SPERRY UNIVAC
COMPUTER SYSTEMS

ATTN: CHARLIE GIBBS

01100
CAV208M45541 UP 8379 R4A

UAS

SPERRY UNIVAC
1 - 1818 CORNWALL STREET
VANCOUVER B C

V6J 1C7

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

System/3 to OS/3 Transition

User Guide/Programmer Reference

UP-8379 Rev. 4-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) System/3 to OS/3 Transition User Guide/Programmer Reference", UP-8379 Rev. 4.

This update covers removal of support for split-cylinder file allocation, corrects several errors in sample OCL and JCL streams, and contains other minor technical changes.

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8379 Rev. 4-A. To receive the complete manual, order UP-8379 Rev. 4.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A50, B00, B50, 18, 18U, 20, 20U, 28U, 76, and 76U (Package A to UP-8379 Rev. 4, 70 pages plus Memo)	Library Memo for UP-8379 Rev. 4-A
		RELEASE DATE:
		September, 1982

PAGE STATUS SUMMARY

ISSUE: Update A – UP-8379 Rev. 4

RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	A
Preface	1	Orig.
	2	A
Contents	1	Orig.
	2, 3	A
	4, 5	Orig.
PART 1	Title Page	Orig.
1	1, 2	Orig.
	3	A
	4 thru 11	Orig.
PART 2	Title Page	Orig.
2	1, 2	Orig.
	3, 4	A
	5 thru 7	Orig.
	8 thru 12	A
	13	Orig.
	14	A
	15 thru 17	Orig.
3	1	Orig.
	2	A
4	1	A
	2 thru 6	Orig.
	7	A
	8, 9	Orig.
PART 3	Title Page	Orig.
5	1 thru 4	Orig.
	5, 6	A
	6a	A*
	7	Orig.
	8	A
	9 thru 19	Orig.
	20, 21	A
	22	Orig.
	23, 24	A
	25	Orig.
	26	A
	26a	A*
	27 thru 30	Orig.
	31, 32	A
	33	Orig.
	34, 35	A
5 (cont)	36, 37	Orig.
	38	A
	39 thru 41	Orig.
	42	A
	43 thru 47	A**
	48	A
	49 thru 51	A**
	52 thru 65	Orig.
	66	A
	67, 68	A**
	69 thru 102	Orig.
6	1 thru 27	Orig.
Index	1 thru 3	A
	4	Orig.
	5 thru 7	A
	8	Orig.
	9, 10	A
User Comment Sheet		

*New pages

**Pages 5–43 thru 5–47, 5–49 thru 5–51, and 5–67 and 5–68 have been deleted.

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY UNIVAC Operating System/3 (OS/3). This manual specifically describes:

- The transcription of your source libraries, operation control language (OCL) procedures, and data files from the IBM System/3 (System/3) for use under OS/3
- Conversion of your System/3 programs for use under OS/3
- The method used to run your existing OCL control streams with a minimum amount of modification. Its intended audience is the programmer, system analyst, or a manager of System/3 installation.

One other manual that covers the subject of file transcription and the use of OCL control streams under OS/3 is an introductory manual. The introductory manual, however, only briefly describes the facilities.

This user guide/programmer reference is divided into the following parts:

- PART 1. INTRODUCTION

  Introduces the various aspects of migrating from a System/3 to OS/3 including transcription, program language conversion, OCL and OCL routines, and SORT3. Describes and illustrates the rules used in describing OCL statements and how these statements should be used.

- PART 2. CONVERSION PROCESS

  Presents the details of converting from a System/3 to OS/3. Describes data and source and procedure file transcription in terms of what it is and how it is used. Describes the various steps that must be taken on both the System/3 and a Sperry Univac system using OS/3. Describes the function and operation of a number of routines run to support the transition effort including SORT3 and OCL-to-JCL converter. Also presents information relevent to the conversion of your language programs so that they can be recomplied and executed on OS/3.

- PART 3. USE OF OPERATION CONTROL LANGUAGE (OCL)

  Describes the methods needed to run an OCL control stream in the OS/3 environment. Provides an explanation of how each OCL statement is used by OS/3.

The following listing identifies the documents that you should be familiar with or have available to help you in your conversion effort. Each document is referenced in the appropriate section of this manual. Since OS/3 is used on two types of systems (Series 90 and System 80), two distinct manuals may exist for the same topic, one for each type of system. The list shows the appropriate document number for use with each system. In addition to the documents listed, the following documents are also referenced: OCL to JCL converter (JCLCON801) user guide, UA-0423; CCP screen source statement to SFS converter (CCPCON301) user guide, UA-0485; and System/3 COBOL to OS/3 COBOL '74 (COBTRN309) user guide, UA-0466.

Document Title	UP Number	
	Series 90	System 80
OS/3 system service programs (SSP) user guide	8062	8841
OS/3 job control user guide	8065	8065
OS/3 data utilities user guide/programmer reference	8069	8834
assembler user guide	8061	8913
OS/3 system messages manual	8076	8076
consolidated data management concepts and facilities	8825	8825
basic data management user guide	8068	NA
1974 American National Standards COBOL programmer reference	8613	8613
FORTRAN IV programmer reference	8193	8814
OS/3 report program generator II (RPG II) user guide	8067	8067
OS/3 general editor user guide/programmer reference	8828	8828
supervisor user guide	8075	NA
supervisor macroinstruction language user guide/programmer reference	NA	8832
screen format services user guide/ programmer reference	8802	8802

Contents

PART 3. USE OF OPERATION CONTROL LANGUAGE (OCL)

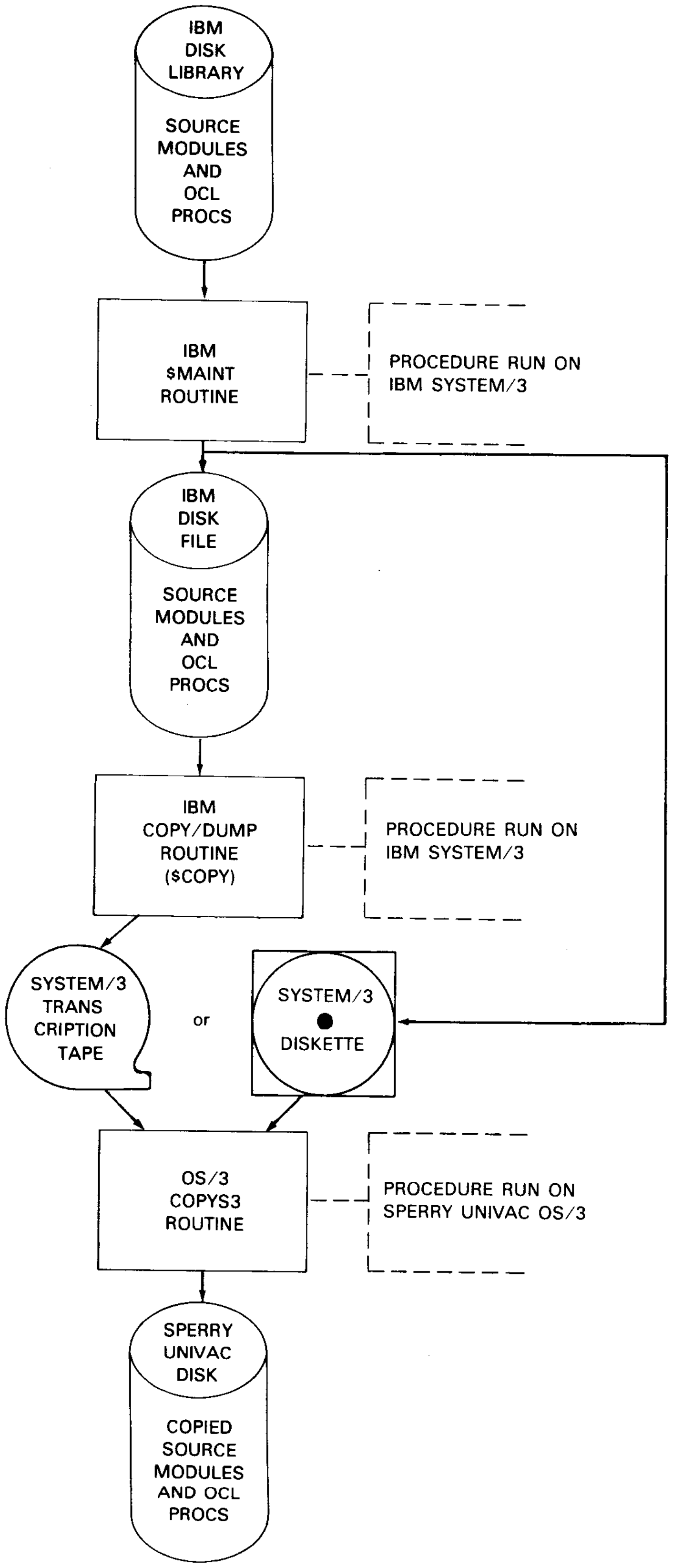
IBM DISK LIBRARY
SOURCE MODULES AND OCL PROCS
IBM $MAINT ROUTINE
PROCEDURE RUN ON IBM SYSTEM/3
IBM DISK FILE
SOURCE MODULES AND OCL PROCS
IBM COPY/DUMP ROUTINE ($COPY)
PROCEDURE RUN ON IBM SYSTEM/3
SYSTEM/3 TRANS CRIPTION TAPE
or
SYSTEM/3 DISKETTE
OS/3 COPYS3 ROUTINE
PROCEDURE RUN ON SPERRY UNIVAC OS/3
SPERRY UNIVAC DISK
COPIED SOURCE MODULES AND OCL PROCS

Figure 1—1. Source Module and OCL Procedure Transcription Flow for Model 10, 12, and 15

1.2.2. Data Transcription for System/3 Models 10, 12, and 15

The data transcription process has one procedure and two steps. In step 1, you use the IBM $COPY routine on the System/3 to transcribe your data files from an IBM disk to a magnetic tape or diskette. In step 2, processed in an OS/3 environment, you use the OS/3 data utility routine to copy the contents of the transcribed tape or diskette (from step 1) onto a SPERRY UNIVAC DISK. This procedure is illustrated in Figure 1-2.

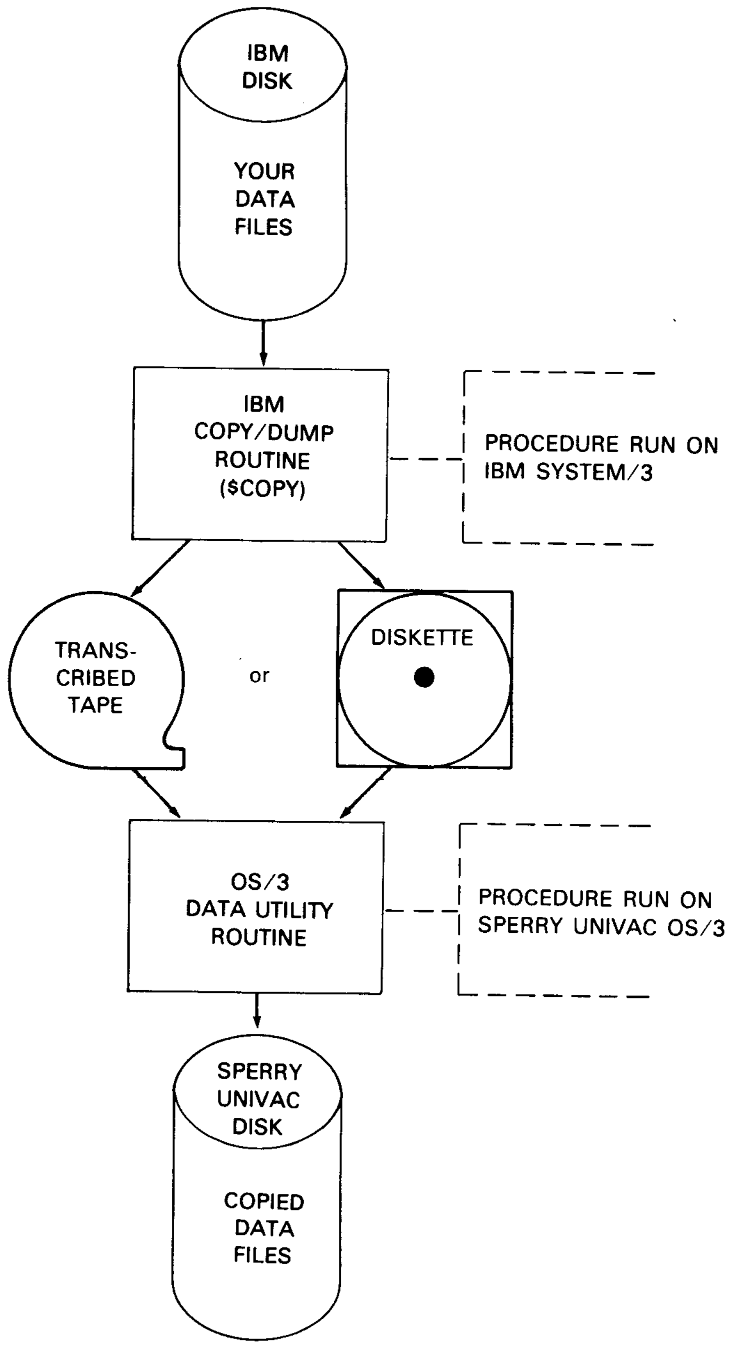
IBM DISK
YOUR DATA FILES
IBM COPY/DUMP ROUTINE ($COPY)
PROCEDURE RUN ON IBM SYSTEM/3
TRANS-CRIBED TAPE
or
DISKETTE
OS/3 DATA UTILITY ROUTINE
PROCEDURE RUN ON SPERRY UNIVAC OS/3
SPERRY UNIVAC DISK
COPIED DATA FILES

Figure 1—2. Data File Transcription Flow

By specifying *R1* as the unit specification, you indicate that the library file to be searched for the program is $Y$LOD, and it's located on the volume that contains the temporary job run library file ($Y$RUN) for the job. Each job that enters the OS/3 environment is assigned its own $Y$RUN file on the volume set aside for all the $Y$RUN job files.

Each installation can have a separate volume containing the area set aside for all the $Y$RUN job files, or they can have it residing on SYSRES. This is determined when you perform the initial program load. If you specify *R1*, and the area for all the $Y$RUN job files is on its own volume, then this volume is searched for $Y$LOD. ($Y$RUN must be allocated since it does not exist on SYSRES.) If you specify *R1*, and the area for all the $Y$RUN job files is on SYSRES, then, by default, SYSRES would be searched for $Y$LOD.

By specifying *R2*, you indicate that $Y$OCLOD on the volume containing the job's $Y$RUN file is to be searched for the program (remember, you have to allocate $Y$OCLOD, since it's not an OS/3 system library). The same default (in respect to not having $Y$RUN files on a separate volume) applies.

You can also specify your own private load library name.

NOTE:

If the volume containing the $Y$RUN job files is specified as the SYSRES device (determined at system IPL time), then specifying the parameter R1 *is exactly the same as specifying* F1. *Likewise, specifying the parameter* R2 *is the same as specifying* F2.

2.3.1. Generating Diskette Transfer Files

To generate diskette transfer files, you must run the System/3 library maintenance routine ($MAINT). To execute this routine you must code a control stream using the System/3 OCL. The control stream you use should look like this:

```
// LOAD $MAINT,F1
// FILE NAME-LIBMODR1,UNIT-3741,RECL-96
// RUN
// COPY FROM-R1,TO-3741,FILE-LIBMODR1
// ENTRY LIBRARY-S,NAME-ALL
// ENTRY LIBRARY-P,NAME-ALL
// NEND
// END
```

The first statement (LOAD) identifies the program to be run and indicates which disk contains the program. In this example, the program name is $MAINT and it is located on a disk with a unit designation of *F1*. Possible designations are *F1, F2, R1,* and *R2* (use the designation appropriate for your system).

The FILE statement is used to define the diskette file to be generated. The parameters of the FILE statement identify the characteristics of the file.

The *NAME* parameter assigns a name to the file. The *UNIT* parameter identifies the device in question as a diskette. The RECL parameter specifies the number of bytes per record. In this example, there are 96 bytes per record.

The next statement (RUN) initiates the execution of the library maintenance routine. Following the RUN statement are the control statements for the library maintenance routine. The COPY statement initiates the copy function of $MAINT and the parameters indicate the files to be copied. The *FROM* parameter indicates which disk drive contains the library to be copied. This value can be *F1*, *F2*, *R1*, or *R2*. In our example we use *R1*.

The *TO* parameter specifies the output file of the library to diskette copy. The 3741 entry indicates a diskette is the output file.

The diskette is further identified with the next parameter, the *FILE* parameter. The entry specified for this parameter must be the same entry as was made for the *NAME* parameter of the preceding FILE statement that identified the diskette file.

Following the COPY statement are two ENTRY statements. These statements identify the modules within the library that we wish to have copied. Only one entry per ENTRY statement is permitted and since we wish to copy both the source and procedure modules, two ENTRY statements are required.

In addition, since we also wish to copy all of the source and procedure modules from the library, we specify the *ALL* parameter.

The last two statements in the control stream are the NEND statement and the END statement which terminate the library-to-disk function and the library maintenance routine, respectively.

This procedure must be used for each source and procedure library being copied. You must create a new diskette file for each execution.

NOTE:

If your diskette device has been allocated as a punch device, then the TO parameter in the COPY statement should be changed to TO-PUNCH.

2.3.2. Generating Tape Transfer Files

The 2-step process used to generate tape transfer files from your System/3 source and procedure libraries is described in 2.3.2.1 and 2.3.2.2.

The next statement (FILE) defines the output file for tape. The first parameter needed is *NAME*. For this, an exact value of *COPYO* must be used. The second parameter is *UNIT*, which indicates the unit designator where the tape that's going to receive the data file is loaded. This can be either *T1, T2, T3,* or *T4.*

We assumed the tape is loaded on *T1*. If you want to assign a name to the transcribed data file on tape, you can do so with the *LABEL* parameter (this is optional). This name is the file identifier, which is physically written in the header label on the tape, and can be alphanumeric characters.

We assigned the filename LIBMODR1, which was specified on our FILE statement. The next parameters are the *BLKL* and *RECL* parameters. These parameters indicate the length in bytes for each physical block and logical record on tape.

The *RECL* parameter must be the same as the record length in the disk file, and the *BLKL* must be a multiple of the record length. The *RECL* value must be in the range of 21 to 128 bytes, while the maximum *BLKL* value is 1280 bytes. In our example, we used a length of 96 bytes for both parameters (because this is the value we used in our COPY statement back in step 1).

The next parameter is *RECFM* which indicates the record format. The record format can either be fixed (F) or fixed length blocked (FB). Since we made our *RECL* and *BLKL* parameters the same, our *RECFM* parameter would be fixed length (F).

The next parameter, *REEL* is required and identifies the tape volume being used. When you use the OS/3 COPYS3 routine to copy this tape to a disk, you indicate, on a FILE job control statement, which volume to use, thus ensuring the correct tape is loaded. Since this is the tape that's going to contain the library file, we called this tape LIBFIL.

The last parameter is the *DENSITY* parameter. This value must be compatible with the IBM and SPERRY UNIVAC equipment used in the transcription.

Next, since the input data file is not a system library, it also must be defined with a FILE statement. The first parameter needed is *NAME*, and it has an exact value of *COPYIN.* Then, you have to provide the unit designator, indicating where the disk volume is located, with the *UNIT* parameter. This can be either *R1, R2, F1, F2, D1, D2, D3,* or *D4*. We assumed *R2*, because we used this value in our last step.

Next, the *PACK* parameter is used to tell the system the name of the disk being used. For this example, we used DISK01. The *LABEL* parameter identifies the file on the disk (if you omit this parameter, the name from the *NAME* parameter is used). Assume that the file is identified as LIBMODR1.

Following the FILE statements, you code a RUN statement to initiate the execution of the copy operation. Immediately following RUN, you must code the control statements used to control the copy operation. In this case you must include the COPYFILE statement. The parameter *OUTPUT-FILE* instructs the program to copy the input file to the tape file identified in the output FILE statement. The final statement is the END statement which terminates the $COPY program.

2.3.3. Source and OCL Procedure Modules from Tape or Diskette to OS/3 Disk

This step consists of transcribing the source and OCL procedure modules from your IBM tape or diskette to a SPERRY UNIVAC disk library via the OS/3 COPYS3 routine. However, the COPYS3 routine runs under the direction of OCL statements, as well as OS/3 job control statements. (For a detailed discussion of the COPYS3 program, see 2.5.)

The following coding example shows a typical OCL control stream, used to copy all the source and OCL procedure modules from tape to our disk library.

```
1.   // LOAD COPYS3,F1
2. { // FILE NAME-INPFILE,UNIT-T1,REEL-LIBFIL,LABEL-LIBMODR1,RECL-96,
   { //1 BLKL-96, RECFM-F,DENSITY-16ØØ
3.   // RUN
4.   // PARAM DEFAULT-LIBRARY-R1
5.   // FIN
```

1. The LOAD statement indicates that the OS/3 COPYS3 routine is to be executed. *F1* indicates that this routine is located in the OS/3 load library file ($Y$LOD).

2. The FILE statement allows access to the tape file created in the previous step. The *NAME* parameter must have the value of *INPFILE*. The *UNIT* parameter is required, and indicates the unit designator where the tape is to be mounted (T1). The remaining parameters (*REEL, LABEL, RECL, BLKL, RECFM,* and *DENSITY*) must agree with the FILE statement parameters. If no *LABEL* parameter is specified, then we specify COPYO as the label file name for this statement (COPYO corresponds to the *NAME* parameter). We could omit specifying the *RECL, BLKL,* and *RECFM* parameters because they would default to 96, 96 and F, respectively. The *DENSITY* parameter indicates the recording density for the tape in bits per inch. The density given must be the same as it was when the tape was written.

   NOTE:

   The parameters for the // FILE statement in this example require a continuation line. As is shown, such a continuation line must begin with //1 and at least one space before entering the parameters.

3. The RUN statement causes the routine named on the LOAD statement (COPYS3) to be executed.

4. The PARAM statement tells the COPYS3 routine to copy the modules to the default library (R1). This is done in case the COPY librarian control statements in the file do not have the *TO code* parameter, which shows where the copy modules are to be placed. If no default library parameter is given, the default library is *F1*. The possible codes are *F1, F2, R1,* and *R2*.

5. The FIN statement indicates the end of the data, the end of the job, and turns off the card reader.

The following coding shows a control stream used to copy all the source and OCL procedure modules from diskette:

```
// LOAD COPYS3.F1
// FILE NAME-INPFILE,UNIT-K1,LABEL-LIBMODR1,BLKL-128,RECL-96,PACK-LIBFIL
// RUN
// PARAM DEFAULT-LIBRARY-R1
// FIN
```

NOTE:

All libraries except F1 ($Y$LOD on SSRES) must be created prior to using the COPYS3 routine.

2.4. DATA FILE TRANSCRIPTION FOR MODELS 10, 12, AND 15

The procedure for transcribing data files is comprised of two steps. The first step is run on the System/3 $COPY utility routine. The second step, the data utility routine, is run in an OS/3 environment. For simplification, each step of the procedure is discussed separately in 2.4.1 and 2.4.2.

2.4.1. Data Files to Tape or Diskette (System/3 Step)

This step consists of executing the IBM copy/dump program ($COPY) to copy your data files to tape or diskette transfer files. The following sample control stream shows a disk-to-tape file copy. The control stream can be used as a model for you to develop control streams to meet your particular requirements. A control stream for a disk-to-diskette file copy is presented at the end of this subsection.

```
// LOAD $COPY.F1
// FILE NAME-COPYIN,UNIT-D1,PACK-DATADK,LABEL-INDATA
// FILE NAME-COPYO,UNIT-T1,LABEL-DATAFILE,BLKL-128,RECL-128,RECFM-F,REEL-DATAFL
// RUN
// COPYFILE OUTPUT-FILE
// END
```

The first control statement needed is the LOAD statement to identify the program to be run. You must also supply a unit designation of either *R1, R2, F1,* or *F2* to indicate where the program is stored. We assumed *F1*.

The next statement defines the input file. Since it is not a system library, it must be defined with a FILE statement. The first parameter needed is *NAME*, and it has an exact value of *COPYIN*. Then, you have to provide the unit designator, indicating where the disk volume is located, with the *UNIT* parameter. This can be either *D1, D2, D3, D4, F1, F2, R1,* or *R2.* We assumed *D1* because data files usually reside in what is referred to as "data areas" (*D1, D2, D3,* or *D4*) rather than in what are called "simulation areas" (*R1, R2, F1,* or *F2*).

The *PACK* parameter is used to tell the system the name of the disk being used. For this example, we used DATADK. The *LABEL* parameter identifies the file on the disk (if you omit this parameter, the name from the *NAME* parameter is used). Assume that the data file is identified as INDATA.

The next statement (FILE) defines the output tape file. The first parameter is *NAME*. For this, an exact value of *COPO* must be used. The second parameter is *UNIT,* which indicates the unit designator where the tape that's going to receive the data file is loaded. This can be either *T1, T2, T3,* or *T4.* We assumed the tape is loaded on *T1.*

If you want to assign a name to the transcribed data file on tape, you do so with the *LABEL* parameter (this is optional). This name is the file identifier, which is physically written in the header label on the tape, and can be one to eight alphanumeric characters. We assigned a file identifier of DATAFILE.

BLKL and *RECL* are the next parameters, which indicate the length (in bytes) of each physical block on the tape and the length (in bytes) of each logical record. We used a length of 128 bytes for each of these parameters.

NOTE:

Do not use the LABEL, BLKL, or PACK parameters in the output file statement when creating diskette transfer files.

The *RECFM* parameter, indicating the format of the records, can be either fixed length (F) or fixed length blocked (FB). We made ours fixed length (F). The *REEL* parameter, which identifies the tape, also is required.

When you use the OS/3 data utility routine to copy this tape to a disk, you indicate on a FILE job control statement which volume to use, thus ensuring the correct tape is loaded. Since this is the tape that's going to contain the data file, we called this tape DATAFL.

A RUN statement, indicating that the program named on the LOAD statement ($COPY) is to be executed is the next requirement. Also, the $COPY routine has a control statement that must be used to copy the data file from disk to tape (COPYFILE).

You then place an END control statement (no parameters) in the control stream.

The control stream you use to transcribe your data file to diskette is:

```
// LOAD $COPY,F1
// FILE NAME-COPYIN,UNIT-D1,PACK-DATADK,LABEL-INDATA
// FILE NAME-COPYO,UNIT-3741,RECL-96
// RUN
// COPYFILE OUTPUT-3741
// END
```

2.4.2. Data File from Tape or Diskette to Disk (OS/3 Step)

This step utilizes the OS/3 data utility routine to transcribe your data file from tape or diskette to disk. This routine makes use of OCL control statements to direct its execution; you don't have to use OS/3 job control. If you want to know more about the data utility routine used, consult the data utility user guide.

```
1.  // LOAD DATA-F1
2.  // PARTITION 40960
3.  // FILE REEL-DATAFL,UNIT-T1,NAME-INPUT1,LABEL-DATAFILE
4.  // FILE UNIT-F2,PACK-DATADK,NAME-OUTPUT1,LABEL-INDATA,RECORDS-4000,RETAIN-P
5.  // RUN
6.   UTD A=(128,128),B=(128,128),FF,L0,MK1=(6,0),OM=(I,1,V,R)
7.  /*
    // FIN
```

1. The LOAD statement indicates that the OS/3 data utility routine (DATA) is to be executed. *F1* indicates that this program is located in $Y$LOD.

2. The PARTITION statement allocates 40960 decimal bytes of main storage for the job. The data utility routine is designed to run most efficiently in this amount of main storage. However, it does not absolutely need this amount; it can run in less main storage if necessary.

3. This FILE statement allows access to the tape file created by the IBM $KCOPY utility routine. The *REEL* and *LABEL* parameters, therefore, must agree with those specified when the tape was created. OS/3 assigns a device based on the default volume characteristics (*DENSITY* parameter not used). The *UNIT* parameter is required, and tells OCL that the FILE statement is for a tape. The value of *INPUT1* must be used for the *NAME* parameter.

   NOTE:

   The OS/3 FILE parameters should match the System/3 FILE parameters used to create the file. Be careful of the default value differences between systems.

4. This FILE statement defines the disk file that will contain the data. The values of the *PACK* and *LABEL* parameters can vary according to the number of the disk being used and the identifier of the file. In this file definition, the *UNIT* parameter tells OCL that this FILE statement is for a disk file. The *RECORDS* parameter performs the allocation in this case. It allocates enough space for 4000 records, each 256 bytes in length. If you feel this amount is too large or too small, you can change this value. And, if it's more convenient, the file size may be specified with the *TRACKS* parameter. The *RETAIN* parameter specifies that the file is a permanent file. It may also be specified as *RETAIN-T*, but *RETAIN-S* should never be specified, as it would cause the file to be scratched at the end of the program. The value of OUTPUT1 must be used for the *NAME* parameter.

5. The RUN statement causes the program named on the LOAD statement (DATA) to be executed.

6. The UTD mnemonic indicates that this is a tape or diskette-to-disk copy operation. The values of A indicate that the input records are 128 bytes long, and the input file blocks are also 128 bytes long. The values of B indicate that the records to be output to the disk file are to be 128 bytes long, and the output file blocks are also to be 128 bytes long. FF specifies fixed-length records. LO indicates that the standard INPUT1 tape label is to be used. MK1=(6,0) indicates that the file is to have one key, six bytes in length, located at address 0. OM=(I,1,V,R) indicates that the output file is to be an indexed (keyed) file, that the file index buffer generated by data utilities must be large enough to hold one 256-byte sector, the multivolume file indicator is to be set, and no record control byte is to be used.

7. This indicates the end of data (the data utility control statement) and end of job and turns off the card reader.

The following coding example shows a control stream that transcribes your data file from diskette to disk.

```
// LOAD DATA-F1
// PARTITION 40960
// FILE NAME-INPUT1,UNIT-K1,LABEL-DATAFILE
// FILE UNIT-F2,PACK-DATADK,NAME-OUTPUT1,LABEL-INDATA,RECORDS-4000,RETAIN-P
// RUN
 UTD A=(128,128),B=(128,128),FF,L0,MK1=(6,0),OM=(I,1,V,R)
/*
// FIN
```

2.5. COPYS3 ROUTINE

The COPYS3 routine transcribes the contents of either your diskette or tape files created by $MAINT on the System/3 into OS/3 library files. This routine transcribes all source and procedure modules but has no specific inclusion capabilities. If you only want specific modules, you can use the OS/3 librarian (LIBS) after the transcription is complete. COPYS3, when run under the control of the OCL processor, is system transparent since the transcription is to simulated System/3 libraries.

2.5.1. Input File Characteristics

Since the input to the COPYS3 routine can either be a diskette or tape file, the format of the files is essentially the same regardless of the input. The first record of each module in the file must be a // COPY record and the last record must be a // CEND record. The maximum record size is 128 bytes while the minimum is 21 bytes. On the // COPY record, however, only the first 72 bytes are analyzed.

When your input is a tape file, the block length must be a multiple of the record length up to a maximum size of 1280 bytes. If you recall, there is a default value of 96 bytes for both the block and record length parameters. However, you may change the record length of both diskette and tape files either via a DD job control statement (*RCSZ* parameter) or the OCL FILE statement (*RECL* parameter).

2.5.2. Input Record Characteristics

There are three types of records on these diskette or tape files:

- // COPY records
- Source data records
- // CEND records

Each type of record is discussed separately in 2.5.2.1, 2.5.2.2, and 2.5.2.3.

2.5.2.1. COPY Record

The COPY record is required and must be the first record of the module. This record is normally generated by the $MAINT routine when the file was created on your System/3. The COPY record itself is not copied, but it supplies needed information about your module.

The COPY record has the following format:

$$\texttt{// COPY FROM-READER,LIBRARY-}\begin{Bmatrix}\texttt{S}\\\texttt{P}\\\texttt{O}\\\texttt{R}\end{Bmatrix}\texttt{,NAME-name,TO-}\begin{Bmatrix}\texttt{F1}\\\texttt{F2}\\\texttt{R1}\\\texttt{R2}\end{Bmatrix}\texttt{,RETAIN-}\begin{Bmatrix}\texttt{T}\\\texttt{P}\\\texttt{R}\end{Bmatrix}$$

The *FROM* parameter must be READER; otherwise, the module is not copied. The *LIBRARY* parameter indicates the type of module to be copied. An S is for source modules, a P for procedure modules, while the O and R entries will prevent the module from being copied.

The *NAME* parameter contains the module name and can have a maximum of eight characters (six when using OCL). The *TO* parameter indicates the destination library. The libraries associated with the parameter entries are:

Parameter	OS/3 Library File
F1	$Y$LOD on SYSRES
F2	$Y$OCLOD on SYSRES
R1	$Y$LOD on SYSRUN
R2	$Y$OCLOD on SYSRUN

If this parameter is omitted, the default is the library specified on the // PARAM DEFAULT LIBRARY statement. If no // PARAM DEFAULT LIBRARY statement is supplied, the default is F1.

2.5.2.2. Source Data Record

All of the data records are copied from the input file to the specified OS/3 library. There is no restriction on the information these records can contain.

2.5.2.3. CEND Record

A // CEND record is required at the end of each module. The CEND record is not copied into the OS/3 library.

2.5.3. Executing COPYS3

You can execute the COPYS3 routine by using OCL, JCL, or simulating OCL under JCL. First, we'll discuss the OCL control streams.

Our first coding example shows the use of a diskette being specified as our input file.

```
1. // LOAD COPYS3,F1
2. // FILE NAME-INPFILE,UNIT-K1,PACK-DK0001,LABEL-FILEX
3. // RUN
4. // PARAM DEFAULT-LIBRARY-R1
5. // FIN
```

1. The LOAD statement indicates that the COPYS3 routine is to be executed. F1 indicates that this routine is located in the OS/3 load library file ($Y$LOD).

3. Transition Support

3.1. SORT3 ROUTINE

The System/3 compatible sort (SORT3) is an easy-to-use, *canned* sort program. It is modular in design and requires a minimum of user programming and does not need to be assembled or linked to your program. It increases the versatility of the OS/3 sort package by providing you with a program that is compatible with the System/3 sort. That is, SORT3 accepts, with minor differences, all System/3 sort specifications and offers all of the features of the System/3 sort that are feasible within the OS/3 operating system.

In addition to disk and tape input files, the SORT3 program is capable of processing input data from card files. It also provides you with added control over record sequencing, data reduction, and data disposition without the necessity of reverting to user own-code routines.

SORT3 is designed to operate under control of the OS/3 supervisor and data management systems. However, it can be initiated through either OS/3 job control language (JCL) or the operation control language (OCL) processor.

Running SORT3 under the OCL processor does not require you to make any changes to your existing System/3 sort job stream; the OCL statements and sequence specification remain the same as though you were running in a System/3 environment. After achieving full production or when you find it convenient, you should convert your OCL control streams to OS/3 job control streams using the Sperry Univac-supplied OCL-to-JCL converter, JCLCON801.

3.1.1. Executing SORT3

The System/3 compatible sort executes the $DSORT (disk) and $TSORT (tape) programs. To run these programs, you should specify $DSORT or $TSORT as the program name in the OCL LOAD statement.

3.1.2. Operation Considerations

You need to allocate 20K (minimum) to 64K (maximum) of main storage for SORT3. Also, the allocation of a work file ($SCR1) on the FILE statement is necessary. If these are not assigned, the OCL processor interfaces with SORT3 to ensure that the memory requirement and work file allocation is satisfied. (See the SORT3 user guide for an in-depth look at SORT3.)

3.2. OCL TO JCL CONVERTER

Even though Sperry Univac offers an OCL processor as part of OS/3, you may find it beneficial to convert your OCL streams to OS/3 job control streams. As you become more accustomed to using job control, it may prove easier to have all of your procedures standardized under one control language. To help you convert your System/3 OCL to OS/3 JCL, Sperry Univac provides an OCL to JCL converter (JCLCON801). This converter translates syntactically correct OCL streams into usable JCL control streams.

To convert your OCL streams in JCL through the converter, they must first reside as modules in an OS/3 program library. Once there, they can be submitted to the converter. The converter cannot always resolve differences between OCL and JCL capabilities. If in the process of converting an OCL stream it detects areas that cannot be directly converted, these areas are left unconverted and a message warning of the unresolvable difference is issued. The converter outputs the converted control streams as modules to a program library that you specify.

For more detailed information on the operation of the OCL to JCL converter, refer to the OCL to JCL converter (JCLCON801) user guide.

3.3. CCP SCREEN SOURCE STATEMENT TO OS/3 SFS CONVERTER

This program converts Communications Control Program (CCP) screen source statements into a format usable by OS/3 Screen Format Services.

The program, CCPCON301, reads the System/3 screen source statements from an OS/3 library file; therefore, before you can convert the statements, you must copy them to an OS/3 library. You do this in the same way you copy language programs, using the COPYS3 utility.

The converted screen formats are ready for use in programs through the Screen Format Coordinator (SFC). If you want to modify the System/3 source statements further than can be done by the converter, use the Screen Format Generator (SFG).

Before you attempt this conversion, we recommend that you read the OS/3 screen format services user guide/programmer reference, UP-8802, and the CCPCON301 user guide, UA-0485.

4. Language Processors

4.1. GENERAL

The SPERRY UNIVAC Operating System/3 (OS/3) supports programming languages that are not only compatible with their System/3 counterparts but are also more powerful and versatile then those supported by System/3. If you have been using RPG, COBOL, or FORTRAN with your IBM system, you can quickly and easily resume your productive programming effort and even exceed your former capabilities under System/3. The conversion of programs written in these languages is a straightforward procedure that permits you to continue to operate in much the same manner as before conversion with the option to improve your programs with enhancements not available to you before.

To make your COBOL program conversion even easier, Sperry Univac makes available a program that automatically performs much of the required conversion. We'll discuss the program, COBTRN309, further in section 4.3.

The first step toward converting programs written in any of the languages is to transcribe your program source modules to OS/3 libraries using the procedure described in Section 2 of this manual. After you have transcribed your program modules, you then recompile each program using the appropriate language processor. You should find that nearly all of your programs are executable after you correct the compilation errors detected by the OS/3 processor. A flowchart showing the general procedure for program conversion is presented in Figure 4-1.

4.2. REPORT PROGRAM GENERATOR II

Since there is a high degree of compatibility between System/3 RPG II and OS/3 RPG II, most of your RPG programs can be recompiled on the OS/3 system and any compilation errors resulting from the minor differences can be corrected. You should have no difficulty in reaching full program productivity with your RPG applications.

To further simplify the transition of RPG programs, OS/3 offers you the option of compiling your RPG programs in System/3 mode thus further reducing the number of potential incompatibilities. Additionally, you can continue to use your original System/3 OCL streams because of the OCL processor incorporated as part of OS/3 software.

This discussion provides an overview of the procedure for compiling and executing your System/3 RPG programs under OS/3 and briefly lists the areas of incompatibility. For more detailed information refer to the OS/3 report program generator II user guide.

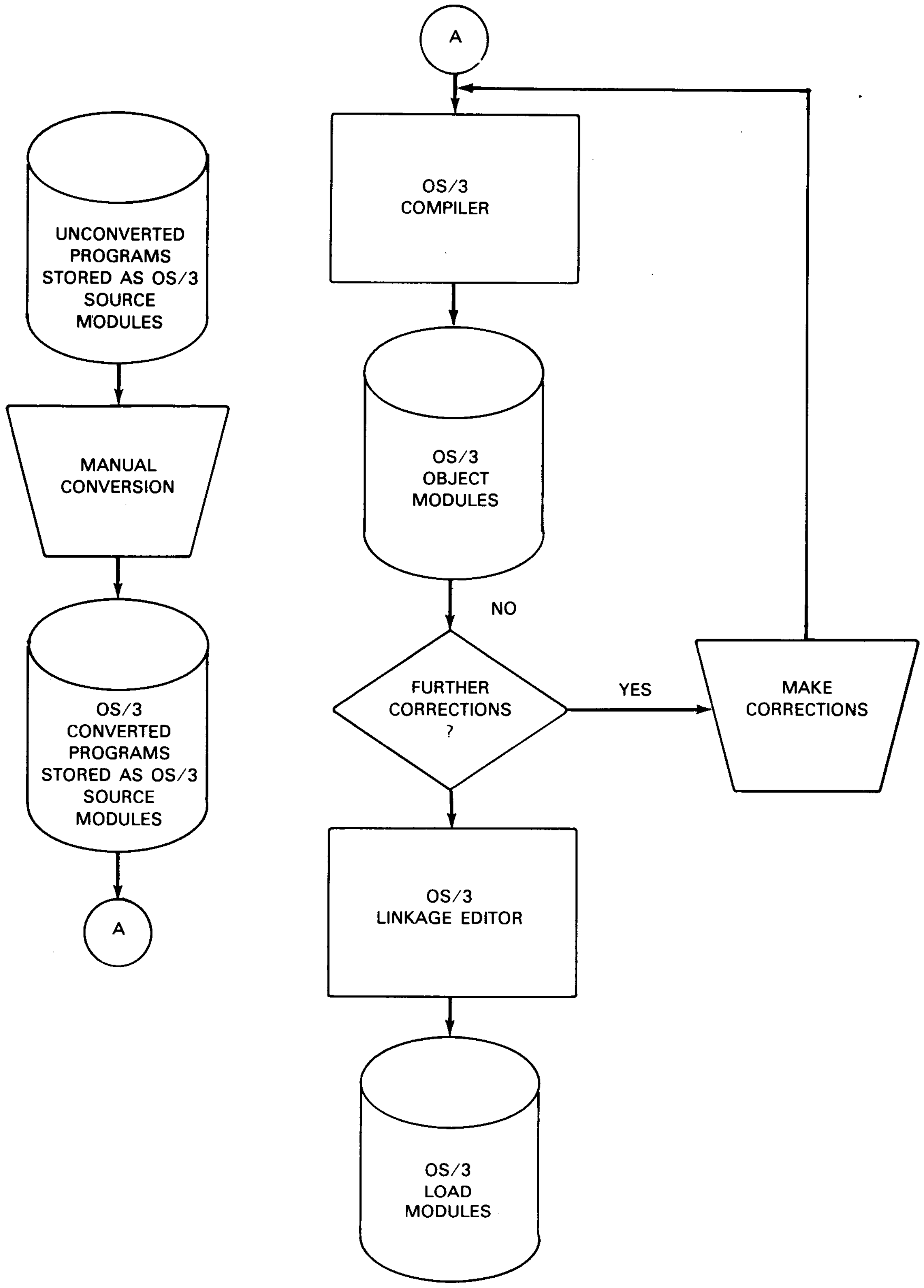

Figure 4—1. Conversion Process for Program Source Modules

- polling characters (columns 61 and 62)

  Polling characters are not supported. This entry should be replaced with a blank, otherwise a warning note will appear.

- addressing characters (columns 63 and 64)

  Addressing characters are not supported. This entry should be replaced with a blank, otherwise a warning note will appear.

- interspersed mode

  Interspersed mode is not supported. This entry should be replaced with a blank, otherwise a warning note will appear.

4.3. COBOL

The COBOL language supported by System/3 and that supported by OS/3 are very compatible and should present few problems in conversion. To make your conversion job easier, Sperry Univac makes available COBTRN309, a program that handles must of the conversion from System/3 COBOL to OS/3 COBOL automatically. COBTRN309 can greatly reduce the amount of time needed to convert your COBOL programs and thus get them running in your new OS/3 environment that much sooner. For detailed information on the use of COBTRN309, refer to the COBTRN309 user guide, UA-0485.

The COBOL language supported by OS/3 system adheres to the American National Standard COBOL, X3.23-1974 COBOL. In addition, OS/3 COBOL supports Federal Information Processing Standards Publication 21-1 (Appendix D). OS/3 COBOL supports the high processing level (level 2) of all incorporated American National Standards 1974 COBOL modules. The modules and processing levels supported are:

- nucleus,2
- table handling,2
- sequential I/O,2
- relative I/O,2
- indexed I/O,2
- sort/merge
- segmentation,2
- library,2
- debug,2
- interprogram communication,2
- communication,2

In addition, the OS/3 COBOL compiler contains extensions that enhance the capabilities of the language beyond the requirements of the standard. For more information on the capabilities of OS/3 COBOL, refer to the OS/3 1974 American National Standards COBOL programmer reference.

System/3 supports a subset of the American National Standard COBOL, X3.23-1968. The System/3 COBOL subset incorporates the following six modules of the full ANSI standard:

- nucleus,1NUC,1,2
- sequential access,1SEQ,1,2
- random access,1RAC,0,2
- table handling,2TBL,1,3
- library,1LIB,0,2
- segmentation,1SEQ,0,2

In addition, selected features of 2NUC 1,2; 2SEQ 1,2; and 2RAC 0,2 as well as certain IBM extensions to the basic modules have been incorporated into this COBOL compiler.

As you can see, the conversion of your IBM COBOL source programs to OS/3 COBOL is simple procedure that can be performed quickly and easily. Yet the conversion provides you with a more powerful programming language offering more capabilities than System/3 COBOL.

4.4. FORTRAN

The FORTRAN language supported by OS/3 FORTRAN IV offers a high degree of compatibility with System/3 FORTRAN and provides a more powerful and versatile language that extends the capabilities of FORTRAN programming. Because of the high degree of compatibility between the two languages, no converter is required. System/3 FORTRAN programs can be simply recompiled on the OS/3 FORTRAN IV compiler and the compilation errors caused by the few incompatibilities resolved.

OS/3 supports a FORTRAN IV compiler that includes the American National Standard FORTRAN, X3.9-1966 with extensions and the IBM 360/370 DOS FORTRAN IV languages. FORTRAN programs that conform to either of these specifications are accepted without change. OS/3 FORTRAN IV does not support a commercial subroutine package. If this package is not involved, little effort is required when converting your System/3 FORTRAN IV programs to OS/3 FORTRAN IV. For further information on the capabilities of the OS/3 FORTRAN IV language, refer to the OS/3 FORTRAN IV programmer reference.

System/3 supports American National Standard Basic FORTRAN, X3.10-1966. The compiler accepts as input, source programs written in IBM System 360 basic FORTRAN IV and 1130 basic FORTRAN IV. In addition, its extensions include DEBUG, IMPLICIT, relational IF, and explicit length specification for the INTEGER and REAL type statements. Also included are commercial subroutines similar to the 1130 commercial subroutine package.

However, you can include certain OS/3 job control statements in your OCL control stream to offer further enhancements to it and to effect a gradual transition to OS/3. The OS/3 product, the general editor, is available to perform this function from the workstation. The general editor enables you to edit the OCL streams from your OCL library file. To initialize the general editor, you enter the following command from the workstation:

```
EDT
```

A copy of the OCL library file is read into the EDT workspace file in main storage. All edit operations are accomplished here. The contents in the EDT workspace file is then written to an output file, which may be the original OCL library file or your own private library.

You can edit only disk and diskette files from the workstation.

For detailed information on the general editor and its capabilities, see the OS/3 general editor user guide/programmer reference.

5.1.2.1. Considerations for Creating OCL Streams at a Workstation

When you write your completed OCL stream out to a module in a library file, you must specify that the module is a type P module.

The first line in your OCL stream should be as follows:

```
PROC
```

(The letters PROC can be in any except the first space of the line).

The second line of your OCL stream should be as follows:

```
program-name NAME
```

(The program-name should begin on the first space of the line).

Your OCL stream should not be terminated with an // END statement.

5.2. SYSTEM/3 OCL STATEMENTS

The following paragraphs provide a description of the System/3 OCL statements. In some cases, you'll have to make some minor changes, but in most cases no changes are needed to any of your current statements. However, we will briefly explain each statement as to its function, and the functions of its associated parameters (and how they relate to the scheme of operations in OS/3). The conventions applicable to OCL statements when used in an OS/3 environment are described in 1.6.

5.2.1. Allocating Main Storage for a Job (PARTITION)

After the supervisor is loaded into the system, the remaining main storage space is available to OCL (job control) and your programs. As jobs are loaded and executed, the size of the available space varies. OCL (job control) assigns a portion of the available main storage to a job as it's initiated. When the job is completed, this space is returned to the system and becomes available to subsequent jobs.

When a job is initiated, a routine is loaded that determines the amount of main storage needed by the job and compares this requirement with the amount of main storage space available. If enough main storage is available, the job is scheduled; if not, the job is held until enough main storage becomes available.

By using the PARTITION statement, you can assign more main storage for your job than is automatically assigned, which it can use to improve or speed up job execution. This is, of course, contingent upon your job being written in a way that can use the additional main storage to an advantage.

The format of the PARTITION statement is:

$$\text{// PARTITION [minimum size]} \left[, \begin{Bmatrix} \text{maximum size} \\ \text{MAX} \end{Bmatrix} \right]$$

The minimum size parameter indicates the minimum bytes, in decimal, of main storage you want to assign to the job. This value does not include the prologue. The minimum size you can assign is 8192 decimal bytes. If the OCL processor determines that the amount of main storage specified by the minimum size parameter is insufficient for the job to run, the parameter is ignored.

The maximum size parameter specifies the maximum size of main storage to be allocated for the job. You should not ask for more main storage than your job can use, since the extra main storage may be needed for some other job. The *MAX* parameter, when specified, requests the maximum main storage on the machine. You must specify either the minimum or maximum size parameters when using the PARTITION statement.

The PARTITION statement is allowed anywhere in the OCL control stream except between a RUN statement and card data to the program.

To assign 12,000 bytes of main storage to your job, code the following:

```
// PARTITION 12ØØØ
```

NOTE:

The meaning of this statement differs in OS/3 OCL from that in System/3 OCL.

5.2.2. Indicating Program to be Run (LOAD)

The LOAD statement names the program to be executed, indicates the library where the program is located, and gives the switching priority. This statement must be placed before the RUN statement in the control stream.

The format of the LOAD statement is:

```
// LOAD programname ⎧,F1          ⎫[,switch-priority]
                    ⎪ F2          ⎪
                    ⎨ R1          ⎬
                    ⎪ R2          ⎪
                    ⎩ library-name⎭
```

The *programname* parameter identifies the program that's to be loaded from disk. Up to six characters may be used and the use of commas, apostrophes, and blanks is not permitted in the *programname* parameter.

The UNIT specification of the device containing the library is used by System/3 to define the library being accessed. *F1, F2, R1,* and *R2* are the unit designators indicating that the source and load modules and OCL procs are on either disk drive 1 (*F1*), fixed disk drive 2 (*F2*), removable disk drive 1 (*R1*), or removable disk drive 2 (*R2*). Under OS/3, these four UNIT codes are translated into the following four library files:

OCL Library Unit	OS/3 Library File
F1	$Y$LOD on SYSRES
F2	$Y$OCLOD on SYSRES
R1	$Y$LOD on SYSRUN
R2	$Y$OCLOD on SYSRUN

Using the LOAD statement with unit specification *F1* will access the system load library file ($Y$LOD) on the system resident device (SYSRES) to look for the program. $Y$LOD is an OS/3 permanent library that contains executable programs (load modules) that are generated as output from the linkage editor.

If you use unit specification *F2*, the OCL load library file ($Y$OCLOD) on SYSRES is the library file that's searched for the program. If you want to use this library file, you must first allocate it. This is not an OS/3 system file; you must allocate it yourself. You can do this by using the $MAINT routine's allocate function. Worth mentioning at this time are these two points:

1. What is known as the physical file name (LABEL) on the FILE statement in OCL is known as the file identifier in OS/3. The file identifier is a physical label that's used to identify a file.

2. In the OS/3 environment, all file identifiers on a given volume must be unique. That is, no two files on the same volume can have the same name.

By specifying *R1* as the unit specification, you indicate that the library file to be searched for the program is $Y$LOD, and it's located on the volume that contains the temporary job run library file ($Y$RUN) for the job. Each job that enters the OS/3 environment is assigned its own $Y$RUN file on the volume set aside for all the $Y$RUN job files. Each installation can have a separate volume containing the area set aside for all the $Y$RUN job files, or they can have it residing on SYSRES. This is determined when you perform the initial program load. If you specify *R1*, and the area for all the $Y$RUN job files is on its own volume, then this volume is searched for $Y$LOD. ($Y$RUN must be allocated since it does not exist on SYSRES.) If you specify *R1*, and the area for all the $Y$RUN job files is on SYSRES, then, by default, SYSRES would be searched for $Y$LOD.

By specifying *R2*, you indicate that $Y$OCLOD on the volume containing the job's $Y$RUN file is to be searched for the program (remember, you have to allocate $Y$OCLOD, since it's not an OS/3 system library). The same default (in respect to not having $Y$RUN files on a separate volume) applies.

You can also specify your own private load library name.

The optional task switching priority determines the order in which central processor control is passed from task to task. The number of user switching priorities varies from 01 (highest) to 60 (lowest). The supervisor establishes a switching priority queue to control the synchronization and rotation of tasks. If you do not specify a switching priority, the supervisor assigns the lowest priority available for the system.

You can also specify a relative priority value such as +3 or -3 to change the switching priority for a program specified in a particular job step with respect to the job's overall priority. For example, if a job is running at a priority of 7, you should specify a -3 priority in the // LOAD statement of a particular program within the job. That particular program would then run at a priority of 4.

To illustrate the use of the LOAD statement, assume you want to run the program CHECKS from the $Y$LOD library contained on the SYSRES volume. The required coding would appear as:

```
// LOAD CHECKS,F1
```

NOTES:

1. *If the volume containing the $Y$RUN job files is specified as the SYSRES device (determined at system IPL time), then specifying the parameter* R1 *is exactly the same as specifying* F1. *Likewise, specifying the parameter* R2 *is the same as specifying* F2.

2. *The format // LOAD * is not supported.*

5.2.3. Executing the Program (RUN)

The RUN statement is used to tell the operating system to execute the program named on the LOAD statement. It must be the last OCL statement in your control stream for a program. It must precede any card data to the program, and it is required for each program to be run. Card data for the program being run must immediately follow the RUN statement. (See 5.6 for more information.)

The format of the RUN statement is:

```
// RUN
```

It has no parameters.

NOTE:

The RUN statement does not cause immediate execution of the program as it does in a System/3 environment. The entire job stream is processed before any programs are executed.

The format of the NOHALT statement is:

```
// NOHALT [SEVERITY-{1}]
                    {2}
                    {4}
                    {8}
```

The *SEVERITY* parameter, if supplied, will be treated as a comment.

NOHALT is the default mode of processing for multiple programs submitted to OS/3 OCL.

The HALT and NOHALT statements affect only the job in which they are encountered and not other jobs being executed in the system.

5.2.12. Displaying Comments (*comment)

Comment statements to the operator can be displayed on the system console by inserting asterisk (*) statements into the control stream. These statements can be used to provide job setup instructions to the operator, or to simply explain the job.

The format of a comment statement is:

```
*comment
```

The comment can contain any character; there are no restrictions. The asterisk, however, must start in column 1, and the comment can start in any column after column 1.

As many comment statements as are needed can be placed anywhere in the OCL control stream, except between a RUN statement and card data to the program.

5.2.13. Displaying Comments and Halting (PAUSE)

The PAUSE statement is similar to the comment statement in that it's used to pass information to the operator. Additionally, it causes the processing of the job to come to a halt. This is especially helpful when you want the operator to peform some task (such as placing special forms in the printer for nonspooling system) before allowing the job to continue.

It is not necessary to specify the PAUSE statement for cases in which the FORMS/PRINTER or IMAGE statements are used to change forms or the print image. These statements generate their own job pause when the change is required to take place. This is especially true for a spooling environment where printing can be performed after job termination.

The format of the PAUSE statement is:

```
// PAUSE[comments]
```

The PAUSE statement can be used in conjunction with the comment statement to provide a list of instructions to the operator. By using both types of statements, rather than just PAUSE statements, the job will not halt after every comment line is printed, but only after the comment line printed by the PAUSE statement.

The PAUSE statement (just as the comment statement) can appear anywhere within the control stream, except between a RUN statement and card data to the program.

Whenever a PAUSE statement is processed, the following message is also displayed on the system console:

```
OCL PAUSE FOR JOB jobname
```

To continue processing, the operator only need press the MESSAGE WAITING key, enter the message number, and then press the TRANSMIT key.

NOTES:

1. *Any comments specified on the PAUSE statement will be displayed on the operator console before the PAUSE message.*
2. *All device allocations are done before the PAUSE statement is processed. Therefore, all your volumes must be mounted before you can issue the PAUSE to stop the job.*

5.2.14. Defining Data Files (FILE)

The FILE statement provides the information needed to define a data file on a disk, tape, diskette, card input, combined input/output card volumes, printer or punch files. The operating system uses the information supplied on the FILE statement to access the file. The FILE statement must be placed after the LOAD or CALL statement and must precede the RUN statement.

NOTE:

You may use the FILE statement to define multivolume disk, diskette, and tape files. For the special procedures necessary to define multivolume files, refer to 5.5.

The NAME parameter values for all the FILE statements for a given program must be unique and also must not be one of the following reserved names: $Y$RUN, $Y$SRC, $Y$LOD, $Y$OBJ, $Y$MAC, $Y$JCS, OCLF2, OCLR1, OCLR2, and PRNTR. PRNTR1 is a reserved name in a spooling system. The name PUNCH is reserved, but it may be used to change the punch device for a program to a diskette. After the program terminates, the diskette is reset to a card punch device.

NOTE:

If you wish to assign the same devices to more than one step of a multi-step job, the device identification on the UNIT parameter of the FILE statement must be specified in each job step, and the devices must be assigned in the same order in each job step as in the first job step.

5.2.14.1. Defining Disk Data Files

The FILE statement can be used to define a disk file.

The format of the disk FILE statement is:

```
// FILE NAME-{*filename},PACK-{name}[{(NS) }] ,UNIT-{F1}[(did)]:
             {filename }       {RES } [{(NOV)}]        {F2}
                               {RUN }                  {R1}
                                                       {R2}
                                                       {D1}
                                                       {D2}
                                                       {D3}
                                                       {D4}

    [,LABEL-{'character string'}]
    [       { filename         }]

    [,{TRACKS-number }]
    [ {RECORDS-number}]

    [,LOCATION-{cylindernumber            }]
    [          {cylindernumber/tracknumber}]

    [,RETAIN-{P}] [,DATE-{mmddyy}]
    [        {T}] [      {ddmmyy}]
    [        {S}]

    [,HIKEY-{'highest key fields allowed'        }]
    [       {P'highest packed key fields allowed'}]

    [,RECL-recordlength]

    [,BLKL-blocklength]      [,VERIFY-{YES}]
                             [        {NO }]

    [,ACCESS-{EXC }] [,OPEN-{ACCEPT}]
    [        {EXCR}] [      {INIT  }]
    [        {SRDO}] [      {EXTEND}]
    [        {SRD }] [      {RELOD }]
    [        {SUPD}]
    [        {SADD}]

    [,EXTENTS-nn]

 [,RECFM-{F }] [,LACE-nn]
 [       {V }]
 [       {D }]
 [       {FB}]
 [       {VB}]
 [       {DB}]

 [,RCB-{YES}]
 [     {NO }]

 [,VMNT-{ONE}]
 [      {NO }]
```

All the parameters are keyword parameters.

The *NAME* parameter provides the file name that your program uses to reference the file. This must agree with the file name from the file description specification if your program is written in RPG II. (In assembler language, it's the label field of the DTF; COBOL is the file name from the file description entry; FORTRAN is the device number from the READ or WRITE statement prefixed by FORT.) If more than one card is needed to complete the FILE statement, the *NAME* parameter should be on the first card. The * preceding the *filename* indicates that the file is read only protected.

It should be noted that scratch files required by certain software have different file names in System/3 and OS/3. These System/3 file names will be changed to the OS/3 equivalent by the OCL processor as follows:

System/3	OS/3
$SOURCE	$SCR1
$WORK	$SCR2
$WORK2	$SCR3
$WORKX	$SCR3

The *PACK* parameter is used to provide the name of the disk volume that contains the file (in OS/3, this is the volume serial number). (NOV) specifies that the volume is processed as a NOVOL mount and no job control volume checking is performed. (NS) specifies the volume is marked nonshareable and will not be shared between jobs in the system. The parentheses must be included and must immediately follow the name.

For example:

```
PACK-DISKØ1 (NOV)
```

By specifying RES for the pack name, you cause the OCL processor to substitute the volume serial number of the SYRES pack for the name RES. This allows the job stream to be independent of the actual volume serial number of the pack it will use.

Specifying RUN works in the same manner as RES except the volume serial number of the SYSRUN pack is used.

The *UNIT* parameter value indicates the type of System/3 disk pack you would have used for this file. (*R1, R2, F1,* and *F2* indicate an IBM 5444 disk pack; *D1, D2, D3,* and *D4* indicate an IBM 5445 disk pack.) Because these disk packs have different capacities, it is important to know which type you would have used in order to determine the amount of disk space required for the file in an OS/3 environment when the TRACK 1 parameter is given. The calculation for determining the space needed is therefore based on the value specified for the *UNIT* parameter and the value specified in the *TRACKS* parameter if provided. A value indicating the capacity of each type of IBM disk pack in 256 byte blocks (20 for 5445 and 24 for a 5444) has been set internally by OS/3.

The *(did)* parameter is used to request a specific disk drive for your disk pack and specifies the physical address assigned to the unit. This is a hexadecimal number defining the channel number, control unit address, and device number. The parameter must be enclosed in parentheses and must immediately follow the unit code.

The *LABEL* parameter indicates the 44-character physical label used to identify the file. In 5.2.2, we mentioned two points about the physical name of the file:

1. It's known as the file identifier in OS/3.

2. Each file name (file identifier) on a volume must be unique.

The value for the *LABEL* parameter can be specified in either of two ways:

■ LABEL-filename

  In this form, the first character must be alphabetic, and there cannot be any apostrophes, commas, or intervening blanks.

■ LABEL-'character string'

  This form is enclosed within apostrophes and can contain special characters. If you use an apostrophe as a special character, it must be coded as two apostrophes.

If you omit the *LABEL* parameter, then the file name you used on the *NAME* parameter is used, by default.

The *TRACKS* parameter value indicates the number of disk tracks you want assigned to the file. The OCL processor multiplies this value by the number of 256-byte blocks per track for the disk defined in the *UNIT* parameter (20 for 5445 or 24 for a 5444) to provide you with an equivalent number of blocks for OS/3 file allocation. (The number of blocks calculated is internally rounded up to the next higher number of cylinders since actual disk file allocation is done in terms of cylinders for OS/3 OCL.) If the *TRACKS* parameter is specified, the *RECORDS* parameter must not be specified.

The value you assign to the *RECORDS* parameter indicates how many records to assign to the file. This is equivalent to the number of 256-byte blocks. (This is then rounded up to the amount of cylinders.) Because OS/3 OCL assumes a record size of 256 bytes for space allocation, too much or too little space may be allocated. Values therefore may have to be changed when converting to OS/3 OCL. If the *RECORDS* parameter is specified, the *TRACKS* parameter must not be specified.

The *LOCATION* parameter is used to indicate the starting address of the file. You can specify the starting track (*tracknumber*) which, under OS/3, is assumed to be the starting cylinder; you can specify the starting cylinder (*cylindernumber*); or you can assign both the starting cylinder and track (*cylindernumber/tracknumber*). When you specify both the cylinder and track, the cylinder is specified first, and is separated from the track by a slash (/). The *tracknumber* is ignored. All OS/3 OCL files must start on a cylinder boundary.

The *RETAIN* parameter specifies the disposition of the file at the end of the job step. A permanent file (P) is created with an expiration date of 99/999. A temporary file (T) that is also the default, is normally used more than once. The designation of scratch (S) specifies that this file is to be scratched when the job step is completed. Under OS/3, once a file is scratched, all record of it is erased. Therefore, files cannot be reactivated by using the System/3 *RETAIN-A* parameter. If used, this parameter causes an error condition and your job will not be scheduled.

The *DATE* parameter provides a creation date for a file, in either the form mmddyy (month-day-year) or ddmmyy (day-month-year). OS/3 does not support different versions of files with the same name. Therefore, this parameter is not used by the OCL processor.

The *HIKEY* parameter specifies the highest key to be written on multivolume indexed files. This parameter is supported for compatibility and is ignored by the OCL processor.

The *RECL* parameter indicates how many bytes are in each logical record in the file, up to a maximum of 32,767. This parameter overrides the record length defined in the program; therefore, be sure not to specify a length greater than what was specified in your program.

The *BLKL* parameter for disk indicates the number of bytes in every physical block on the disk, in the range of 1 to 32,767. The block length must be a multiple of the record length. Also, this parameter overrides the block length defined in your program.

The *VERIFY* parameter indicates all data written is to be checked. This parameter is supplied for compatibility and is ignored by the OCL processor.

The *ACCESS* parameter specifies the shareability status of the file between the current job and others in the system. This parameter has six different options:

- EXC

  Specifies the exclusive read/write use of this file. No other jobs can access this file while the file is being used.

- EXCR

  Specifies the read/write use of the file and also allows other jobs to read from this file while it is being used.

- SRDO

  Specifies only the read function is allowed for this file as well as other jobs accessing it. No writing to this file is allowed.

- SRD

  Specifies only the read function is allowed for this file, but other jobs accessing it can both read and write to it.

- SUPD

  Specifies both read and write functions may be performed by the job as well as by other jobs accessing this file. No file extension is allowed.

The *OPEN* parameter determines how the file is to be opened. *ACCEPT* indicates that the DTF specification for the file is obtained from format 1 and format 2 labels in the VTOC. Coding *RELOD* and *INIT* have the same effect. Data is written to the file starting at the beginning of the file, with any existing data lost. *EXTEND* allows you to add information to the end of your file.

The *EXTENTS* parameter reserves additional extent storage space for the file in the job's prologue region. If the parameter is omitted, the default is space for eight extents.

The *RECFM* parameter indicates the format of the input or output records, as follows:

- F

  Fixed-length, unblocked records. If this is used, the minimum value for the *RECL* parameter is 18. Blocks and records are equal in size.

- V

  Variable-length, unblocked records. Blocks and records are equal in size; however, the record size can vary in length. The *RECL* parameter must include four bytes for the record description.

- D

  Variable-length, unblocked ASCII records (D-type). The *RECL* parameter must include four bytes for the record description. Ignored by the OCL processor.

- FB

  Fixed-length, blocked records. All records have an equal length, and all blocks have an equal length. If this is used, the minimum value for the *RECL* parameter is 18.

- VB

  Variable-length, blocked records. Each block contains records of variable lengths. The *RECL* parameter must include four bytes for the record description.

- DB

  Variable, blocked ASCII records (D-type). The *RECL* parameter must include four bytes for the record description. Ignored by the OCL processor.

The *RECFM* parameter overrides the format specified in your program.

The *LACE* parameter applies the record interlace technique to sequentially processed input or output files. Lacing permits you to access successive blocks within a predetermined interval of time. If this interval is less than the time the disk takes to complete a single revolution, you will retrieve more than one block per disk revolution. Record interlace, therefore, reduces the effect of rotational delay on your overall disk processing time. Only digits are accepted.

The *RCB* parameter specifies the record control byte, which is used to indicate that a record has been logically deleted from a file. This parameter applies only to newly created MIRAM disk files. For more specific information, see the data management user guide.

The *VMNT* parameter specifies how multivolume files are to be processed. *ONE* indicates that the file is to be processed one volume at a time. This method is used when the file is being processed sequentially. *NO* indicates that the file is to be processed with all volumes online at the same time. This method is used when the file is being processed using random access. The default value of this parameter is *ONE.*

NOTE:

The default value indicated here is valid only for running OCL streams. The default value when running OS/3 job control streams is different.

5.2.14.2. Defining Tape Data Files

The FILE statement can also be used to define a tape file by using a different set of parameters.

The format of the tape FILE statement is:

```
// FILE NAME-{*filename},UNIT-{T1}[(did)][,REEL-{name [{(NS)  }]}]
             {filename }       {T2}                {     [{(NOV) }]}
                               {T3}                {     [{(PREP)}]}
                               {T4}                {               }
                                                   {NL  [,n]       }
                                                   {NS  [  ]       }
                                                   {BLP [  ]       }

     [,LABEL-{filename          }][,DATE-{mmddyy}][,RETAIN-nnn]
     [       {'character string'}][      {ddmmyy}]

     [,BLKL-blocklength][,RECL-recordlength][,RECFM-{F }][,END-{LEAVE }]
                                             [       {V }][     {UNLOAD}]
                                             [       {D }][     {REWIND}]
                                             [       {FB}]
                                             [       {VB}]
                                             [       {DB}]
```

The *SEQNUM* parameter defines the file sequence number of the file to be accessed if more than one file resides on a tape. The *number* value can be from 1-9999. The *X* value parameter indicates the tape has been pre-positioned to where processing is to start.

The *BLKNUM* parameter specifies that the file is not to include block numbers. This parameter should be used when the system supports block numbering but you do not want it for this file.

The *OPEN* parameter determines how the file is to be opened. *ACCEPT* indicates that the DTF specification for the file is obtained from format 1 and format 2 labels in the VTOC. Coding *RELOD* and *INIT* have the same effect. Data is written to the file starting at the beginning of the file with any existing data in the file lost. *EXTEND* allows you to add information to the end of your file.

NOTE:

Tapes may be prepped using the PREP option on the REEL *parameter.*

5.2.14.3. Defining Diskette Data Files

The FILE statement can also be used to define a diskette data file.

The format of the diskette FILE statement is:

```
// FILE NAME-{*filename},UNIT-{K1}[(did)],PACK-name[(NS) ]
            {filename }      {K2}                  [(NOV)]
                             {K3}
                             {K4}

        [,LABEL-{filename          }]
        [       {'character string'}]
        [{,RECORDS-number of}]
        [{,TRACKS-number of }]
        [,RETAIN-{T}]
        [        {S}]
        [        {P}]
        [,RECL-recordsize]
        [,BLKL-blocksize]
        [,OPEN-{ACCEPT}][,EXTENTS-nn]
        [      {INIT  }]
        [      {EXTEND}]
        [      {RELOD }]
    [,RECFM-{F }]
    [       {V }]
    [       {D }]
    [       {FB}]
    [       {VB}]
    [       {DB}]
    [,VMNT-{ONE}]
    [      {NO }]
```

(continued)

or (this form is for System/3 compatibility)

```
// FILE NAME-filename, UNIT-3741
         [,RECL-recordsize]
         [,LABEL-{filename             }]
         [       {'character string'   }]
         [,PACK-name[(NOV)]]
         [{,RECORDS-number of}]
         [{,TRACKS-number of }]
         [,RETAIN-{T}]
         [        {S}]
         [        {P}]
         [,BLKL-blocksize]
         [,OPEN-{ACCEPT}] [,EXTENTS-nn]
         [      {INIT  }]
         [      {EXTEND}]
         [      {RELOD }]
   [,RECFM-{F }]
   [       {V }]
   [       {D }]
   [       {FB}]
   [       {VB}]
   [       {DB}]
```

All the parameters are keyword parameters.

The *NAME* parameter provides the file name your program uses to reference the file.

The *UNIT* parameter identifies this statement as a diskette file statement. The *(did)* parameter indicates that specific diskette drive is to be used for the diskette volume and specifies the physical address assigned to the unit. The physical address is a hexadecimal number defining the channel number, control unit address, and unit address. It must be enclosed in parentheses and immediately follow the unit code.

NOTE:

If the number of drives indicated by the UNIT *parameter is greater than one, two diskette drives are reserved for the file. OS/3 software has a two diskette drive per file capacity.*

The *PACK* parameter specifies the volume serial number of the diskette containing your file.

When *(NOV)* is specified, the volume is processed as a NOVOL mount and no job control volume checking is performed.

The *(NS)* parameter specifies the diskette is not to be shared between jobs. This is the default for diskette volumes and if specified, it is ignored by the OCL processor.

If *UNIT-3741* was specified and the *PACK* parameter was omitted, the first six characters of the *NAME* parameter are used as the pack name. It is treated as a NOVOL mount.

The *LABEL* parameter specifies a 44-character name by which your file is identified on the diskette. If this parameter is omitted, the *NAME* specification is assumed.

The value for the *LABEL* parameter can be specified in either of two ways:

- LABEL-filename

  In this form, the first character must be alphabetic and there cannot be any apostrophes, commas, or intervening blanks.

- LABEL-'character string'

  This form is enclosed within apostrophes and can contain special characters. If you use an apostrophe as a special character, it must be coded as two apostrophes.

The *RECORDS* parameter specifies the amount of space you are allocating for the diskette file. This value is the number of 128-byte blocks you are allocating for your file. If the *RECORDS* parameter is specified, the *TRACKS* parameter must not be specified.

The *TRACKS* parameter specifies the number of diskette tracks to be allocated to the file. It is converted into the number of 128-byte blocks by multiplying the number of tracks requested by 26 (the number of 128-byte sectors in a diskette track). When the *TRACKS* parameter is specified, the *RECORDS* parameter must not be specified.

The *RETAIN* parameter specifies the disposition of the file at the end of the job step. A permanent file (P) is created with an expiration date of 99/999. A temporary file (T), which is also the default, is normally used more than once. The designation of scratch (S) specifies that this file is to be scratched when the job step is completed.

The *RECL* parameter specifies the number of bytes in a logical record for the diskette and can be any number between 1 and 128. This parameter overrides the length defined in your program.

The *BLKL* parameter specifies the number of bytes in the physical block on the diskette. This parameter overrides the length defined in your program.

The *OPEN* parameter determines how the file is to be opened. *ACCEPT* indicates that the DTF specification for the file is obtained from the format 1 and format 2 labels in the VTOC. Coding *RELOD* and *INIT* have the same effect. Data is written to the file starting at the beginning of the file with any existing data in the file lost. *EXTEND* allows you to add information to the end of your file.

The *EXTENTS* parameter reserves additional extent storage space for the file in the job's prologue region. If this parameter is omitted, the default is space for eight extents.

The *RECFM* parameter indicates the format of the input or output records, as follows:

- F

  Fixed-length, unblocked records. If this is used, the minimum value for the *RECL* parameter is 18. Blocks and records are equal in size.

- V

  Variable-length, unblocked records. Blocks and records are equal in size; however, the record size can vary in length. The *RECL* parameter must include four bytes for the record description.

- D

  Variable-length, unblocked ASCII records (D-type). The *RECL* parameter must include four bytes for the record description. Ignored by the OCL processor.

- FB

  Fixed-length, blocked records. All records have an equal length, and all blocks have an equal length. If this is used, the minimum value for the *RECL* parameter is 18.

- VB

  Variable-length, blocked records. Each block contains records of variable lengths. The *RECL* parameter must include four bytes for the record description.

- DB

  Variable, blocked ASCII records (D-type). The *RECL* parameter must include four bytes for the record description. Ignored by the OCL processor.

The *RECFM* parameter overrides the format specified in your program.

The *VMNT* parameter specifies how multivolume files are to be processed. *ONE* indicates that the file is to be processed one volume at a time. This method is used when the file is being processed sequentially. *NO* indicates that the file is to be processed with all volumes online at the same time. This method is used when the file is being processed using random access. The default value of this parameter is *ONE*. When you are defining diskette files, if you are using diskettes prepped in the data-set-label (DSL) mode, you must take the default value. If you are using diskettes prepped in the format label mode, you must override the default value and enter *VMNT - NO.*

System/3 Model 10 does not require a FILE statement to define a diskette file to the system. The Model 15 requires a limited one for your programs using the diskette. The OCL processor supplies Model 15 diskette specifications for the $COPY and $KCOPY programs if diskettes are requested and diskette FILE statements were not provided. Since very little volume and file checking is done for the Model 15 type statements, it is recommended that you change your statements to the OS/3 OCL diskette FILE specification.

5.2.14.4. Defining Combined Card Files

The FILE statement can also be used to define a combined card file. A combined card file is a card input/output file in which the cards are punched as they are read.

The format of the combined card FILE statement is:

```
// FILE NAME-filename,UNIT-COMB[(did)],[,LABEL-{filename          }]
                                       [       {'character string'}]

    [,RECL-recordlength][,BLKL-blocklength]
```

The *NAME* parameter supplies the file name to be used by the program. Just like the *NAME* parameter for the other files (disk, tape, diskette), it must agree with the internal name assigned in your program.

The *UNIT* parameter must be specified to identify the file as a combined file. To use the combined card data file specification, you must have a card punch on your system that also reads cards. The card data must be placed into the punch prior to program execution. This differs from normal card files that are placed after the program's RUN statement.

The *(did)* parameter indicates that a specific combined file is to be used and specifies its physical address. This is a hexadecimal number defining the channel number, control unit address, and the device number. This parameter must be enclosed in parentheses and immediately follow the *UNIT-COMB* designation.

The *LABEL* parameter indicates the physical name identifying the file. For combined files, this parameter is meaningless. If this parameter is omitted, the file name used on the *NAME* parameter is used.

The *RECL* parameter indicates the number of bytes in each logical record in the file. This parameter overrides the record length defined in your program.

The *BLKL* parameter indicates the number of bytes in each physical block in the file. This value must be the same value as specified on the *RECL* parameter. The *BLKL* parameter overrides the block length defined in your program.

5.2.14.5. Defining Card Data Files Not Contained in the Control Stream

The FILE statement can also be used to define card data files not contained in the control stream. The card FILE statement should only be used when the program being executed does not expect card data to be present in the control stream.

RPG II programs compiled in System/3 mode and the OCL COPY routine must not contain this statement.

The format of the card FILE statement is:

```
// FILE NAME-filename, UNIT-⎧C1⎫[(did)]⎡,LABEL-⎧filename          ⎫⎤
                           ⎨C2⎬       ⎣       ⎩'character string'⎭⎦
                           ⎨C3⎬
                           ⎩C4⎭
        [,RECL-recordlength][,BLKL-blocklength]
```

The *NAME* parameter indicates the file name used by the program. Just like the *NAME* parameter used for the other files (disk, tape, diskette), the name must agree with the name you used to reference the file in your program.

The *UNIT* parameter identifies the statement as a card file statement.

The *(did)* parameter indicates that a specific card reader is to be used for the file and specifies the physical address of the card unit. This is a hexadecimal number defining the channel number, control unit address, and the device number. This parameter must be enclosed in parentheses and immediately follow the unit code.

The *LABEL* parameter indicates the physical name identifying the file. For card files in a nonspooling environment, this parameter is meaningless. For an input spooling system, the physical name is the name of the spooled card file (name specified on the DATA statement, see 5.3.6). If the *LABEL* parameter is omitted, the file name specified on the *NAME* parameter is used.

The *RECL* parameter indicates the number of bytes in each logical record in the file. This parameter overrides the record length defined in your program.

The *BLKL* parameter indicates the number of bytes in each physical block in the file. This value must be the same value as specified in the *RECL* parameter. This parameter overrides the block length defined in your program.

5.2.14.6. Defining Printer Files

The FILE statement can be used to define a file that contains output to a printer.

The format of the printer FILE statement is:

```
// FILE NAME-filename, UNIT-PR[(did)]
        [,RECL-recordsize][,BLKL-blocksize]
        ⎡,LABEL-⎧filename          ⎫⎤
        ⎣       ⎩'character string'⎭⎦
```

The *NAME* parameter provides the file name your program uses to reference the file.

The *UNIT* parameter identifies the statement as a printer file statement. The *(did)* parameter indicates that a specific printer is to be used and specifies the physical address of the unit. This parameter is a hexadecimal number defining the channel number, control unit address, and the device number. The *(did)* parameter must be enclosed in parentheses and immediately follow the unit code.

The *RECL* parameter indicates the number of bytes in each logical record, up to a maximum of 32,767. This parameter overrides the record length defined in your program.

The *BLKL* parameter indicates the number of bytes in each physical block in the file. This parameter overrides the block length defined in your program.

The *LABEL* parameter is used for compatibility among file statements. If supplied, it is ignored.

5.2.14.7. Defining Punch Files

The FILE statement can be used to define a file that contains output to a punch.

The format of the punch FILE statement is:

```
// FILE NAME-filename, UNIT-PU[(did)]
          [,RECL-recordsize][,BLKL-blocksize]
          [,LABEL-{filename            }]
                  {'character string'}
```

The *NAME* parameter provides the file name your program uses to reference the file.

The *UNIT* parameter identifies the statement as a punch file statement. The *(did)* parameter indicates that a specific punch is to be used and specifies the physical address of the unit. This is a hexadecimal number defining the channel number, control unit address, and the device number. The *(did)* parameter must be enclosed in parentheses and immediately follow the unit code.

The *RECL* parameter indicates the number of bytes in each logical record. This parameter overrides the record length defined in your program.

The *BLKL* parameter indicates the number of bytes in each physical block in the file. This parameter overrides the block length defined in your program.

The *LABEL* parameter is used for compatibility among file statements. If supplied, it is ignored.

5.2.15. Calling OCL Procedures (CALL)

The CALL statement is used to merge stored OCL statements with OCL statements entering the system through the job stream. This is not a physical merge; the statements called from the library are inserted logically into the control stream via the OCL processor.

The format of the CALL statement is:

$$\texttt{// CALL procedurename,}\begin{Bmatrix}\texttt{R1}\\\texttt{R2}\\\texttt{F1}\\\texttt{F2}\end{Bmatrix}$$

The *procedurename* parameter is used to supply the name that you used when you stored the procedure. The procedure is located on the library defined by the next parameter, (*R1, R2, F1, F2*). Up to eight characters may be used, but the use of commas, apostrophes, and blanks is not permitted.

Coding *R1* indicates the procedure is stored in the system library file ($Y$LOD), and it's on the volume containing the job's $Y$RUN file. By coding *R2*, you indicate that the procedure is stored in $Y$OCLOD, and it's located on the volume containing the job's $Y$RUN file. If you code *F1*, $Y$LOD on SYSRES contains the procedure. The coding of *F2* indicates the procedure is stored on SYSRES, in $Y$OCLOD. (See 5.2.2.)

To illustrate the use of the CALL statement, assume you have placed OCL statements in a procedure module called PAYROL in the F2 library ($Y$OCLOD on SYSRES) using the $MAINT library routine. To run this set of OCL statements, you would use the following CALL statement:

```
// CALL PAYROL,F2
```

NOTE:

If the volume containing the $Y$RUN job files is defaulted to the system resident device (determined at system IPL time), then specifying R1 *is the same as specifying* F1. *Likewise, specifying* R2 *is the same as specifying* F2.

5.2.16. Grouping Related Job Steps (JOB)

The JOB statement allows you to group together related job steps to ensure they are run sequentially. The next step of a job is not initiated until the previous job step has successfully completed. The JOB statement may precede the first LOAD or CALL statement. It cannot be used in a procedure.

- SIMULATE

```
// SIMULATE{ON }
           {OFF}
```

This statement must not appear between a load or CALL statement and a RUN statement.

5.3. JCL STATEMENTS SUPPORTED BY OCL

The following paragraphs give an explanation of the current JCL statements supported by OS/3 OCL.

5.3.1. Making Temporary Changes to a Load Module (ALTER)

The ALTER control statement permits you to make minor temporary changes in up to eight bytes of a load module to see if the changes have the desired effect before these changes are made permanent, because recompiling and linking are time consuming. The ALTER statement must follow a LOAD or CALL statement and precede a RUN statement. As many ALTER job control statements as you need to change the module are grouped between the LOAD or CALL statement and the RUN statement.

The format of the ALTER job control statement is:

```
//[symbol] ALTER [phase-name][,address][,change][,{RESET}]
                                                  [ {ORG  }]
```

The *phase-name* parameter is either the 8-alphanumeric character name of the phase assigned by the linkage editor or the 1- to 6-alphanumeric-character alias name of the phase. If you omit this parameter, the last-phase name used on an ALTER job control statement in this job step is used.

The *address* parameter is the 1- to 5-digit hexadecimal starting location address where the changed information is to be stored. This is in relation to the first byte of the phase area. If you omit this parameter and an address is required, an address of zero is used. An address is not required when *RESET* is used as the fourth parameter.

NOTE:

If the address given is invalid, a change does not take place.

The actual information to be placed in the phase is specified with the *change* parameter. You can specify it in either EBCDIC or hexadecimal. EBCDIC information takes the form *C'c...c'*. The maximum number of characters is eight (eight bytes). Hexadecimal information takes the form *X'c...c'*. The maximum number of characters is 16 (eight bytes). If you omit the *change* parameter, no modification is made for this ALTER job control statement alone, but the information it does contain, such as phase name, is passed to subsequent ALTER job control statements.

The *ORG* parameter indicates that the address specified in the *address* parameter should be added to all the addresses on succeeding ALTER job control statements, until one with a *RESET* parameter or a different phase name is encountered.

The *RESET* parameter resets the alter mode indicator to its original status. If you omit all other parameters of this ALTER job control statement, control is returned to your program.

Once an ALTER job control statement is encountered, each and every phase of the load module expects an ALTER job control statement. This the reason for the *RESET* parameter. It indicates that no other ALTER job control statements are in the control stream.

Consider these examples:

```
// ALTER TSTPGMOO
// ALTER ,4361,X'FAF3F9'
// ALTER ,4700,X'F8'
// ALTER ,,,RESET
```

If a *RESET* parameter is specified, the information is passed along to the program execution phase. When the phase that had the *RESET* parameter specified is loaded for the first time, the option is reset so that no other phases will be altered. This saves time if a phase that is only loaded once is the only phase requiring alteration.

Suppose there is a phase named TSTPGMOO and it constantly needs changes according to weather conditions. The first and last ALTER job control statements could be placed permanently in the control stream, while the variable ALTER job control statements could be inserted as needed. In the preceding example, the information contained in addresses 4361 and 4700 is changed.

5.3.2. Selecting Software Features (OPTION)

The OPTION control statement permits you to select optional software features whenever you wish. They are effective only during the job step in which they are specified. The OPTION control statement must follow a LOAD or CALL statement and precede a RUN statement.

The format of the OPTION job control statement is:

```
//[symbol] OPTION p-1[,...,p-n]
```

As you can see, you can specify as many features as desired, as long as they're separated by commas (there can't be any spaces). For information on the specific features available through the job control OPTION statement, refer to the job control user guide, UP-8065 (current version).

DELETION

Pages 5-43 through 5-46 have been deleted.

DELETION

Page 5-47 has been deleted.

5.3.3. Defining the Software Facilities (SFT)

OS/3 automatically loads the data management modules needed by your job. However, if you have written your own shared-code modules and they are not stored in $Y$LOD or $Y$RUN, you use the SFT job control statement to identify these modules to the system. The SFT statement tells the OCL processor that load modules not in either $Y$LOD or $Y$RUN, or on the volume containing the job's $Y$RUN file, are needed by a particular job.

For more information on the SFT job control statement, refer to the job control user guide, UP-8065 (current version).

DELETION

Pages 5-49 and 5-50 have been deleted.

DELETION

Page 5-51 has been deleted.

5.3.4. Restarting a Job (RST)

The RST job control statement allows you to rerun a job from a specified checkpoint. You don't have to rerun the entire job, just the part that was never completed. The information required to build an RST control statement is displayed on the system console whenever a CHKPT macroinstruction is encountered in your program. So, to use this restart facility, you have to make use of the CHKPT macroinstruction, which is explained in the supervisor user guide. In general, all you have to do is place an RST job control statement in front of the original job control stream; then run the job.

The format of the RST job control statement is:

```
//[symbol] RST file-name,checkpoint-id,number
   [,jobname-library unit[(rename)][,pri]]
   [,key-1=val-1,...,key-n=val-n]
```

The file that contains the checkpoint records must be defined in a FILE statement in the step being restarted. The *file-name* parameter of the RST job control statement must agree with the *NAME* parameter on the FILE statement for that file.

Every time a checkpoint record is displayed on the system console, a checkpoint number is given. When you want to restart the job, you use this checkpoint number as the *checkpoint-id* parameter.

The *number* parameter specifies the number of the job step within the job where the restart should begin.

The *jobname* parameter specifies the name of the job if the job is stored in a library. The library-unit identifier indicates the library. If a job with the same name is already scheduled for execution and you want to restart the stored job, you specify an alternate name for the job to be restarted by using the (*rename*) parameter (one to eight alphanumeric characters). Remember, only one job at a time can use a particular name. When you use the (*rename*) parameter, you must use the *jobname* parameter, and they are not separated by a comma, but by a parenthesis.

The *pri* parameter is the priority at which a restarted job is to be scheduled. This is either *P* for preemptive, *H* for high, or *N* for normal. The default is normal. This overrides the priority of the operator console command, or the OS/3 JOB statement.

The *key=val* parameters represent keywords and their values that may be referenced like the parameters of a GBL job control statement (5.3.19.). The effect of these parameters is as if a GBL job control statement was inserted as the first job control statement of the job. The total length of the value for the parameters cannot exceed 44 characters.

Since bits 0 and 1 were already set by the first SET UPSI job control statement and we want them left on, we code an X in these positions, and code a 1 to set bit 2. Since bit 7 is to be turned off, we code a 0 in this position; otherwise the 1 from the first SET UPSI job control statement would still be effective.

NOTE:

The OS/3 SET UPSI job control statement can be used interchangeably with the System/3 SWITCH statement. Both statements have equivalent functions.

5.3.12.3. Setting the Communications Region (SET COMREG)

The communications region is a 12-byte field in the job preamble that passes information from one job step to the next. For instance, assume your job has two job steps. The first job step generates input for the second. But, if this input is incorrect, you don't want to run the second job step. In the program for the first job step, you insert a routine that checks the validity of the output, and if it's incorrect, writes a code in the communications region. Then, in the program for the second job step, you insert another routine that checks the communications region. If the code is there, transfer control directly to the end of the job.

Once you place these routines in your programs, they are there permanently unless you remove the routines and recompile the programs. It may just happen that sometime you want to run the second job step even if the first job step was wrong (a test). Here is where you would use the SET COMREG job control statement. This allows you to change the code in the communications region.

The format of the SET COMREG job control statement is:

```
//[symbol] SET COMREG,char-string
```

The *char-string* parameter specifies the 1 to 12 EBCDIC characters or the 2 to 24 hexadecimal characters (even amounts only) to be stored in the communications region. It is stored left-justified, and any unspecified rightmost characters remain unchanged. Specify hexadecimal characters as *X'ccc...cc'* and EBCDIC characters as *C,'ccc...cc'*.

At the beginning of the job, the communications region is set to 0's.

Let's say you wanted the hexadecimal code of E2 E3 D6 D7 to be stored in the first four bytes of the communications region; it would be coded as:

```
// SET COMREG,X'E2E3D6D7'
```

5.3.13. Issuing System Commands (CC)

The CC job control statement allows you to issue OS/3 system console and workstation commands, with their associated parameters, from within a job control steam. Because there are many system commands, we will not attempt to discuss each one here. You can find the formats and descriptions of system console commands in your operations handbook. Workstation commands are described in the interactive service commands and facilities user guide/programmer reference. The format of the CC statement is:

```
//[symbol] CC command{command                   }
                     {'command and parameters'}
```

When enclosed in single quotes, any system console or workstation command and parameters can be specified in the CC statement. When the command has no associated parameters or when you do not specify any parameters, the quotes are not used.

Let's say you want to release a job (JOB1) that's being held as the result of a HOLD system command. If you specify the BEGIN command in a CC job control statement, you can include this statement in the job you're going to run. JOB1 will be released when this statement is processed (at your job's execution time). You would code the CC statement as follows:

```
// CC 'BE JOB1'
```

Suppose you wanted to initiate the general editor from a job control stream. The worksation command for the general editor is simply EDT. Because there are no parameters, you'd code the CC statement as follows:

```
// CC EDT
```

Whenever parameters are specified with a command, the total number of characters within the quotes cannot exceed 60.

The CC statement is examined for syntax errors by the run processor during job stream validation. If no syntax errors are found, the job is queued. The command and its associated parameters are sent to the system when the CC statement is encountered by the job step processor. The command is validated by the system independently of your job, so errors associated with satisfying commands do not terminate a job stream. If no EXEC statement follows a CC statement, the specified commands are acted upon prior to job termination.

NOTES:

1. *The following system console commands cannot be specified in the CC job control statement: MIX, SWITCH, AVR, REBUILD, SHUTDOWN, SYSDUMP, and all SET commands.*

2. *When the command string contains no blanks (other than the blank separating the command from its first parameter, you can precede the first parameter with a comma instead of enclosing the command and its parameters in single quotes. For example: // CC BE,JOB1*

DELETION

Pages 5-67 and 5-68 have been deleted.

Index

Term	Reference	Page
O		
OCL procedure use		
adding parameter	5.4.4	5—81
adding statement	5.4.5	5—81
changing parameter	5.4.2	5—80
deleting parameter	5.4.3	5—80
nesting programs/procedures	5.4.7	5—82
printing procedure call	5.4.6	5—81
standard procedure call	5.4.1	5—79
OCL to JCL conversion, SORT3	3.2	3—2
OCL use		
card data files	5.6	5—90
entry points	5.1	5—1
JCL statements	5.3	5—41
multivolume files	5.5	5—85
operations	5.8	5—98
procedures	5.4	5—79
stored control streams	5.7	5—96
System/3 statements	5.2	5—5
OCL utility routines		
copy ($COPY, $KCOPY, $DCOPY)	6.2	6—2
file delete ($DELET)	6.1	6—1
library maintenance ($MAINT)	6.3	6—18
OMIT parameter	6.2.5.4	6—16
Operating control language (OCL)		
benefits of use	1.5.1	1—6
conversion to JCL	1.5.3	1—8
description	1.5	1—6
library unit	5.2.2	5—7
operational considerations	5.8	5—98
stored control streams	5.7	5—96
utility routines description	1.3	1—5
Operator intervention (HALT/NOHALT)	5.2.11	5—18
Option processing, introduction (PARAM)	5.3.8	5—59
OPTION statement	5.3.2	5—42
Optional parameter use	1.6	1—11
Options, compiler (COMPILE)	5.2.6	5—10
OS/3 library file	5.2.2	5—7
OUTPTX and OUTPUT parameters	6.2.5.1	6—12
Output, card data files	5.6.2	5—94
Output specifications form incompatibilities	4.2.3	4—5

Term	Reference	Page
P		
PARAM statement	5.3.8	5—59
Parameter procedures		
adding	5.4.4	5—81
changing	5.4.2	5—80
deleting	5.4.3	5—80
See also OCL procedure use.		
Parameters, description		
keyword	1.6	1—8
optional	1.6	1—11
positional	1.6	1—9
Part of file format		
copy	6.2.4.4.	6—8
copy/print	6.2.4.5	6—8
print	6.2.4.6	6—9
PARTITION statement	5.2.1	5—6
PAUSE statement	5.2.13	5—19
Positional parameter description	1.6	1—9
Print a file format		
complete file	6.2.4.3	6—7
part of a file	6.2.4.6	6—9
Print character set specification (IMAGE)	5.2.7	5—12
Printer files definition	5.2.14.6	5—36
Printer form characteristics	5.2.8	5—14
Printing procedure call	5.4.6	5—81
Procedure call procedures		
adding missing parameter	5.4.4	5—81
adding statement	5.4.5	5—81
changing parameters	5.4.2	5—80
deleting parameters	5.4.3	5—80
nesting programs/procedures	5.4.7	5—82
printing	5.4.6	5—81
standard procedure	5.4.1	5—79
Processors, language	Section 4	
Program execution (RUN)	5.2.3	5—8
Program to be run indication (LOAD)	5.2.2	5—6a
Punch files definition	5.2.14.7	5—37
PUNCH statement	5.2.9	5—16

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.)	*(Revision No.)*	*(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

C ong line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT