { ,EJECT }  
      { CRT } { ,NOEJECT }
```

## ORGANIZE

```
// LOAD $COPY  
// FILE NAME-COPYIN,LABEL-filename-1 [,DATE-date] ,UNIT-F1  
// FILE NAME-COPYO,LABEL-filename-2,RETAIN- { P }  
                                           { S } ,UNIT-F1  
                                           { I }  
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 } ]  
// END
```

## PATCH

```
// LOAD $FEPCH  
// RUN  
// PATCH INPUT- { F1 } ,HEX- { NO }  
                  { I1 } { 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
```

## 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 } ] [ ,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
```



## STATUS

```
// LOAD $STATS
// RUN
```

## SYSLIST

```
// SYSLIST 

|         |
|---------|
| PRINTER |
| CRT     |
| OFF     |


```

## TOLIBR

```
// LOAD $MAINT
// FILE NAME-filename [,DATE-date] ,UNIT- { F1 }
// RUN
// COPY FROM-DISK,FILE-filename,RETAIN- { P }
// END
```

## TRACE

```
// LOAD $SETCF
// RUN
// TRACE 

|                |               |                |             |               |             |
|----------------|---------------|----------------|-------------|---------------|-------------|
| ALL-Y,         |               |                |             |               |             |
| { WAIT-N, }    | [ FDIOS-Y, ]  | [ CSFDIOS-Y, ] | [ PUSH-Y, ] | [ PULL-Y, ]   |             |
| { WAIT-Y, }    | [ FDIOS-N ]   | [ -N ]         | [ -N ]      | [ -N ]        |             |
| [ DISABLE-Y, ] | [ ENABLE-Y, ] | [ QUEUE-Y, ]   | [ LDOS-Y, ] | [ LOADER-Y, ] | [ XIENT-Y ] |
| [ -N ]         | [ -N ]        | [ -N ]         | [ -N ]      | [ -N ]        | [ -N ]      |


// END
```

## TRANSFER

```
// LOAD $BICR
// FILE NAME-COPYIN,LABEL-filename-1 [,DATE-date] ,UNIT- { F1 }
// RUN
```

Transfer disk to diskette:

```
// FILE NAME-COPYO,LABEL-filename-1,PACK-vol-id [ ,retention-days ]
UNIT-I1 or
```

Transfer diskette to disk, with ADD:

```
// FILE NAME-COPYO,LABEL- { filename-2 }
// FILE NAME-COPYO,LABEL- { filename-1 } [ ,DATE-date ] ,UNIT-F1 or
```

Transfer diskette to disk, without ADD, size specified:

```
// FILE NAME-COPYO,LABEL-filename-1 { ,RECORDS-value-3 }
// FILE NAME-COPYO,LABEL-filename-1 { ,BLOCKS-value-4 } ,UNIT-F1
```

## **TRANSFER (continued)**

Transfer diskette to disk, without ADD, using size of input file:

No COPYO FILE statement is generated.

// RUN

Diskette standard interchange file to disk sequential file, or disk sequential, indexed, or direct file to diskette standard interchange file:

[// TRANSFER]

Diskette standard interchange file to disk sequential file with ADD:

// TRANSFER ADD-YES

Diskette standard interchange file to disk indexed file, without ADD:

// TRANSFER ADD-NO,KEYLEN-value-1,KEYLOC-value-2

// END

The following characters are from the standard 64-character print belt. They include the characters on the standard 48-character print belt.

Character	Hexadecimal Equivalent
Blank (not represented on the print belt)	40
¢	4A
.	4B
<	4C
(	4D
+	4E
	4F
&	50
!	5A
\$	5B
*	5C
)	5D
;	5E
⌋	5F
- (minus)	60
/	61
,	6B
%	6C
_ (underscore)	6D
>	6E
?	6F

<b>Character</b>	<b>Hexadecimal Equivalent</b>
' (grave accent)	79
:	7A
#	7B
@	7C
' (single quote)	7D
=	7E
"	7F
A	C1
B	C2
C	C3
D	C4
E	C5
F	C6
G	C7
H	C8
I	C9
J	D1
K	D2
L	D3
M	D4
N	D5
O	D6
P	D7
Q	D8
R	D9

<b>Character</b>	<b>Hexadecimal Equivalent</b>
\	E0
S	E2
T	E3
U	E4
V	E5
W	E6
X	E7
Y	E8
Z	E9
0	F0
1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	F9

**Note:** This character set is shown in ascending sorting sequence; that is, the blank character is the lowest—will be sorted before any other character—and the 9 character is the highest.



## Appendix G. Polling and Addressing Characters for System/32 Tributary Stations

Polling and addressing characters must be used together in certain pairs; that is, once a polling character is selected, the complementary addressing character is determined; once an addressing character is selected, the complementary polling character is determined.

The pairs of valid polling and addressing characters for both EBCDIC and ASCII codes are as follows:

### EBCDIC

<b>Polling Character</b>	<b>Hexadecimal Representation</b>	<b>Addressing Character</b>	<b>Hexadecimal Representation</b>
BB	C2C2	SS	E2E2
CC	C3C3	TT	E3E3
DD	C4C4	UU	E4E4
EE	C5C5	VV	E5E5
FF	C6C6	WW	E6E6
GG	C7C7	XX	E7E7
HH	C8C8	YY	E8E8
II	C9C9	ZZ	E9E9
JJ	D1D1	11	F1F1
KK	D2D2	22	F2F2
LL	D3D3	33	F3F3
MM	D4D4	44	F4F4
NN	D5D5	55	F5F5
OO	D6D6	66	F6F6
PP	D7D7	77	F7F7
QQ	D8D8	88	F8F8
RR	D9D9	99	F9F9

**ASCII**

<b>Polling Character</b>	<b>Hexadecimal Representation</b>	<b>Addressing Character</b>	<b>Hexadecimal Representation</b>
AA	4141	aa	6161
BB	4242	bb	6262
CC	4343	cc	6363
DD	4444	dd	6464
EE	4545	ee	6565
FF	4646	ff	6666
GG	4747	gg	6767
HH	4848	hh	6868
II	4949	ii	6969
JJ	4A4A	jj	6A6A
KK ,	4B4B	kk	6B6B
LL ,	4C4C	ll	6C6C
MM	4D4D	mm	6D6D
NN	4E4E	nn	6E6E
OO	4F4F	oo	6F6F
PP	5050	pp	7070
QQ	5151	qq	7171
RR	5252	rr	7272
SS	5353	ss	7373
TT	5454	tt	7474
UU	5555	uu	7575
VV	5656	vv	7676
WW	5757	ww	7777
XX	5858	xx	7878
YY	5959	yy	7979
ZZ	5A5A	zz	7A7A

To specify the addressing characters in the ADDR-nn parameter of the SETR utility control statement or the OVERRIDE command statement format, give the hex representation of one of the characters. It will be duplicated by the system to provide the two addressing characters.

For example, ADDR-F7 is given to specify the EBCDIC addressing characters 77 which correspond to EBCDIC polling characters PP. ADDR-70 is given to specify the ASCII addressing characters pp which correspond to ASCII polling characters PP.



## Appendix H. ASCII and EBCDIC Codes

The coded character sets for ASCII (American National Standard Code for Information Interchange) and EBCDIC (extended binary coded decimal interchange code) are shown in the following tables. Use the set that your programming system supports.

EBCDIC Codes

Main Storage Bit Positions 4, 5, 6, 7		Main Storage Bit Positions 0, 1, 2, 3															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0	NUL	DLE	DS		SP	&	-					{	}	\	0	
0001	1	SOH	DC1	SOS					a	j	~		A	J		1	
0010	2	STX	DC2	FS	SYN				b	k	s		B	K	S	2	
0011	3	ETX	DC3						c	l	t		C	L	T	3	
0100	4	PF	RES	BYP	PN				d	m	u		D	M	U	4	
0101	5	HT	NL	LF	RS				e	n	v		E	N	V	5	
0110	6	LC	BS	EOB ETB	UC				f	o	w		F	O	W	6	
0111	7	DEL	IL	PRE ESC	EOT				g	p	x		G	P	X	7	
1000	8		CAN						h	q	y		H	Q	Y	8	
1001	9	RLF	EM					\	i	r	z		I	R	Z	9	
1010	A	SMM	CC	SM		d	!		:								
1011	B	VT				.	\$	,	#								
1100	C	FF	IFS		DC4	<	*	%	@								
1101	D	CR	IGS	ENQ	NAK	(	)	_	'								
1110	E	SO	IRS	ACK		+	;	>	=								
1111	F	SI	IUS	BEL	SUB		⌋	?	"								



Duplicate Assignment

ASCII Codes

Main Storage Bit Positions 4, 5, 6, 7		Main Storage Bit Positions 0, 1, 2, 3															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0	NUL	DLE	SP	0	@	P	'	p								
0001	1	SOH	DC1	!	1	A	Q	a	q								
0010	2	STX	DC2	"	2	B	R	b	r								
0011	3	ETX	DC3	#	3	C	S	c	s								
0100	4	EOT	DC4	\$	4	D	T	d	t								
0101	5	ENQ	NAK	%	5	E	U	e	u								
0110	6	ACK	SYN	&	6	F	V	f	v								
0111	7	BEL	ETB	'	7	G	W	g	w								
1000	8	BS	CAN	(	8	H	X	h	x								
1001	9	HT	EM	)	9	I	Y	i	y								
1010	A	LF	SUB	*	:	J	Z	j	z								
1011	B	VT	ESC	+	;	K	[	k	{								
1100	C	FF	FS	,	<	L	\	l									
1101	D	CR	GS	-	=	M	]	m	}								
1110	E	SO	RS	.	>	N	⌋	n	~								
1111	F	SI	US	/	?	O	_	o	DEL								



## Appendix I. Data Link Control Characters

The following characters and character sequences are recognized by System/32 BSCA. For detailed information on data link control characters, see *General Information – Binary Synchronous Communications, GA27-3004*.

Name	Mnemonic	ASCII Code	EBCDIC Code
Start of heading	SOH	SOH	SOH
Start of text	STX	STX	STX
End of transmission block	ETB	ETB	ETB
End of text	ETX	ETX	ETX
End of transmission	EOT	EOT	EOT
Enquiry	ENQ	ENQ	ENQ
Negative acknowledge	NAK	NAK	NAK
Synchronous idle	SYN	SYN	SYN
Data link escape	DLE	DLE	DLE
Intermediate block character	ITB	US	IUS
Even acknowledge	ACK 0	DLE 0	DLE (70)
Odd acknowledge	ACK 1	DLE 1	DLE/
Wait before transmit – Pos. Ack.	WACK	DLE;	DLE,
Mandatory disconnect	DISC	DLE EOT	DLE EOT
Reverse interrupt	RVI	DLE <	DLE@
Temporary text delay	TTD	STX ENQ	STX ENQ
Transparent start of text	XSTX		DLE STX
Transparent intermediate block	XITB		DLE IUS
Transparent end of text	XETX		DLE ETX
Transparent end of trans. block	XETB		DLE ETB
Transparent synchronous idle	XSYN		DLE SYN
Transparent block cancel	XENQ		DLE ENQ
Transparent TTD	XTTD		DLE STX DLE ENQ
Date DLE in transparent mode	XDLE		DLE DLE



IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the *American National Standard Vocabulary for Information Processing* (Copyright © 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3.

ANSI definitions are identified by an asterisk. An asterisk to the right of the term indicates that the entire entry is reprinted from the *American National Standard Vocabulary for Information Processing*; where definitions from other sources are included in the entry, ANSI definitions are identified by an asterisk to the right of the item number.

**alternate sector:** A sector on disk assigned by the system in place of a sector that has become unusable.

**attribute bytes:** A 2-byte field called attributes in the library directory that contains the characteristics of a library member.

**block:** A unit of space assignment for files on the disk. One block is equal to 10 sectors.

**BSCA (binary synchronous communications adapter):** A flexible form of line control which provides a set of rules for communications between two devices.

**CE cylinder:** A cylinder on disk for use as a save area for the dump of main storage, control storage, and the history file. Also used as a read/write area for CE diagnostics.

**command statement:** A statement that is used to request the performance of a particular function. It always contains the command name and may include parameters. Specifically, a command statement is a special form of the // INCLUDE OCL statement. A command statement evokes a procedure and can pass information to the procedure via parameters included in the statement. The procedure named by the command name is evoked by the command statement.

**comparison parameter:** A parameter that is used in a test that compares one value to another. A comparison test is performed within a procedure; the next action performed by the procedure usually depends on the result of the test.

**condition parameter:** See *comparison parameter* and *existence testing parameter*.

**conditional expression:** A conditional expression is used among OCL statements to modify procedures. Valid conditional expressions for IBM System/32 are IF and ELSE.

**configuration record:** See *system configuration record*.

**continuation:** The use of more than one record to contain a single OCL statement. Continuation is valid only for FILE OCL statements.

**default value:** A value assigned by the system when no value is selected by the user.

**directory:** See *library directory*.

**directory area:** The area on the disk that contains the library directory.

**disk block:** See *block*.

**error logging area:** A variable number of sectors on disk used for recording hardware, and hardware-related system errors.

**existence testing parameter:** A parameter that is used in a test performed within a procedure that tests for the existence of a specified file or library member, or tests the setting of an external indicator.

**file segment:** The portion of an offline multivolume file that can reside on the disk. A segment consists of the information stored on one diskette.

**format 1:** A record containing system information that describes a file. There is one format 1 record for each file on the disk. Format 1's are in the volume table of contents (VTOC).

**identifier:** The group of characters that distinguishes one control statement from another. For example, SWITCH is the identifier in a // SWITCH OCL statement.

**indicator settings parameter:** The parameter on the SWITCH OCL statement that defines the setting of the external indicators.

**initial program load (IPL):** The loading of the system control programming into main storage. This prepares the system for execution of jobs.

**initialize:** To use the INIT procedure or \$INIT utility program to prepare a diskette for use by naming the diskette, removing the entries from the VTOC, and formatting the diskette.

**inquiry option:** The system function that processes inquiry requests.

**inquiry request:** A request by the operator (made by pressing the INQ key) that stops the job that is running, if it is an interruptable program so that another program or function can be performed.

**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*.

**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.

**null entry:** An entry that contains no value. For example, if CATALOG,11 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.

**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 possibly, utility control statements stored in the library.

**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 directory (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.

**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*.

**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.

**subroutine member:** A subroutine that needs to be link edited (joined) before being loaded for execution. Subroutine members are stored in the library.

**symbolic parameter:** A parameter that does not contain a keyword. For example, // LOAD \$COPY. \$COPY is a symbolic parameter. See also *keyword parameter*.

**system configuration record:** Information stored in the library directory that describes the system and its programming support.

**system library:** See *library*.

**system utility programs:** A set of programs provided with the system that are used to perform the everyday routine tasks required by a data processing system.

**utility control statement:** A control statement that gives a utility program information concerning the output you want the program to produce or the way you want the program to perform its function.

**utility program:** See *system utility programs*.

**value:** For OCL, the user entry in response to a keyword parameter — for example, UNIT-F1. F1 is a value. If no value is entered by the user, the system control programming may assign a default value.

**volume label (VOL1):** A one-sector area on the disk or diskette containing user and system control programming information for the disk.

**volume table of contents (VTOC):** A table of contents on the disk or diskette that describes each file on the disk or diskette.

**VTOC:** See *volume table of contents*.

- \$BACK utility program (backup library) 116
- \$BICR utility program (standard interchange)
  - control statements 117, 118
  - description 117, 118
  - example 118
- \$BUILD utility program (alternate sector rebuild)
  - control statements 121
  - description 119-121
  - example 120
- \$COPY utility program (disk copy/display)
  - control statements 122-130
  - description 122-131
  - examples 131
- \$DELET utility program (file delete)
  - control statements 132-134
  - description 132-135
  - examples 134, 135
- \$DUPRD utility program (diskette copy)
  - control statements 136, 137
  - description 136-138
  - examples 138
- \$HIST utility program (HISTORY file display)
  - control statements 139
  - description 139, 140
  - examples 140
- \$INIT utility program (diskette labeling and initialization)
  - control statements 142-145
  - description 141-145
  - examples 145
- \$LABEL utility program (VTOC display)
  - control statements 149
  - description 146-149
  - examples 146-149
- \$LOAD utility program (reload library)
  - control statements 153
  - description 150
  - example 150
- \$LOADI program 150
- \$MAINT utility program (library maintenance)
  - allocate function
    - control statements 155, 156
    - description 155, 156
    - examples 156
  - copy function
    - control statements 158-166
    - description 156-172
    - examples 166-172
  - delete function
    - control statements 173, 174
    - description 172-175
    - examples 175
  - general description 153-155
- \$MGBLD utility program (create message member)
  - control statements 176, 177, 178
  - description 176-178
  - example 178
- \$PACK utility program (disk reorganization) 179
- \$REBLD utility program (rebuild data file)
  - control statements 181
  - description 180, 181
- \$SETCF utility program (set)
  - OVERRIDE RPG BSCA specifications 184
    - control statements 184
    - description 181, 184
    - example 185
  - set BSCA environment 183
    - control statements 183
    - description 181, 183
    - example 184
  - set functions to be traced
    - control statements 186, 187
    - description 181, 182, 185-187
    - example 187
  - set system environment
    - control statements 181, 182
    - description 181, 182
    - example 182
- \$SOURCE file 98
- \$STATS utility program (status display) 188
- \$WORK file 98
- \* (comment) statement (see also comments)
  - description 33
  - statement summary 11
- /\* (end of data) statement
  - description 34
  - statement summary 11
- //\* (message) statement
  - description 34
  - example 34
  - parameter summary 14
  - statement summary 11
- // COMPILE statement
  - description 15
  - example 15
  - parameter summary 12
  - statement summary 10
- // DATE statement
  - description 16
  - example 16
  - parameter summary 12
  - statement summary 10
- // FILE statement
  - description
    - disk 17-20
    - diskette 17, 20-22
  - example
    - disk 20
    - diskette 22
  - parameter summary
    - disk 12
    - diskette 12-13
  - statement summary 10

**// FORMS statement**  
description 23  
example 23  
parameter summary 13  
statement summary 10  
**// IMAGE statement**  
description 24-25  
examples 25  
parameter summary 13  
statement summary 10  
**// INCLUDE statement**  
as a command statement 39  
description 26-27  
example 26-27  
parameter summary 13  
statement summary 10  
**// LOAD statement**  
description 27  
example 27  
parameter summary 13  
statement summary 10  
**// LOG statement**  
description 28  
example 28  
parameter summary 13  
statement summary 10  
**// MEMBER statement**  
description 29-30  
examples 30  
parameter summary 13-14  
statement summary 10  
**// PAUSE statement**  
description 30  
statement summary 11  
**// RUN statement**  
description 31  
statement summary 11  
**// SWITCH statement**  
description 32  
example 32  
parameter summary 14  
statement summary 11  
**// SYSLIST statement**  
description 33  
example 33  
parameter summary 14  
statement summary 11  
? restrictions  
in comment statements 33  
in file names 17, 21  
?n? 42-43, 44  
?n'default'? 43, 44  
?nR? 43, 44  
?nR'msg-id'? 43, 44  
?nT'default'? 43, 44  
?R? 43, 44  
##MSG1 210, 211  
##MSG3 40  
##MSG4 210, 211  
#LIBRARY (see system library)

abbreviations viii  
add  
a disk file to a diskette 82, 122  
a disk file to the library 89  
(see also \$MAINT utility program copy function)  
a standard interchange file to a disk file 91, 117  
library members to a file 65  
(see also \$MAINT utility program copy function)  
library members to the library 89  
(see also \$MAINT utility program copy function)  
allocate function (see \$MAINT utility program allocate function)  
ALTERBSC command statement  
description 56  
format summary 49  
ALTERBSC procedure  
contents 231  
description 56  
alternate sector 119, 253  
alternate sector rebuild utility program (see \$BUILD utility program)  
an example of creating a message source and load member 178  
APAR command statement  
description 224  
format summary 223  
APAR procedure  
contents 231  
description 224  
APARFILE 224, 225  
application programs 197  
APPLYPTF command statement  
description 199, 200  
format summary 191  
APPLYPTF procedure  
contents 231  
description 199, 200  
ASCII codes 249  
attribute bytes 168-169, 253

backup  
program products 203  
system library 57, 116  
BACKUP command statement  
description 57  
format summary 51  
backup diskettes  
creating 59, 136  
program products 203  
system 195, 198  
backup library utility program (see \$BACK utility program)  
BACKUP procedure  
contents 232  
description 57  
belt image option 201  
(see also print belt)  
block 253  
block number to first sector in block conversion 216

BSCA 253  
 BSCA environment 183  
 BSCA library requirements 209  
 BSCA support 201  
 BUILD command statement  
   description 225  
   format summary 223  
 BUILD procedure  
   contents 232  
   description 225  
 bypass unreadable data 120, 121

calculating the number of backup diskettes required  
 for the system 198, 199  
 CATALOG command statement  
   description 58  
   format summary 51  
 CATALOG procedure  
   contents 232  
   description 58  
 CE cylinder 224, 225  
 changing  
   directory and library size  
     using \$MAINT allocate function 155  
     using RELOAD display 211-214  
     using RELOAD procedure 79  
   disk space allocation 81  
 characters, list of 241-243  
 CMD key 39  
 CNFIGSCP command statement  
   description 201  
   format summary 191  
 CNFIGSCP procedure 200, 201  
 coding rules  
   general 6-7  
   OCL statements 5-6  
   utility control statements 113  
 command key messages 40  
 command key request 39-40  
 command statements (see also ALTERBSC, APAR, APPLYPTF,  
 BACKUP, BUILD, CATALOG, CNFIGSCP, COMPRESS,  
 COPY11, CREATE, DATE, DELETE, DISPLAY, DUMP,  
 FROMLIBR, HISTORY, INIT, INSTALL, LINES,  
 LISTLIBR, LOG, ORGANIZE, OVERRIDE, PATCH,  
 REBUILD, RELOAD, REMOVE, RESTORE, SAVE, SET,  
 STATUS, SYSLIST, TOLIBR, TRACE, TRANSFER)  
   as INCLUDE statements 26, 39  
   definition 253  
   in sample jobs 107-109  
   tables of  
     SCP 51-54  
     service 223  
   system configuration, installation, and modification 191  
 comments (see also \* (comment) statement)  
   definition 6  
   examples 7  
   for messages 178  
   rules for using 7

comparison parameter 45, 253  
 COMPILE OCL statement (see // COMPILE statement)  
 COMPRESS command statement  
   description 58  
   format summary 51  
 COMPRESS procedure  
   contents 234  
   description 59  
 condition parameter 44-47  
 conditional expressions: IF and ELSE 44-47, 253  
 configuration record, system 85, 255  
 configuration, system 4, 189-213  
 continuation 7-8, 253  
 continued FILE statements 7-8  
 control statement for message source member 177  
 control storage dump 225, 226  
 conventions for describing  
   command statements 55  
   OCL statements 9  
   utility control statements 114  
 conversions  
   block number to first sector in block 216  
   hex and decimal 217-218  
   records to blocks 216  
   sector number to block number 216  
 convert a  
   disk file to standard interchange file 91, 117  
   standard interchange diskette file to disk file 91, 117  
 copy function (see \$BACK utility program; \$COPY utility  
 program; \$DUPRD utility program; \$MAINT utility  
 program copy function)  
 COPY11 command statement  
   description 59, 60  
   example 60  
   format summary 51  
 COPY11 procedure  
   contents 234  
   description 59-60  
 correct unreadable data 120, 121  
 CREATE command statement  
   description 61  
   example 61  
   format summary 51  
 create message member utility program (see \$MGBLD  
 utility program)  
 CREATE procedure  
   contents 234  
   description 61  
 creating another version of an existing output file 19  
 creation date  
   disk 20  
   diskette 22

data file utility (see DFU)  
 data link control characters 251  
 data set label 219, 220  
 DATE command statement  
   description 62  
   format summary 51

date format (see also // DATE statement, DATE command statement, SET command statement)  
   display 86  
   in // FILE statement 16, 21-22  
   option 201  
 DATE OCL statement (see // DATE statement)  
 DATE procedure  
   contents 234  
   description 62  
 date, setting (see also // DATE statement, DATE command statement, SET command statement)  
   job 16  
   system 16  
 decimal and hex conversions 217, 218  
 decreasing the library size 155  
 default value  
   definition 253  
   showing in formats 55, 114  
 DELETE command statement  
   description 63  
   example 63  
   format summary 51  
 delete function of \$MAINT utility program 173-175  
 DELETE procedure  
   contents 234  
   description 63  
 deleting a file  
   at diskette initialization 69, 141  
   caution 194  
   using DELETE 63, 132  
 deleting from the library 80, 209-211  
   (see also \$MAINT utility program delete function)  
 deleting records from a file 75, 122  
 describing a disk file 97, 98  
   (see also // FILE statement)  
 determining space available in the library 209  
 determining space available on the disk 210  
 DFU (data file utility)  
   applying PTFs to 196-198  
   installing 203  
 diagnostic information 224  
 directory (see library directory)  
 disk block 253  
 disk capacity display 86  
 disk copy/display utility program (see \$COPY utility program)  
 disk files  
   adding to diskette 83, 122  
   adding to library 89  
     (see also \$MAINT utility program copy function)  
   converting to standard interchange diskette 91, 117  
   copying 75, 83  
   creating 97, 98  
   deleting (see deleting a file)  
   deleting records from 75, 122  
   describing 97, 98  
     (see also // FILE statement)  
   displaying 64, 122  
   obtaining space for 97  
   space allocation 81  
 disk free space, compressing 58, 179

disk read/write error 119, 225  
 disk record to block conversion 216  
 disk reorganization utility program (see \$PACK utility program)  
 disk volume label (VOL1) 154, 221  
 diskette copy utility program (see \$DUPRD utility program)  
 diskette data set label 219, 220  
 diskette defects 142  
 diskette files  
   adding to disk 91, 117  
   converting to disk 91, 117  
   creating 98, 99  
   expiration date for 21-22  
   standard interchange 219, 220  
   system 220, 221  
 diskette formats 219-221  
   (see also \$INIT utility program; INIT procedure)  
 diskette formats and diskette data files 219-221  
 diskette free space, compressing 59, 136  
 diskette labeling and initialization utility program (see \$INIT utility program)  
 diskettes  
   backup 198  
   PID 193  
   PTF 193  
   SCP 193  
 DISPLAY command statement  
   description 64  
   example 64  
   format summary 51  
 DISPLAY procedure  
   contents 235  
   description 64  
   displaying a file 64, 122  
   displaying messages and OCL statements 28, 74  
   displaying system information 86, 187  
   displaying VTOC 58, 146  
 DUMP command statement  
   description 225, 226  
   format summary 223  
 DUMP procedure  
   contents 235  
   description 225, 226  
  
 EBCDIC codes 2-48  
 ELSE expression 46-47  
 end of data 34  
   (see also /\* (end of data) statement)  
 end of OCL statements 31  
 entering OCL statements 3  
 erasing a file 63, 132  
 error logging area 154, 253  
 evoking a procedure 39-41  
 examples  
   \$BICR utility program 118  
   \$BUILD utility program 120  
   \$COPY utility program 131  
   \$DELET utility program 134, 135

examples (continued)

- \$DUPRD utility program 138
- \$HIST utility program 140
- \$INIT utility program 145
- \$LABEL utility program 146-149
- \$LOAD utility program 150
- \$MAINT utility program
  - allocate function 156
  - copy function 166-172
  - delete function 175
- \$MGBLD utility program 178
- \$SETCF utility program
  - BSCA environment 183
  - OVERRIDE RPG BSCA specifications 184
  - system environment 181
  - trace functions 185
- // \* (message) statement 34
- // COMPILE statement 15
- // DATE statement 16
- // FILE statement
  - disk 20
  - diskette 22
- // FORMS statement 23
- // IMAGE statement 25
- // INCLUDE statement 26-27
- // LOAD statement 27
- // LOG statement 28
- // MEMBER statement 30
- // SWITCH statement 32
- // SYSLIST statement 33
- COPY11 command statement 59, 60
- CREATE command statement 61
- creating a message source and load member 178
- creating an offline multivolume file 100
- DELETE command statement 63
- disk VTOC display 146
- diskette VTOC display 148
- DISPLAY command statement 64
- ELSE expression 46, 47
- FROMLIBR command statement 67
- IF expression 46, 47
- INIT command statement 70
- LISTLIBR command statement 73
- OCL and procedure jobs 107
- ORGANIZE command statement 76
- printing of library directory entry 168
- printing of system information 167
- procedure coding 47-49
- reading an offline multivolume file 101
- REMOVE command statement 80
- RESTORE command statement 82
- SAVE command statement 84
- TRANSFER command statement 93
- existence testing parameter 44-45, 253
- expiration date 21-22
- extended format, diskette 219-221
  - (see also \$INIT utility program; INIT procedure)
- external indicators 32, 82

file

- disk (see disk files)
- diskette (see diskette files)
  - permanent 19
  - scratch 19
  - temporary 19
- file delete utility program (see \$DELET utility program)
- FILE OCL statement (see // FILE statement)
- file segment 100, 253
- FILEBKUP procedure, example of procedure coding 47-49
- FIXDFILE 224, 225
- format diskette 69, 141
- format 1 record 189, 253
- FORMS OCL statement (see // FORMS statement)
- free space, disk 58, 179
- FROMLIBR command statement
  - description 65-67
  - examples 67
  - format summary 52
- FROMLIBR procedure
  - contents 235
  - description 65-67

general form of OCL statements 5-6  
glossary 253-256

hex and decimal conversions 217, 218  
hex form of standard characters 241-243  
HISTORY command statement

- description 68
- format summary 52

HISTORY file 68, 225, 226  
HISTORY file display utility program (see \$HIST utility program)  
HISTORY procedure

- contents 236
- description 68

how to use this manual ix

identifier

- definition 253
- OCL statement 5
- utility control statement 114

IF expression 44-46  
IMAGE OCL statement (see // IMAGE statement)  
INCLUDE OCL statement (see // INCLUDE statement)  
increasing the library size 155  
indicators, external 32, 86  
information in OCL statements 5-6  
INIT command statement

- description 69, 70
- examples 70
- format summary 52

**INIT procedure**  
   contents 236  
   description 69-70  
**Initial program load (IPL)**  
   definition 3, 254  
   from diskette 79  
**initialization** 254  
   (see also \$INIT utility program; INIT procedure)  
**INQ key** 152  
**inquiry**  
   Interrupt 115, 151, 152  
   option 151, 152, 254  
   request 152, 254  
   support 79  
**INQUIRY/OFFLINE option** (see also RELOAD display)  
   availability on system 86  
   deleting 210, 211  
   requesting 79, 150  
**INSTALL command statement**  
   description 202  
   format summary 191  
**INSTALL procedure** 202  
**installation**  
   application program 197  
   program product 203-208  
   system 193, 196-198  
**international format of expiration dates** 21  
**introduction**  
   to OCL statements 3  
   to procedures 37  
   to system configuration, installation, and  
   modification 189  
   to system utility programs 114  
**IPL (initial program load)**  
   definition 3, 254  
   from diskette 79

**job date** 16, 62  
**job stream**  
   and // INCLUDE 26  
   definition 4, 254  
   modifying procedure 42-47

**keyword parameter**  
   definition 254  
   OCL statement 6  
   utility control statement 114

**label**  
   data set 219, 220  
   disk file 17  
   diskette file 21

**level, procedure** 41, 255  
**librarian** (see \$MAINT utility procedure)  
**library** (see system library)  
**library directory**  
   area 154, 254  
   changing the size of 79, 150  
   (see also RELOAD display)  
   definition 254  
   entry 154, 168  
   formula for number of entries 150  
   information in entries 168  
**library maintenance utility program** (see \$MAINT utility  
 program)  
**library members**  
   creating a file from 65  
   definition 3, 254  
   deleting 80, 209-211  
   naming 154, 166  
   organization of 154, 155  
**library requirements** 209  
**LINES command statement**  
   description 71  
   format summary 52  
**lines printed per page**  
   displaying number of 86  
   setting number of  
     using \$SETCF utility program 181  
     using // FORMS statement 23  
     using LINES procedure 71  
     using SET procedure 85  
**LINES procedure**  
   contents 236  
   description 71  
**list of abbreviations** viii  
**listing the**  
   files 64, 122  
   HISTORY file 68  
   system library 72  
   VTOCs 58  
**LISTLIBR command statement**  
   description 72-73  
   examples 73  
   format summary 50  
**LISTLIBR procedure**  
   contents 236  
   description 72, 73  
**load member** 3, 254  
**LOAD OCL statement** (see // LOAD statement)  
**load program** 27  
**loading and running programs** 105, 106  
**LOG command statement**  
   description 71  
   format summary 50  
**LOG OCL statement** (see // LOG statement)  
**LOG procedure**  
   contents 237  
   description 71



main storage  
  display 152  
  dump 225  
megabyte 254  
member (see library members)  
MEMBER OCL statement (see // MEMBER statement)  
message control statement 177, 254  
message display 28, 74  
message identification code (see MIC)  
message levels 29  
message load member  
  and command key requests 40  
  creating 171, 61, 176  
  definition 254  
  example of creating 178  
message member 29, 34  
  (see also message load member; message source member)  
message OCL statement (see // \* (message) statement)  
message source member 177, 254  
message text statement 177  
messages to operator 34  
MIC (message identification code)  
  definition 254  
  for creating command key requests 40  
  for creating message load members 176, 177  
modifying a procedure job stream 42-47  
multivolume file 99, 254  
  (see also offline multivolume file)

naming library members 154, 166  
nested procedure 41, 254  
null entry 46, 254

object program  
  definition 254  
  error in 224  
  running 15, 105, 106  
obtaining space for a disk file 97  
OCL (operation control language) statements (see also  
  \* (comment), /\* (end of data), // \* (message), // COMPILE,  
  // DATE, // FILE, // FORMS, // IMAGE, // INCLUDE,  
  // LOAD, // LOG, // MEMBER, // PAUSE, // RUN,  
  // SWITCH, // SYSLIST)  
  and job stream 3-4, 107-109  
  coding rules for 5-8  
  definition of 5-6, 255  
  description of 15-34  
  displaying 28, 74  
  entering 3  
  general form of 5  
  identifiers for 5  
  information in 5-6  
  introduction to 3-4  
  tables of 10-14  
offline multivolume file 99-104, 254

operation control language (OCL) statements (see  
  OCL statements)  
ORGANIZE command statement  
  description 75, 76  
  examples 76  
  format summary 53  
ORGANIZE procedure  
  contents 237  
  description 75, 76  
overflow, printer 23  
OVERRIDE command statement  
  description 77  
  format summary 53  
OVERRIDE procedure  
  contents 237  
  description 77  
OVERRIDE RPG BSCA specifications  
  control statements 184  
  description 181, 184  
  example 185

parameters  
  condition 44-45  
  definition 255  
  existence testing 44-45  
  keyword 6  
  OCL 6  
  positional  
    defined 42, 255  
    showing in formats 53  
  procedure 26, 42  
  statement 45  
  symbolic 6  
  table of OCL 12-14  
  utility control statement 114  
PATCH command statement  
  description 226, 227  
  format summary 223  
PATCH procedure  
  contents 226, 227  
  description 226, 227  
pause message 30  
PAUSE OCL statement (see // PAUSE statement)  
permanent file 19  
PID distribution diskette 193  
polling and addressing characters 245, 246  
positional parameter  
  defined 42, 255  
  showing in formats 55  
print belt  
  characters  
    entering from keyboard 24-25  
    entering from source member 25  
    list of 241-243  
  displaying image of 86  
  setting image for  
    \$SETCF utility program 181  
    // IMAGE statement 24  
  SET procedure 85

- printing from the library 167-170
- printing system information 167
- procedure coding, example 47-49
- procedure member 3, 255
- procedure name 26
- procedure parameters 26, 42
- procedures (see also ALTERBSC, APAR, APPLYPTF, BACKUP, BUILD, CATALOG, CNFIGSCP, command statements, COMPRESS, COPY11, CREATE, DATE, DELETE, DISPLAY, DUMP, FROMLIBR, HISTORY, INIT, INSTALL, LINES, LISTLIBR, LOG, ORGANIZE, OVERRIDE, PATCH, REBUILD, RELOAD, REMOVE, RESTORE, SAVE, SET, STATUS, SYSLIST, TOLIBR, TRACE, TRANSFER)
  - creation of 38
  - definition 37, 255
  - evoking 26, 39-41
  - execution of 41
  - introduction to 37-47
  - levels of 41, 255
  - nested 41
  - parameters 26, 42
  - SCP 38
  - service 38, 223
  - system configuration, installation, and modification 191
- program check 225
- program date (see job date)
- program product installation 203-208
- program product PTFs 196
- programs, loading and running 105, 106
- PTF diskette 193

- read/write error, disk 119, 225
- reading an offline multivolume file 101
- REBUILD command statement
  - description 78
  - format summary 53
- rebuild data file utility program (see \$REBLD utility program)
- REBUILD procedure
  - contents 237
  - description 78
- record mode 89, 255
- record, block, and sector conversions 216
- release level, displaying 86
- RELOAD command statement
  - description 79
  - format summary 53
- RELOAD display 211-214
- reload library utility program (see \$LOAD utility program)
- RELOAD procedure
  - contents 237
  - description 79
- relocation directory (RLD) 255
- REMOVE command statement
  - description 80
  - examples 80
  - format summary 53
- REMOVE procedure
  - contents 238
  - description 76
  - rename diskette 142
  - reorganize disk 174, 59, 179
  - reorganize library 57, 116
- RESTORE command statement
  - description 81, 82
  - examples 82
  - format summary 53
- restore disk files (see RESTORE command statement; RESTORE procedure)
- RESTORE procedure
  - contents 238
  - description 81, 82
- restore system information 78
- retention period 16, 62
- RLD (relocation directory) 255
- rollout area
  - definition 255
  - use of 152, 154
- RPG II
  - applying PTFs to 196
  - compiler 15
  - installation 203
  - installation verification 206-208
- RUN OCL statement (see // RUN statement)

- SAVE command statement
  - description 83, 84
  - examples 84
  - format summary 53
- SAVE procedure
  - contents 238
  - description 83, 84
- scheduler work area (SWA) 154, 255
- SCP diskette 193
- SCP procedures 38, 51-93
  - (see also procedures)
- scratch file 19
- sector 255
- sector mode 89, 255
- sector number to block number conversion 216
- segment, file 99, 253
- selecting library members to delete 210
- service procedures 223-229
  - (see also procedures)
- SET command statement
  - description 85
  - format summary 53
- SET procedure
  - contents 238
  - description 85
- set utility program (see \$SETCF utility program)
- setting
  - job date 16
  - number of lines printed per page 85, 182
  - print belt image 85, 182

setting (continued)  
   system date/date format  
     \$SETCF utility program 181, 182  
     // DATE statement 16  
     DATE procedure 62  
     SET procedure 85  
   system environment 85, 181  
   trace functions 181, 185-187  
 SEU (source entry utility)  
   applying PTFs to 196  
   installation of 203  
   installation verification 204, 205  
 skip to next page, printer 28  
 source entry utility (see SEU)  
 source member 3, 255  
 source program  
   causing error 224  
   definition 255  
   specified in // COMPILE statement 15  
 sort  
   applying PTFs to 196  
   installation of 203  
 space allocation, changing  
   disk 81  
   library 79, 155  
   library directory 79, 150  
 specifying library size 155  
 standard interchange diskette 219  
   (see also INIT procedure; TRANSFER procedure)  
 standard interchange file 219, 220  
   (see also TRANSFER procedure)  
 standard interchange utility program (see \$BICR utility program)  
 statement identifiers, OCL 5  
 statement parameter 45, 255  
 statement tables  
   command  
     SCP 51-54  
     service 223  
     system configuration, installation, and modification 191  
   OCL 10-14  
 statements  
   command (see command statements)  
   OCL (see OCL statements)  
   utility control (see utility control statements)  
 STATUS command statement  
   description 86, 87  
   format summary 54  
 status display utility program (see \$STATS utility program)  
 STATUS procedure  
   contents 239  
   description 86, 87  
 subroutine member 3, 255  
 substitution in procedures 42-44  
 SWA (scheduler work area) 154, 255  
 switch indicators 32, 86  
 SWITCH OCL statement (see // SWITCH statement)  
 symbolic parameter 6, 255  
 SYSLIST command statement  
   description 88  
   format summary 54  
   SYSLIST OCL statement (see // SYSLIST statement)  
   SYSLIST procedure  
     contents 239  
     description 88  
   system configuration 4, 189-214  
   system configuration, installation, and modification 189-214  
   system configuration record 85, 255  
   system date/date format  
     displaying 86  
     setting  
       \$SETCF utility program 181, 182  
       // DATE statement 16  
       DATE procedure 62  
       SET procedure 85  
   system environment, setting 85, 181  
   system failure 179, 180  
   system file 220, 221  
   system information  
     displaying 86, 188  
     restoring 78  
   system installation 193, 196-198  
   system library  
     definition 3, 255  
     deleting from 80, 209-212  
     description 153-155  
     determining space available in 209  
     listing 72  
     requirements 209  
     space allocation 79, 150  
   system list (see // SYSLIST statement; SYSLIST procedure)  
   system modification 209-214  
   system status, displaying 86-188  
   system utility programs (see also \$BACK, \$BICR, \$BUILD, \$COPY, \$DELET, \$DUPRD, \$HIST, \$INIT, \$LABEL, \$LOAD, \$MAINT, \$MGBLD, \$PACK, \$REBLD, \$SETCF, \$STATS)  
     definition 255  
     description 116-188  
     introduction to 113, 114  
     utility control statements for (see utility control statements)  
 tables  
   OCL statement 10-11  
   OCL statement parameters 12-14  
   SCP command statements 51-54  
   service command statements 223  
   system configuration, installation, and modification  
     command statements 191  
   temporary file 19  
   text statement, message 177  
   TOLIBR command statement  
     description 89, 90  
     format summary 54  
   TOLIBR procedure  
     contents 239  
     description 89, 90

TRACE command statement  
  description 227-229  
  format summary 223  
trace functions 181, 185-187  
TRACE procedure  
  contents 239  
  description 227-229  
trace table 227  
TRANSFER command statement  
  description 91-93  
  examples 93  
  format summary 54  
TRANSFER procedure  
  contents 229-230, 239, 240  
  description 91-93

using OCL statements and procedures 95-110  
utility control statements  
  conventions for describing 114  
  definition 4, 255  
  rules for coding 114  
  writing 113  
utility programs (see system utility programs)

value 256  
vol-id (volume identification) 22  
VOL1 (volume label)  
  disk 154, 256  
  diskette 221  
volume label (see VOL1)  
volume table of contents (see VTOC)  
VTOC (volume table of contents)  
  definition 58, 256  
  listing 58, 146-149  
VTOC display utility program (see \$LABEL utility program)

writing over an existing disk file 19  
writing utility control statements 114

## READER'S COMMENT FORM

IBM System/32  
System Control Programming  
Reference Manual

GC21-7593-1

### YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

*Page*            *Comment*

Due to the current paper shortage, we will not send a reply to your comments unless you check the box below.

I would like a reply.

Name \_\_\_\_\_

Address \_\_\_\_\_

● Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Cut Along Line

IBM S/32 System Control Programming Reference Manual (File No. S32-36) Printed in U.S.A.

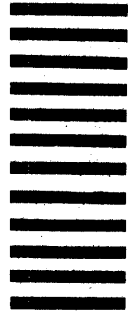
GC21-7593-1

Fold

Fold

FIRST CLASS  
PERMIT NO. 387  
ROCHESTER, MINN.

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



POSTAGE WILL BE PAID BY . . .

IBM Corporation  
General Systems Division  
Development Laboratory  
Publications, Dept. 245  
Rochester, Minnesota 55901

Fold

Fold



**International Business Machines Corporation**  
General Systems Division  
5775D Glenridge Drive N.E.  
Atlanta, Georgia 30301  
(USA Only)

**IBM World Trade Corporation**  
821 United Nations Plaza, New York, New York 10017  
(International)



**International Business Machines Corporation**  
**General Systems Division**  
**5775D Glenridge Drive N.E.**  
**Atlanta, Georgia 30301**  
**(USA Only)**

**IBM World Trade Corporation**  
**821 United Nations Plaza, New York, New York 10017**  
**(International)**