UNISYS

PUBLICATIONS UPDATE

System 80

OS/3
System Service
Programs (SSP)
Operating Guide

UP-8841 Rev. 4-A

This Library Memo announces the release and availability of Update A to *System 80 OS/3 System Service Programs (SSP) Operating Guide*, UP-8841 Rev. 4.

This guide is a standard library item (SLI). It is part of the standard library provided automatically with the purchase of the product.

This update provides an index.

Copies of Update A are now available. You can order the update only, or the complete manual with the update, through your local Unisys representative. To receive only the update, order UP-8841 Rev. 4-A. To receive the complete manual, order UP-8841 Rev. 4.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists MBZ, MCZ, MMZ, M28U, and M29U	Mailing Lists MB00, MB01, and MBW (18 pages plus Memo)	Library Memo for UP-8841 Rev. 4-A RELEASE DATE: February 1989

UNISYS

PUBLICATIONS REVISION

System 80

OS/3
System Service
Programs (SSP)
Operating Guide

UP-8841 Rev. 4

This Library Memo announces the release and availability of *System 80 OS/3 System Service Programs (SSP) Operating Guide*, UP-8841 Rev. 4.

This guide is a standard library item (SLI). It is part of the standard library provided automatically with the purchase of the product.

Changes to this document for Release 12.0 include the following:

- Buffer analysis routine
- Disk dump/restore (DMPRST) routine enhancements providing multidisk volume dump/restore, tape restart, global file allocation, and file listing capabilities
- Multiple MIRAM file creation on a single-tape volume
- List software maintenance corrections (SMCLIST) function enhancements providing a full format option and the capability to print SMCs for specific program products
- PCTRAN enhancements
- Support for 8494 disk

All other changes in this document are corrections, deletions, or expanded descriptions applicable to items present in the software prior to this release.

Additional copies may be ordered through your Unisys representative.

Destruction Notice: This revision supersedes and replaces the *OS/3 System Service Programs (SSP) User Guide*, UP-8841 Rev. 3, released on Library Memo dated February 1984, and its Updates A through C. Please destroy all copies of UP-8841 Rev. 3 with its Updates and associated Library Memos.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists MBZ, MCZ, MMZ, M28U, and M29U	Mailing Lists MB00, MB01, and MBW (514 pages plus Memo)	Library Memo for UP-8841 Rev. 4 RELEASE DATE: October 1988

UNISYS

System 80 OS/3 System Service Programs (SSP)

Operating Guide

OS/3 Release 12.0

Priced Item

October 1988

Printed in U S America
UP-8841 Rev. 4

UNISYS

System 80 OS/3 System Service Programs (SSP)

Operating Guide

OS/3 Release 12.0

October 1988

Printed in U S America
UP-8841 Rev. 4

Priced Item

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the User Comments form at the back of this manual or remarks addressed directly to Unisys Corporation, to E/MSG Product Information, P.O. Box 500, M.S. E5-114, Blue Bell, PA 19424 U.S.A.

PAGE STATUS SUMMARY
ISSUE: Update A - UP-8841 Rev. 4
RELEASE LEVEL: 12.0 Forward

Part/Section	Page Number	Update Level
Cover		Orig.
Title Page/Disclaimer		Orig.
PSS	iii	A
About This Guide	v thru vii	Orig.
Contents	ix thru xx	Orig.
PART 1	Tab Breaker	Orig.
1	1 thru 12	Orig.
PART 2	Tab Breaker	Orig.
2	1 thru 32	Orig.
3	1 thru 116	Orig.
PART 3	Tab Breaker	Orig.
4	1 thru 45	Orig.
5	1 thru 4	Orig.
6	1 thru 22	Orig.
7	1 thru 10	Orig.
8	1 thru 34	Orig.
PART 4	Tab Breaker	Orig.
9	1 thru 46	Orig.
10	1 thru 8	Orig.
11	1 thru 4	Orig.
12	1 thru 64	Orig.
13	1 thru 7	Orig.
14	1 thru 5	Orig.
15	1 thru 21	Orig.
16	1, 2	Orig.
17	1	Orig.
PART 5	Tab Breaker	Orig.
Appendix A	1 thru 3	Orig.
Appendix B	1 thru 22	Orig.
Appendix C	1 thru 6	Orig.
Appendix D	1 thru 18	Orig.
Index	1 thru 15	A*
User Comments Form		
Back Cover		Orig.

Part/Section	Page Number	Update Level

* New pages

About This Guide

Purpose

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

Scope

This guide describes the OS/3 system service programs (SSP) and their effective use. The system service programs include the system librarians, the linkage editor, and the standard system utilities.

Audience

This guide is intended for the novice programmer with a basic knowledge of data processing, but with limited programming experience, and for the more sophisticated programmer whose experience is limited to systems other than Unisys systems.

Organization

This guide is divided into the following parts, each containing one or more sections:

Part 1. OS/3 System Service Programs Repertoire

Introduces you to the various system service programs through descriptions of their intended purposes within the OS/3 operating system, their capabilities, and the terms peculiar to their functional operation.

Part 2. The Librarians

Describes the functional characteristics of the system librarians relevant to you, the control statements you may use to direct their operation, and the various library mapping elements they are capable of producing.

Part 3. The Linkage Editor

Describes the functional characteristics, programming considerations, and control statements required to allow you to effectively use the linkage editor as it is intended to be used. Also describes the link-edit mapping data produced by the linkage editor for every load module it produces.

Part 4. System Utilities

Describes the utility programs provided by OS/3 to initialize disk, diskette, and tape volumes; copy disk, diskette, and tape volumes; patch system programs; print software maintenance correction listings; maintain the operating system; and perform buffer analysis.

Part 5. Appendixes

Appendix A presents canned job control streams and the document numbers where they are described.

Appendix B describes the code set components that, when combined in a particular sequence, make up a program source module, a macro/jproc source module, an object module, a load module, or a group code set module.

Appendix C describes the SAT library structures.

Appendix D describes the file transfer utility (PCTRAN) that permits the transfer of files between a Unisys personal computer (PC) and the Unisys System 80.

Related Product Information

To fully understand and appreciate the functions performed by the system service programs, you should be familiar with the information contained in current versions of the following Unisys publications. The degree of familiarity required varies with the product in question. For example, the linkage editor user has to be familiar with almost all of the documents. On the other hand, those using the librarian and system utilities need only a few of them.

System Service Programs Programming Reference Manual **(UP-8842)**

Job Control Language Programming Guide **(UP-9986)**

System Messages Reference Manual **(UP-8076)**

1974 American Standard COBOL Programming Reference Manual **(UP-8613)**

Consolidated Data Management Programming Guide **(UP-9978)**

Consolidated Data Management Macroinstructions Programming Guide **(UP-9979)**

Supervisor Macroinstructions Programming Reference Manual (UP-8832)

Integrated Communications Access Method (ICAM) Programming Reference Manual **(UP-9749)**

Installation Guide (UP-8839)

Interactive Services Operating Guide (UP-9972)

Menu Services Technical Overview (UP-9317)

Contents

Section 3. Librarian Control Statements

Section 5. Programming Considerations

Section 6. Control Statements

Section 7. Link-Edit Map

Section 8. Program Examples

PART 4. SYSTEM UTILITIES

Section 9. Initialize Disk Routine (DSKPRP)

PART 5. APPENDIXES

Appendix A. Canned Job Control Streams

Appendix B. Code Set Components

Appendix C. SAT Library Structure

Appendix D. File Transfer Utility (PCTRAN)

Figures

Tables

Section 1
Introduction

1.1. General Information

The system service programs (SSP) are those programs required to support the operation and organization of the operating system in which your problem programs are to be executed. These programs allow you to construct and reorganize the program libraries in your system, create program modules for execution in your system, initialize tape and disk volumes for the storage of your program and data files, and obtain printouts of main storage.

The system service programs are introduced and outlined briefly in this section and discussed in full detail in the subsequent parts of this document. The common names and program names of the system service programs and where they are described are:

Common Name	Program Name	Section
System librarian for system access technique (SAT) files	LIBS	2 and 3
System librarian for multiple indexed random access method (MIRAM) files	MLIB	2 and 3
Linkage editor	LNKEDT	4 thru 8
Initializing disk volumes	DSKPRP	9
Assign alternate track	DSKPRP	10
Tape prep	TPREP	11
Disk dump/restore	DMPRST	12
List software maintenance corrections	SMCLIST	13
Diskette copy	SU$CPY	14
8416/8418 disk copy	SU$C16	15
8419 disk copy	SU$C19	15
8430/8433 disk copy	SU$CSL	15

Common Name	Program Name	Section
System utility symbiont	SU	16
Create file definition program	CFDP	17
Hardware utilities (interactive DMPRST and SU$C19)	HU	12 and 15

1.2. System Librarians

There are two system librarians that can maintain and manipulate both your system and user libraries within OS/3. For all non-MIRAM library files, you use the SAT librarian (LIBS). For MIRAM libraries, you use the MIRAM librarian (MLIB).

The librarians are also used during OS/3 system generation to tailor the SYSRES program libraries. The librarians are capable of manipulating the library files at the user's request and in the specific manner directed. The functions performed by the librarians are controlled by a set of integrated subroutines, file tables, and overlay segments associated with the supported individual functions.

Your OS/3 system can support several independent system and user program libraries, and the librarians can be used to maintain each one. A program library consists of one or more library files. A single library may contain both user and system files, or it may be used exclusively for one or the other. Each file within the program library contains a directory partition, the library file directory, and two data partitions.

The program library files can be composed of any combination of the following:

- Program source modules (language processor code)
- Macro/jproc source modules (language processor code/job control)
- Object modules (language processor output/linkage editor input)
- Load modules (linkage editor output)
- Module groups

The library files may be composed of system or user code used for either program generation or execution. The code may be in any of the listed formats and may, from time to time, change in form, content, or relative position within a given file. The mixing and grouping of module types is a user option, and module groups can contain modules of the same or different types. Figure 1-1 depicts the structure of a SAT program library, showing various component configurations.

The librarians can perform, on all or specific portions of library files, such tasks as copying, merging, listing, or punching on cards the contents of specified files. The librarians can also add or delete from a file or files. In fact, the OS/3 librarians can perform all the tasks you may be expected to require for program file management. These tasks are initialized and directed through a set of control statements introduced to the librarians through the control stream. The librarians and the function associated with each task are fully explained in Part 2 of this guide.

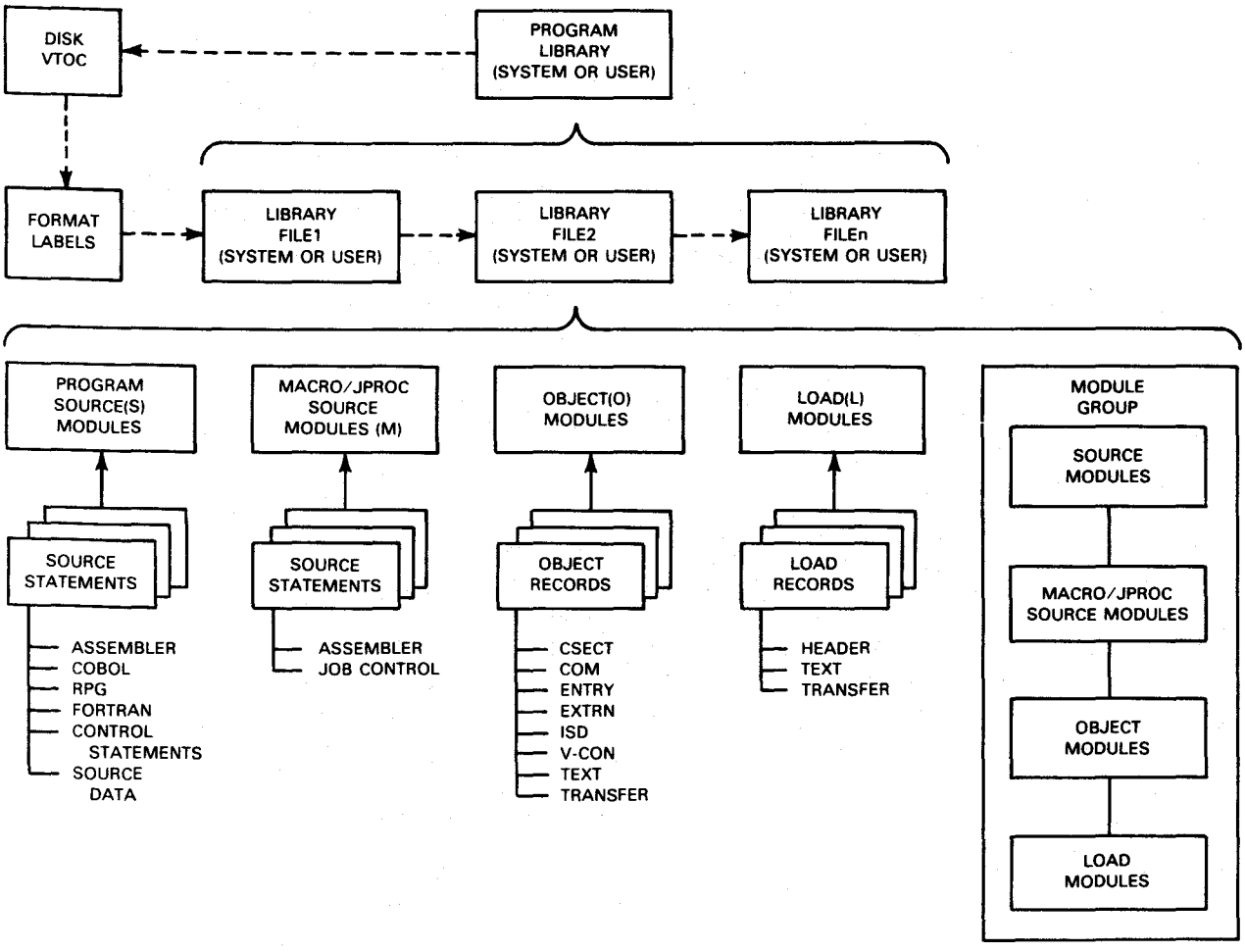


Figure 1-1. SAT Program Library Structure

1.3. Linkage Editor

The OS/3 linkage editor (LNKEDT) converts and combines object modules and object module elements (control sections and common sections) produced by the OS/3 language processors into modules that can be loaded into a system by the supervisor for execution. The modules produced by the linkage editor are called load modules. Only programs in load module form can be executed in an OS/3 environment, and the only way to convert object modules into a load module is by using the linkage editor.

The linkage editor produces three types of load modules:

- Single-phase (reentrant or nonreentrant)
- Multiphase (nonreentrant)
- Multiregion (nonreentrant)

A single-phase load module consists of a single program segment loaded into main storage each time the program is to be executed. Unless otherwise directed, the linkage editor will always produce a single-phase load module. Multiphase and multiregion load modules are composed of more than one program segment, each segment being a program phase loaded into main storage and executed individually, as required by the logic of the program. The linkage editor will create a multiphase or multiregion load module from one or more object modules only if directed to do so by the user through the linkage editor control statements. Savings in main storage space and increased system performance can be realized through proper application of multiphasing and multiregioning.

The capabilities of the linkage editor provide the system user with the following advantages:

- If a program logic error is discovered in a particular object module or control section of a program, only the incorrect element needs to be recompiled or reassembled. Afterward, the entire program can be relinked without extensive reassembling or recompiling.
- Subroutines or elements required in more than one program phase need to be preserved only once as relocatable object code because a single module can be individually included in any number of load modules by the linkage editor.
- A single load module may actually consist of object elements produced by several different language processors because all processors generate compatible output object code acceptable to the linkage editor.
- Reentrant modules can be shared by other load modules, resulting in the overall reduction of main storage requirements.

The capabilities of the linkage editor are described in detail in Part 3.

1.4. System Utilities

The system utilities are available to do the following:

- Test and prepare all tape, diskette, and disk volumes for use by OS/3.
- Apply patch corrections to certain areas of OS/3 normally inaccessible to the user.

1.4.1. Disk Utilities

The disk utilities perform the following functions:

- Initialize 8417, 8470, and 8494 nonremovable disks; 8417 fixed-head nonremovable disk; and 8416, 8417, 8418, 8419, 8430, and 8433 removable disks
- Perform surface analysis on disk volumes
- Assign alternate tracks on disk volumes
- Dump, restore, or copy disk, tape, or diskette volumes
- Dump or restore disk, tape, or diskette files
- Copy disk files
- Place initial microprogram load (IMPL) modules on disk
- Assign new volume serial numbers to disks

1.4.2. Diskette Utilities

The diskette utilities perform the following functions:

- Initialize diskettes
- Perform surface analysis on diskette volumes
- Place IMPL modules on diskette volumes
- Indicate whether diskette is to be used as an initial program load (IPL) volume
- Format diskettes in data set label mode (DSL) or format label mode (FLB)

1.4.3. Tape Utilities

The tape utilities initialize tape volumes for use in the system. Up to 36 tapes can be initialized at one time.

1.4.4. Hardware Utilities (HU)

The hardware utilities consist of the dump/restore and the disk copy routines executed interactively.

1.5. List Software Maintenance Corrections (SMCLIST)

You can use the SMCLIST canned job control stream to print a listing of all the software maintenance corrections (SMCs) contained in the SMCLOG file.

1.6. System Utility Copy Routines

System utility copy routines perform the following functions:

- Copy and verify System 80 diskettes (SU$CPY)
- Copy and verify these disks: 8416/8418 (SU$C16), 8419 (SU$C19), and 8430/8433 (SU$CSL).

1.7. System Utility Symbiont (SU)

The system utility symbiont is a multipurpose utility enabling the operator to perform different utility operations from the system console. For example, printing a diskette and printing a tape.

1.8. Program Error Checking (UPSI Byte)

The OS/3 system provides every job with a 12-byte communications region residing in the job preamble. The last byte of this region is the user program switch indicator (UPSI). The UPSI byte is used to pass information from one job step to the next job step and to indicate the presence of program errors. The librarian, the linkage editor, the utilities, the dump routines, and other executable system components set the UPSI byte if errors are detected. You can test the UPSI byte during program execution to determine the nature and severity of any errors.

The basic bit usage of the UPSI byte is

- Bit 0

  A 1 in the first bit (X'80') indicates a catastrophic error. Subsequent job steps probably will not function and the job will terminate.

- Bit 1

  A 1 in this bit (X'40') indicates a serious error. A serious error could affect subsequent job steps or result in incomplete or erroneous processing results.

- Bit 2

  A 1 in this bit (X'20') indicates a statement format or syntax error. The affected statement will not function, and this may or may not have an effect on subsequent job steps.

The UPSI byte can be useful in contingency error processing. For example, the byte can be examined and, if certain conditions prevail, can cause a branch to error handling routines. The SKIP job control statement is used to perform the test. The following examples show how you can use the SKIP job control statement.

Example 1

```
      1        10    16
1.  // JOB DSKPRP
2.  // DVC 20 // LFD PRNTR
3.  // DVC 51 // VOL DSP028  // LFD DISKIN
4.  // EXEC DSKPRP
5.  /$
6.     SERNR=DSP028,PARTL=V
7.  /*
8.  // SKIP ENDS,1
9.     .                    } Additional
10.    .                    } job steps
11.    .                    } as required
12. //ENDS NOP
13. /&
14. // FIN
```

In Example 1, you check the UPSI byte to see if a fatal error (X'80') has occurred. If the leftmost bit (bit 0) of the UPSI byte contains a binary 1 (line 8), then all the other job steps are bypassed and control is transferred to the NOP job control statement with the label ENDS (line 12). The NOP job control statement provides you with an address for the SKIP with no function being performed. The /& job control statement terminates our job while the //FIN terminates the card reader operation.

Example 2

```
        1        10    16
1.  // JOB DSKPRP
2.  // DVC 20 // LFD PRNTR
3.  // DVC 51 // VOL DSP028 // LFD DISKIN
4.  // EXEC DSKPRP
5.  /$
6.     SERNR=DSP028,PARTL=V
7.  /*
8.  // SKIP WARN,01
9.  // SKIP FATAL,1
10. // SKIP EXIT
11. //WARN OPR 'WARNING-A NON-FATAL ERROR HAS OCCURRED'
12. // SKIP EXIT
13. //FATAL OPR 'FATAL ERROR-JOB TERMINATED-CORRECT AND RERUN'
14. // SKIP ENDOFJOB
15. //EXIT NOP
16.   .                    } Additional
17.   .                    } job steps
18.   .                    } as required
19. //ENDOFJOB NOP
20. /&
21. // FIN
```

In Example 2, you check for both the fatal (X'80') and warning errors (X'40') and the display of appropriate messages on the system console. If a warning error has occurred, bit 1 of the UPSI byte is a binary 1 (line 8), then you skip to the label WARN on the OPR job control statement and print the warning message (line 11). After processing the OPR statement, the SKIP job control statement (line 12) is the next job control statement processed. Here, you skip down to the label EXIT on the NOP job control statement (line 15). As mentioned earlier, the NOP acts as an ending point for the SKIP control statement. The remaining job steps follow the NOP statement and are processed accordingly. Following the last job step, the NOP statement on line 19 is processed with no action being performed. Your job then terminates normally through the /& and //FIN job control statements.

If a fatal error occurs, bit 0 of the UPSI byte is a binary 1 (line 9) and you skip down to the label FATAL on the OPR statement (line 13) and print the specified message. The SKIP job control statement (line 14) skips down to the label END OF JOB on the NOP statement, thus bypassing your remaining job steps, and terminates your job.

If no errors occurred, neither SKIP (lines 8 and 9) will be taken, and the SKIP job control statement (line 10) skips over the OPR statements to the remaining job steps.

The UPSI byte setting and the error count appear on the printout or map listing for the particular job. The UPSI byte value can also be retrieved by issuing the GETCOM supervisor macroinstruction in your BAL program. For more information on the GETCOM macro, refer to the *Supervisor Macroinstructions Programming Reference Manual* (UP-8832). For more information on the SKIP job control statement, refer to the *Job Control Language Programming Guide* (UP-9986).

1.9. Using Workstations

OS/3 enables you to run system service programs interactively. This means two things:

1. You can build a control stream to execute the system service programs at a workstation, as opposed to punching it on cards or writing it to a diskette.

2. You can initiate the running of the control stream from the workstation, as opposed to asking the system operator to run your job for you.

The easiest way to build a control stream from a workstation is by using the job control dialog. The job control dialog is an interactive facility of OS/3 that allows you to describe your job's requirements to it in English, in response to a series of questions, and then produces as its output the job control stream needed by OS/3 to run your job. The control stream produced by the job control dialog is virtually identical to the control stream that you would have to produce if you were running your job in a batch environment. Only now, you do not have to be concerned with the intricacies of the job control language. The job control dialog eliminates this requirement on your part.

After you have answered all the questions presented to you by the job control dialog, the job control dialog builds a control stream and stores it in a permanent library file for you. From here, you can initiate its running by simply keying in the appropriate system RUN command, or, if you'd rather, you can change the contents of the control stream using another interactive facility of OS/3 called the general editor.

The procedures for activating the job control dialog, initializing the running of a job, and activating the general editor are described in detail in the *Interactive Services Operating Guide* (UP-9972).

More detailed descriptions of the job control dialog and the file editor are presented in the *Job Control Language Programming Guide* (UP-9986), and the *General Editor (EDT) Operating Guide* (UP-9976).

1.10. Statement Conventions

The conventions used to illustrate the control statements and system console message displays presented in this manual are:

- Positional parameters must be written in the order specified in the operand field and must be separated by commas. When a positional parameter is omitted, the comma must be retained to indicate the omission, except for the case of omitted trailing parameters.

  Examples

  Assume that INCLUDE is a linkage editor control statement with three optional positional parameters: A, B, and C.

  ```
  INCLUDE A
  INCLUDE A,B
  INCLUDE A,B,C
  INCLUDE A,,C
  ```

- A keyword parameter consists of a word or a code immediately followed by an equals sign, which is, in turn, followed by a specification. Keyword parameters can be written in any order in the operand field. Commas are required only to separate parameters.

  Examples

  Assume that LINKOP is a linkage editor control statement with three optional keyword parameters: ALIB, RLIB, and OUT.

  ```
  LINKOP ALIB=OBJFIL,RLIB=$Y$OBJ,OUT=$Y$LOD
  LINKOP ALIB=OBJFIL,RLIB=$Y$OBJ
  LINKOP RLIB=$Y$OBJ,ALIB=OBJFIL
  LINKOP OUT=$Y$LOD
  ```

- A positional or keyword parameter may contain a sublist of parameters, called subparameters, separated by commas and enclosed in parentheses. The parentheses must be coded as part of the list. The subparameters within the parentheses may be positional, in which case the comma must be retained if a parameter is omitted, except for the case of trailing parameters, or they may be nonpositional. The description of the subparameters indicates whether they are positional or nonpositional.

  Examples

  ```
  FIELDS=([ADDR][,A2TD][,FREQ])
  REDO=(MERGE,label,reel,to)
  ```

- Uppercase letters, commas, equals signs, apostrophes, and parentheses must be coded and displayed exactly as shown. The exceptions are acronyms, which are part of generic terms representing information to be supplied by the programmer.

  Examples

  ```
  CMcc NUMBCHAR=n
  X'aa' (NOV)
  ALIB=
  ```

- Lowercase letters and words are generic terms representing information that must be supplied by the user. Such lowercase terms may contain hyphens and acronyms (for readability).

  Examples

  ```
  lfn
  name
  group-name
  comments
  s1,...,sn
  ```

- Information contained within braces represents mandatory entries of which one must be chosen.

  Examples

  ```
  { filename }
  { (N)      }
  { $Y$RUN   }
  ```

- Information contained within brackets represents optional entries that (depending upon program requirements) are included or omitted. Braces within brackets signify that one of the specified entries must be chosen if that parameter is to be included.

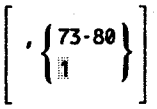
  Examples

  ```
  [sequence-no]
  [ALIB=filename]
  [ { fn     } ]
  [ { $Y$RUN } ]
  ```

- An optional parameter with a list of optional entries may have a default specification that is supplied by the operating system when the parameter is not specified by the user. Although the default may be specified by the user with no adverse effect, it is considered inefficient to do so. For easy reference, when a default specification occurs in the format delineation, it is printed on a shaded background. If, by parameter omission, the operating system performs some complex processing other than parameter insertion, it is explained under an "If omitted" statement in the parameter description.

  Examples

- An ellipsis (...) indicates the presence of a variable number of entries.

  Example

```
param-1,...,param-n
```

- Commas are required when positional parameters are omitted, except after the last parameter specified.

  Example

```
positional-parameter-1,positional-parameter-2,,positional-parameter-4
```

Note: *There are three standard character sets used with Unisys printers: two are 48-character print sets, and the third is a 63-character print set. Thus, not all characters are printable on all machines, and print code conversions are necessary to represent nonprintable characters when a 48-character print set is being used. The programming examples presented in this guide were produced using the standard 48-character business print set and, therefore, make use of some of these conversion print characters. For example, an equals sign (=) is represented by a percent symbol (%), a left parenthesis by a number symbol (#), and a right parenthesis by an at symbol (@).*

Section 2
Overview of the Librarians

2.1. Introduction

The Unisys Operating System/3 (OS/3) provides two librarian utility programs to maintain the SAT and MIRAM libraries. The SAT librarian utility called LIBS is used to maintain SAT files in system and user program libraries. The MIRAM librarian utility called MLIB is used to maintain screen format modules, help screen modules, saved run library modules, and menu modules in MIRAM files.

Although LIBS and MLIB are primarily disk utilities, they can be used to maintain files on tape, diskette, or punched cards. They can also transfer files from one storage medium to another.

Library maintenance is performed by running a job that executes the appropriate librarian utility program. The input that the user must supply with this program is a series of librarian control statements. Each statement specifies a particular librarian operation, such as a module copy or deletion. So the statements used in each librarian job vary, depending on the kind of maintenance needed. When the job is run, the statements are executed in the order that they are supplied in the job. As output, the librarian produces updated files, punched cards, listings, or a combination of these, as dictated by the control statements.

Figure 2-1 illustrates that the librarian can process modules to or from any kind of storage device. The librarian can also print a librarian map that lists the control statements executed in the job and any listings requested by the control statements.

This section describes the structure of the SAT and MIRAM libraries. It also gives an overview of the capabilities of the two librarians and the job control required to run them. Section 3 describes the function and syntax of all the librarian control statements available for each librarian. Sample librarian jobs are also included in Section 3.

Canned librarian job control streams are available in the system to process modules and files on the system release volume. These are described in 3.5 and Appendix A.

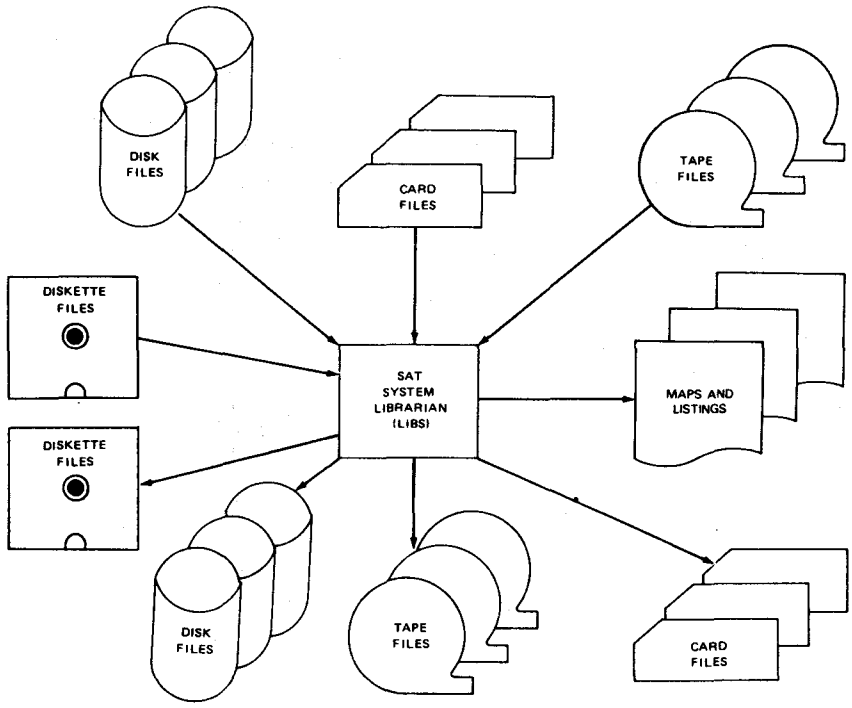


Figure 2-1. Librarian Input/Output Capabilities

2.2. SAT Librarian Capabilities

The SAT librarian (LIBS) is a utility program used to maintain both system and user SAT libraries. A SAT library is a collection of program files. Each file can contain any assortment of program modules. A program module is made up of either source, macro, jproc, object, or load code. The most commonly used system libraries are YLOD, YOBJ, YSRC, YMAC, YJCS, and the job run library YRUN, which contains any temporary modules created by OS/3 during job execution. User libraries contain any program modules created and stored by the user.

The SAT librarian can perform the following kinds of operations:

- Copy modules from one file to another
- Copy modules from one storage medium to another
- Update existing modules in a file
- Delete modules from a file
- Correct module records
- Compare modules in a file

- Print listings of modules on the printer
- Rename modules in a library file
- Compress unused space in a file
- Sequence source modules
- Convert standard load modules to blocked load modules

2.3. SAT Library Structure

The structure of each library depends on the characteristics of the storage medium, whether it be disk, tape, diskette, or cards. These are described in 2.3.1 through 2.3.4.

2.3.1. Disk Libraries

System and User Disk Files

System library files reside permanently on the SYSRES volume. These files are created during system generation and contain all of the programs and data needed to run the system. Job control also creates a job run file called YRUN for each active job in the system. The YRUN file is used to process data during job execution and is automatically scratched when the job terminates. Therefore, any modules created in YRUN that will be needed later should be copied to a permanent user file before the job terminates. Any number of additional disk files can be created by the user to store program modules of his choice.

Structural Levels (Volumes, Files, Modules, and Records)

Each disk pack is called a volume. A volume can contain any number of files, each with a unique file name. Each file contains modules. A module can contain either source, object, or load code for a program. A module can also contain a set of assembler macroinstructions or a job control procedure. Modules are constructed in segments called records. These records are variable in length and are grouped within fixed-length blocks. Figure 2-2 illustrates these structures. Appendix C contains additional details about internal structure of various kinds of records and blocks.

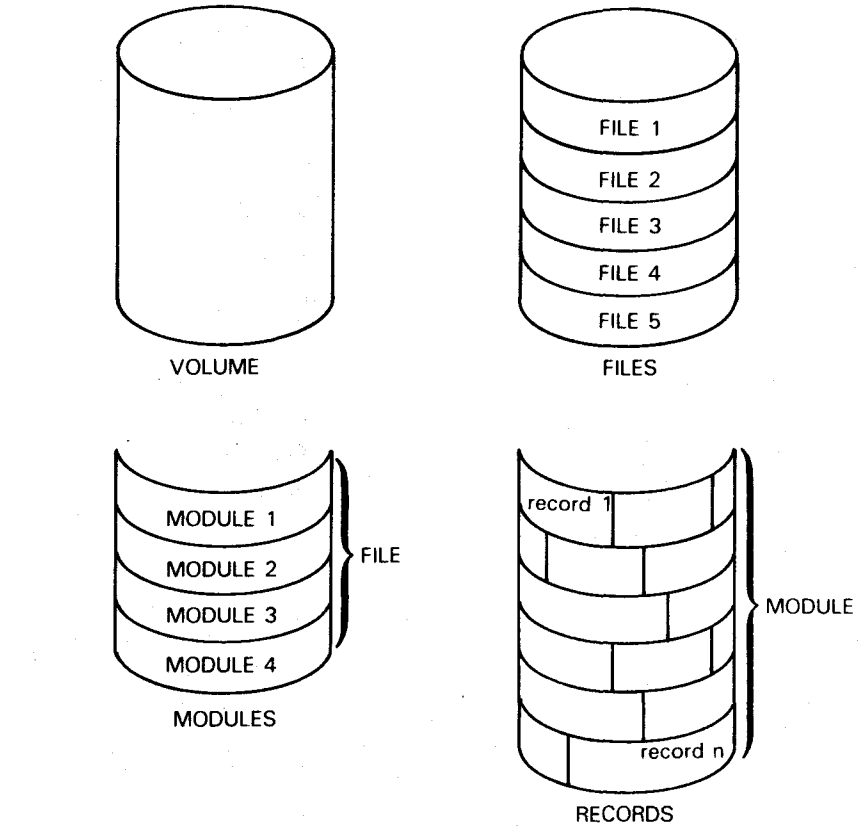


Figure 2-2. SAT Library Structures

Disk File Space Allocation

Each SAT file on disk has three partitions, or sections, as illustrated in Figure 2-3. One partition is used for a file directory. The other two partitions are data areas used to store program modules. When the librarian initializes a SAT file, it allocates two percent for the directory partition and 48 percent for the first data partition. It does not initially allocate any space for the second data partition. (The second partition is only used later if and when blocked load modules are stored in the file.)

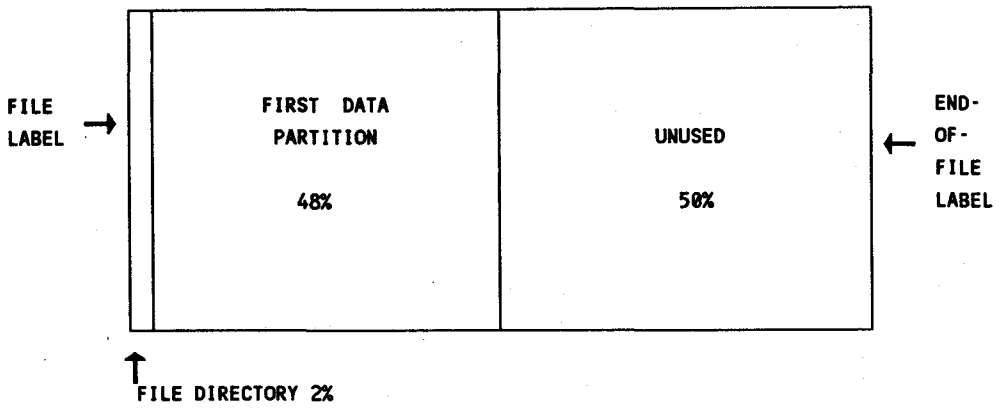


Figure 2-3. Initial SAT File Allocation

Note that 50 percent of the file is initially unassigned. When either the directory or the first data partition is full and requires more space, the librarian will use some of this free space to extend that partition. The size of this extension will be based on the file extension increment specified when the file was initially allocated. When all the allocated file space has been used, the system will automatically extend the file by allocating additional free space in the same increments. This technique conserves space by allocating it only when needed.

Data Records

The data partition of a SAT file on disk contains a series of data records containing source, macro/jproc, object, or load code for all the modules in the file. The data partition also contains delimiter records which identify such things as the beginning of a module or a load module phase. Details on the content and format of each type of record are provided in Appendix B.

Directory Records

For each disk file, the librarian maintains a directory it uses to locate delimiter records within the data area. Each directory record identifies the name, type, and location of its corresponding delimiter record within the data area (see Figure 2-4). This access method requires that each module have a unique name and type combination. The names in other directory entries need not be unique.

The following kinds of delimiter records are indexed in the directory:

- Module header records
- Load module phase header records

  Note: There is a separate directory record for each phase name and each alias phase name.

- Macro and jproc procedure module header records

  Note: There is a separate directory record for each macro or jproc procedure name.

- ENTRY records for object code
- Named CSECT records for object code
- Beginning-of-group (BOG) records
- End-of-group (EOG) records
- End-of-file (EOF) record

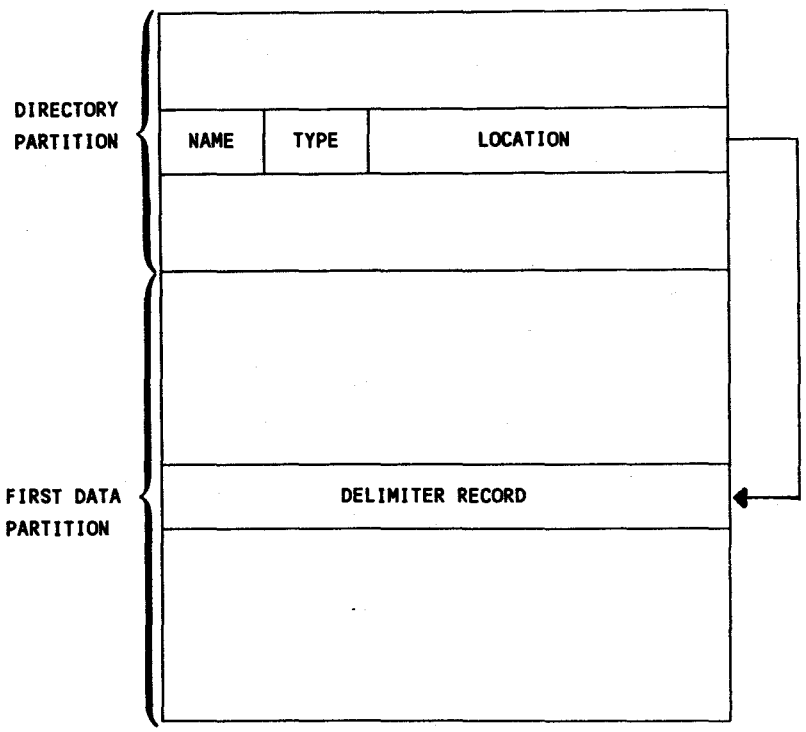


Figure 2-4. Directory Operation

2.3.2. Tape Libraries

Each reel of tape of a tape library is a volume. Tape libraries have the same block and record format as disk libraries; however, tape libraries do not have directories. Modules on tape are located by reading the tape sequentially.

2.3.3. Diskette Libraries

For diskette libraries, each diskette is a volume. A diskette must be prepped as either a data set label diskette or a format label diskette (see 2.6.8 for instructions.)

A file on a format label diskette is treated and structured like a disk file with a data area and a directory. Program modules stored on format label diskettes can be executed directly from diskette.

A file on a data set label diskette is a sequential file like a tape file. It has a record size of 128 bytes. These files can be used to store backup copies of program modules; but the programs must be copied to format label diskettes or disks to be executed. Data set label diskettes can be used as a substitute for cards.

2.3.4. Card Libraries

When a source module is punched on cards, each card is considered a record. Because card records have a fixed length of 80 characters, when a source module from another storage medium is punched on cards, any records having more that 80 characters will require more than one card. Each card should contain a sequence number so that the librarian can automatically check the order of the cards as it reads them. Modules that are not sequenced when they are originally punched can be sequenced later using the SEQ librarian control statement.

When an object or load module is punched on cards, it is divided into 80-byte records containing a 5-byte sequence number and 75 bytes of data. When the librarian copies the records back to a disk or tape file, it reassembles the records as they originally existed in the original file.

Data set label diskettes can be used as a substitute for cards. On these diskettes, the record size is 128 bytes.

2.4. SAT File Allocation and Management

The SAT librarian cannot allocate files. It can only manipulate existing files on disk, tape, or diskette. System files are automatically allocated by the system. User files must be allocated explicitly. A user file on disk or format label diskette must be allocated by either an // EXT job control statement or the interactive services ALLOCATE command. The file type must be ST for SAT. A tape file or data set label diskette is essentially allocated when it is prepped.

Once a file has been allocated, modules can be stored in it. The entire contents of the file can then be copied to any other storage medium, compared with another file, deleted, or compressed using the librarian. The librarian can also be used to print a file directory or listing.

2.5. SAT Module Creation and Management

2.5.1. Module Types

A module is a set of one type of code for one program. There are four types of program code: program source, macro/jproc source, object, and load. The module type refers to the type of code a module contains. A single file can contain modules of assorted types arranged in any order.

2.5.2. General Module Operations

Many librarian operations can be applied to one or more modules at a time, as specified through options and parameters in the librarian control statements. For example, an operation can apply to one module, all modules with a certain name prefix, all modules of a certain type, or a series of consecutive modules within a file. Module groups containing any assortment of modules can also be processed collectively in group operations.

All types of modules can be copied to another file or be renamed. However, librarian operations that change the content of the module itself may be restricted to certain types of modules. The following sections explain the restrictions on each type of module and on module groups.

2.5.3. Program Source Modules

A program source module is a set of program source statements written in a programming language such as COBOL or RPG II. A job control stream is also considered a source module.

Program source modules can be created interactively at a workstation using the general editor (EDT) or a specialized subeditor of EDT, such as the RPG II editor or the COBOL editor. These modules can then be copied to tape, disk, or diskette through the use of the EDT @WRITE command. They can also be punched on cards through the use of the EDT @PUNCH command. Source modules can also be created on punched cards by using coding forms and a keypunch.

Once a program source module has been created, it can be copied, deleted, listed, resequenced, compared, punched, or renamed using the librarian. The librarian can also be used to change, delete, add, or resequence records within a source module.

2.5.4. Macro and Jproc Source Modules

A macro source module contains a series of assembler macroinstructions and directives used as input to the assembler. A jproc (or proc) source module contains a series of job control statements and directives that make up a job control procedure. All of the librarian operations that can be used on program source modules can also be used on macro and jproc source modules.

The librarian identifies both macro and jproc source modules as macro source modules in the file directory. Therefore, a macro source module and a jproc source module with the same name may not exist in the same disk file.

Each macro and jproc source module may be referenced by more than one name. In this case, the directory contains a separate record for each additional name. There can also be more than one directory record specifying the same name. However, each directory record will reference the same location for the module in the data area.

In librarian control statements, the module type M is used to refer to macro, proc, or jproc source modules.

2.5.5. Object Modules

When a source program is compiled by its appropriate language compiler, the output of the compiler is an object module. This object module can then serve as input to the linkage editor.

The job control statements used to compile the source program specify the output file where the object module is to be stored. The output file can be the job run file YRUN, the YOBJ system file for object modules, or any user file. The YRUN file is scratched at the the end of job execution, so the object module is only saved permanently if another output file is specified.

Object modules can be copied, deleted, compared, punched, listed, renamed, and patched using the librarian. COM, ENTRY, CSECT, V-CON and EXTRN records within object modules can also be renamed. The librarian can also be used to insert or delete linkage editor control statements in an object module.

Whenever object modules are serviced by the librarian, they are checked for proper content and record sequence. Discrepancies trigger diagnostic processing.

All object modules produced by the various language compilers are assumed to be nonsharable (nonreentrant) modules. However, sharable (reentrant) modules may be flagged by the librarian to enable them to produce sharable (reentrant) load modules when they are link-edited.

2.5.6. Load Modules

When an object module is processed by the linkage editor, the output is a load module, which is a program in executable form. Part 3 of this guide explains how to use the linkage editor. A LINK jproc call statement that executes the linkage editor is described in the *Job Control Language Programming Guide* (UP-9986).

The // PARAM or // LINKOP control statements used in the linkage editor job step specify the output file where the load module is to be stored. The output file can be the job run file YRUN, the YLOD system file for load modules, or any user file. The YRUN file is scratched at the the end of job execution, so the load module is only saved permanently if another output file is specified.

Load modules can be copied, deleted, compared, punched, listed, and renamed using the librarian. The librarian can also be used to patch load modules. Patches are inserted at the end of a specified phase, which is a portion of a load module that is loaded into main storage at one time.

Each phase in a load module is given a name at link-edit time. This name can be supplied automatically by the linkage editor or explicitly by the programmer. The programmer may also assign each phase an additional alias-phasename to be used to reference the phase in program subroutines. Alias-phasenames can be changed using the librarian.

Most load modules produced by the linkage editor can be converted to blocked load modules by the librarian. Blocked load modules can usually be loaded for execution faster than their unblocked versions. Exceptions to this are explained in the description of the BLK librarian control statement (see 3.3.13).

2.5.7. Module Groups

The librarian can process modules in groups. A group is a consecutive series of modules within a file. It can contain any number of modules of assorted names and types. When a module group is created through the librarian, it is given a group name that is used to reference the entire group of modules so that they can be processed collectively. For example, an entire group of modules can be copied or deleted with one librarian control statement that references the group name. The librarian can also be used to change the group name. More information on creating and processing module groups is given in 3.3.18.

2.5.8. Module and Group Header Records

The first record in each module is a module header record that identifies the module type, name, date and time of creation, and any comments the user includes. When a librarian job is run, this header information is printed on the librarian map to verify which modules were processed by each operation. Similarly, each module group within a file begins with a module group header record that identifes the name of the group and an optional comment. In disk files, all module header records and group header records are indexed in the file directory to help the librarian locate them when needed.

2.5.9. Naming Conventions

Module Names

Each module within a library file has a name assigned by the user. This name can contain up to eight characters, including letters, digits, &, $, ?, or #. If the name assigned to a source or object module contains less than eight characters, it is left-justified and padded with blanks to the right. A load module is padded with zeros.

Modules are named when they are created, as follows:

- Assembly or compile time for object modules
- Link-edit time for load modules

- Allocation time for program source modules
- Job run time for macro and jproc source modules

A module can be renamed by using the REN librarian control statement.

See 2.5.4 for more information about naming macro and jproc source modules.

Module Group Names

When a module group is created, it is given a name containing up to eight characters, including letters, digits, &, $, ?, or #. This group name is independent of the individual module names within the group. It is used only to refer to all modules in the group at once, never part of the group.

A group name need not be unique. The use of options and parameters in the librarian control statements can specify all groups with a certain group name, only the next group with a certain group name, or all groups with a certain name prefix. Every module in a disk file must still have a unique name and type combination, even if they are in different groups within the file.

A group name may be changed by using the REN librarian control statement.

Name Prefixes

Librarian control statements that allow the C option can process all modules or groups with the same name prefix. Thus, related modules or groups can be given related names and then be processed together. Any consecutive series of characters from the beginning of a module or group name can be treated as a prefix. All of the following module names have the name prefix COB:

COBACCT

COBMAST

COBSRC43

2.6. Using the SAT Librarian

2.6.1. Basic Job Control for the SAT Librarian

The following is a sample job control stream used to run the SAT librarian. For more information on job control, see the *Job Control Language Programming Guide* (UP-9986).

```
1.  // JOB SAMPLE
2.  // DVC 20  // LFD PRNTR
3.  // DVC 40  // LFD PUNCH
4.  // DVC 50  // VOL DSK001
5.  // LBL DISKFILE  // LFD DISK1
6.  // DVC 100 // VOL TAP001
7.  // LBL TAPEFILE  // LFD TAPE1
8.  // EXEC LIBS
9.  /$
              .
              .
              .
10. /*
11. /&
```

Device assignments depend on devices and files required for job

Librarian control statements

1. The // JOB statement names the job SAMPLE. This statement can also specify additional main storage to improve the performance of the librarian (see 2.6.5).

2. This is the device assignment set for a printer that is needed to print the librarian map. If a printer is not assigned to the job, the // PARAM PRINT=OFF statement must be used to suppress printing of the librarian map (see 2.6.3).

3. through 7.
   These are device assignment sets for the devices and files required for the job. These will vary with each job. In this case, line 3 is a device assignment set for a card punch. Lines 4 and 5 are a device assignment set for one disk file. Lines 6 and 7 are a device assignment set for one tape file.

   For each disk, tape, and diskette file, the name following the // LBL is the name of the file as recorded on the storage medium. The logical file descriptor (LFD) is the name given to the file within the program. In the librarian job step, FIL librarian control statements equate each LFD with a logical file name that is then used in subsequent librarian control statements to reference that file.

   For more information on device assignment sets, see the *Job Control Language Programming Guide* (UP-9986).

8. The // EXEC LIBS statement executes the SAT librarian called LIBS.

9. and 10.
   The /$ indicates the beginning of the librarian control statements that describe the librarian functions to be performed. The /* indicates the end of the librarian control statements.

11. The /& indicates the end of the job.

Notes:

1. *All statements from the / / EXEC LIBS statement to the /* statement, inclusive, comprise a separate job step that executes the librarian. Therefore, other job steps could be inserted in this job between the device assignment statements and the / / EXEC LIBS statement or between the /* and the /& statements.*

2. *When librarian job control stream is created on punched cards, a / / FIN job control statement is required after the /& end-of-job statement to end the card reader operation.*

3. *A sample session illustrating the use of the general editor (EDT) to create and run a librarian job interactively is given in 3.6.1.*

4. *A single librarian job can process any number of files. However, the maximum number of files that can be open at one time is six. See 3.3.1 for more information.*

5. *The job run file YRUN can be processed in the same way as any other file and be declared in the FIL librarian control statement. Even if it is not declared in the FIL statement, the YRUN file can still be used by default in some librarian control statements. No device assignment set is required for YRUN.*

6. *A program should not be executed while it is being restored, updated, or punched on cards.*

2.6.2. Error Handling

Error Messages

Normally, when the librarian encounters an error in processing a control statement, it prints an error message on the librarian map and terminates the execution of that statement. Each error message contains an error number and a brief error description. More detailed error descriptions and corrective actions are contained in the current version of the *System Messages Reference Manual* (UP-8076).

Stopping or Canceling the Job (// PARAM ERROR)

Normally, if an error is not critical, the librarian will continue to process subsequent librarian control statements in the job step. There are two // PARAM statements that can be used to stop execution of the librarian when an error occurs.

// PARAM ERROR=STOP ends the librarian job step when an error is encountered in the librarian control statements. Subsequent librarian control statements are not executed. However, the UPSI bytes are set and the librarian job step terminates normally. Also, any subsequent job steps are executed.

// PARAM ERROR=CANCEL causes immediate abnormal termination of the job in the event of an error, with abnormal termination of the librarian job step. No subsequent job steps are executed. This can be used by Unisys systems analysts to document librarian processing problems.

Only one of these statements should be used in a job. If used, it must be placed between the // EXEC LIBS and the /$ job control statements.

2.6.3. Librarian Map

Description

Each time the librarian is executed, it can produce a librarian map containing the following information:

- All librarian control statements in the order that they were processed. Each control statement is identified with the label COMMAND .
- Header information for all modules processed by each statement (unless the N option is used) including module flags set by the librarian for internal use only. (They are: R, reentrant module; B, base0 module; K, key0 module; C, module has been corrected.)
- Any error messages caused by each librarian control statement.
- Any module listings requested by each librarian control statements.
- All records added, deleted, or changed during module correction operations.
- Any discrepancies found during module or file compare operations.

Sample librarian maps are provided with the SAT librarian job examples in 3.6.

Suppressing the Librarian Map (// PARAM PRINT=OFF)

If a printer device is assigned to the librarian job, the librarian produces a librarian map automatically. When a librarian map is not needed, the following job control statement must be used to suppress the printing of the librarian map:

```
// PARAM PRINT=OFF
```

If used, this must be the first // PARAM statement in the librarian job step. It must be placed immediately after the // EXEC LIBS job control statement and precede the /$ job control statement. This statement suppresses the printer for the duration of the librarian job step only. After that, any printer declared in a device assignment set is still available for other job steps.

If this job control statement is used when a printer is not assigned to the job, a DM03 error is returned.

Printing Source Modules in Hex Format (// PARAM PRTOBJ=ON)

Whenever program source or macro/jproc source modules are printed on the librarian map by a COP statement or a D option, they are printed in EBCDIC exactly as they were coded. To print them in hex format instead, include the following statement after the // EXEC LIBS statement and before the /$ job control statement in the librarian job control stream:

```
// PARAM PRTOBJ=ON
```

This parameter remains in effect for the entire librarian job step.

2.6.4. Specifying the Date and Time of Module Creation (// PARAM UPDATE)

Each module header record contains the date and time that the module was created, corrected, or updated. When the librarian corrects or updates a module, it normally obtains this date from the system information block while the librarian job is being executed.

The // PARAM UPDATE statement can be used to set a different date and time to be in effect for the duration of the librarian job only. If used, it must be placed between the // EXEC LIBS and the /$ job control statements. The // PARAM UPDATE statement has the following format:

```
// PARAM UPDATE=yymmdd/hhmm
```

The value used for yymmdd/hhmm represents the date and time in the form year, month, day, slash, hour, minute.

2.6.5. Allocating Additional Main Storage

Execution time for a librarian job can be reduced by allocating additional main storage space for the job in the // JOB job control statement. This additional space is allocated in track-sized buffers. To determine the amount of main storage to allocate, compute the number of buffers needed using the following recommendations:

- Specify at least two buffers for disk access (four or more is optimal).
- Add one more buffer for each diskette or variable block tape used (one for each T*n* file declared in the FIL librarian control statement).
- Allocate additional main storage for ESC processing (see Table 2-1).

The second positional parameter in the // JOB statement must specify the amount of additional main storage in decimal or hex bytes, as shown in Table 2-1. For example, each of the following // JOB statements allocate enough main storage for the librarian base plus one buffer:

```
// JOB SAMPLE,,B620
// JOB SAMPLE,,X'B620'
// JOB SAMPLE,,D'46624'
```

Table 2-1. Additional Main Storage Requirements for Using the SAT Librarian

Buffer Requirements	Decimal Bytes	Hex Bytes
Librarian base	34,304	8600
ESC processing	29,696	7400
1 buffer	46,624	B620
2 buffers	62,528	F440
3 buffers	78,432	13260
4 buffers	94,336	17080
5 buffers	110,240	1AEA0
6 buffers	126,144	1ECC0

2.6.6. Processing Disk Files

The following notes apply to the librarian processing of disk files:

- A disk volume may contain more than one file.
- Each module must have a unique name and type combination.
- Modules are located using a file directory.
- Modules and files can be deleted.
- Modules are updated by deleting the old version and adding the new version to the end of the file.
- Files are extended automatically by the system, when necessary.
- Files can be compressed to obtain contiguous space at the end of the file.

2.6.7. Processing Tape Files

Librarian Restrictions for Tape Files

In general, any librarian operation that alters the original file or processes the file directory cannot be performed with a tape file. These operations, and their associated control statements, include:

- Blocking load modules (unless the blocked version is output to disk)
- Deleting modules and files (DEL)
- Compressing files (PAC)
- Listing the file directory (LST)
- Renaming files and modules (REN)
- Sequencing modules (SEQ)

Other points to remember when using tapes are the following:

- A tape file may contain more than one module with the same name and type. In this case, when a librarian control statement references a module by name and type, the librarian processes the next one it encounters with that name and type.
- Modules are located by reading the tape sequentially from the pointer position and rewinding the tape, if necessary.
- New modules are added at the end of a tape file.

- A module on tape cannot be updated; however, a new version of the module can be copied to the same tape and be given the same name and type as the original.
- A module cannot be deleted from a tape file; however, a tape file may be edited by selectively copying the modules to be saved from the original tape to a new tape.

Note: The librarian will only support multiblock tape files, or variable block tape files, if the system supports consolidated data management (also called CDI) or mixed mode. Mixed mode includes both consolidated data management and basic data management (also called DTF). If the system supports DTF only, multiblock tape files are not supported by the librarian.

Prepping Tapes

Each tape must be prepped before any data can be stored on it. This can be done by using either the PREP option in the // VOL job control statement or the tape prep routine (TPREP) described in Section 11.

Note: When the PREP option is used in the // VOL job control statement, the tape is prepped each time it is opened as an output file. Therefore, if the tape is used as both an input and an output file in a single job, every time the file is reopened as an output file, all data on the tape as a result of previous operations will be overwritten. This prevents the continuous accumulation of data on the tape during the course of a job step. To avoid this problem, the tape should be prepped in a separate job step or through TPREP.

Header and Trailer Labels

Tape libraries must have standard header and trailer label records. The name specified in the // LBL job control statement must agree with the file 2 label in the tape library. See the current version of the *Consolidated Data Management Programming Guide* (UP-9978).

Optional Block Length Selection

Tape libraries use the same block and record format as disk libraries. There is, however, an option of specifying a block length other than the default length of 256 bytes. To do this, include the following job control statement in the device assignment set for the tape file (between the // VOL and // LBL statements):

```
// DD BKSZ=n
```

This statement tells the librarian to produce an output block or expect an input block of the size specified. The following rules apply:

- The value *n* must be a multiple of 256.

- The maximum value allowed for *n* is 8192.
- For an input tape, *n* must be at least as large as the value used when it was created.

For a complete description of the BKSZ parameter, see the *Consolidated Data Management Programming Guide* (UP-9978).

When processing tapes with variable-length blocks, one additional buffer of main storage space should be allocated for each tape in the job. This space is needed for processing operations and for I/O buffers (see 2.6.5 for instructions).

Creating a Multifile Tape (// PARAM TAPEFILES=MULTI)

The following // PARAM statement allows the librarian to output more than one file to a single-tape volume:

```
// PARAM TAPEFILES=MULTI
```

If used, this statement must be placed between the // EXEC LIBS statement and the /$ job control statement. If this statement is not used, only one file can be be written to a tape. This statement is not required to read a file on a multifile tape.

A multifile tape is created by copying files from another storage medium to output files on the tape. The tape must already be prepped and it may or may not already contain some files. The files must be copied in sequence as they are to be arranged on the tape. These files cannot be extended later.

The librarian job, which copies the files to the tape, must contain the // PARAM statement. Also, the // LBL job control statement for each new tape file must include a file sequence number parameter (in positional parameter 4) to indicate the position of the output file on the tape. For example, if the tape already contains two files, the // LBL statement for the first new tape file must specify a 3 for the file sequence number. If the tape is a newly prepped tape that does not yet contain any files, the file sequence number for the first new tape file must be 1.

As the files are being output to the tape, only one tape file can be open at a time. Therefore, the librarian job step should use the same logical file name (T*n*) for every tape file, but redefine that logical file name each time a new file is to be processed. (See the FIL librarian control statement description for more information.)

Normally, after each file is written to the tape and then closed, the librarian would rewind the tape to the load point. Then it would reopen the tape and advance to the end of the tape again to write the next file. However, a // DD job control statement with a rewind parameter can be used in the device assignment set for each tape to eliminate unnecessary rewinding at each open and close operation.

- The statement // DD OPRW=NORWD specifies no rewind at file open.
- The statement // DD CLRW=NORWD specifies no rewind at file close.

Specifically, the device assignment sets for the tape files should specify

- CLRW=NORWD for the first tape file
- OPRW=NORWD for the last tape file
- Both OPRW=NORWD and CLRW=NORWD for all other files

The following sample job copies five files from disk to the same new tape volume:

```
// JOB MULTFILE
// DVC 20 // LFD PRNTR
// DVC 50  // VOL D00410  // LBL DISKFIL1  // LFD DISK1
// DVC 50  // VOL D00410  // LBL DISKFIL2  // LFD DISK2
// DVC 50  // VOL D00410  // LBL DISKFIL3  // LFD DISK3
// DVC 50  // VOL D00410  // LBL DISKFIL4  // LFD DISK4
// DVC 50  // VOL D00410  // LBL DISKFIL5  // LFD DISK5
// DVC 90  // VOL S01841  // DD CLRW=NORWD
// LBL TFIL1,,,,1  // LFD TAPE1
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL2,,,,2  // LFD TAPE2
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL3,,,,3  // LFD TAPE3
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL4,,,,4  // LFD TAPE4
// DVC 90  // VOL S01841  // DD OPRW=NORWD
// LBL TFIL5,,,,5  // LFD TAPE5
// EXEC LIBS
// PARAM TAPEFILES=MULTI
/$
        FIL   D0=DISK1,T0=TAPE1
        COP   D0,,,T0
        FIL   D0=DISK2,T0=TAPE2
        COP   D0,,,T0
        FIL   D0=DISK3,T0=TAPE3
        COP   D0,,,T0
        FIL   D0=DISK4,T0=TAPE4
        COP   D0,,,T0
        FIL   D0=DISK5,T0=TAPE5
        COP   D0,,,T0
/*
/&
```

2.6.8. Processing Diskette Files

Before a diskette volume can be used, it must be prepped in either data set label or format label mode. A diskette that has been prepped in data set label mode will be treated as a tape volume (see 2.6.7 for restrictions). A diskette that has been prepped in format label mode will be treated as a disk volume. The librarian operations permitted for the diskette are then determined accordingly.

For a diskette file, the // DVC job control statement must specify a logical unit number of 130 through 159. The logical file name assigned in the FIL librarian control statement specifies the kind of diskette by using a (T*n*) keyword parameter for a data set label diskette and a (D*n*) keyword parameter for a format label diskette.

When processing diskette files, the allocation of one additional buffer of main storage space will improve the performance of the librarian. See 2.6.5 for instructions.

Single-device, multivolume SAT files cannot be copied to diskette.

2.6.9. Processing Card Libraries

When a librarian job is going to produce output on punched cards, the job control stream must contain a device assignment set for a card punch, for example,

```
// DVC 40  // LFD PUNCH
```

On cardless systems, output can be written to a data set label diskette instead of to cards.

2.6.10. Current File Position Pointer

The librarian maintains a pointer in each file it processes. The modules processed by each librarian control statement depend on the options and parameters used in the control statement and the current pointer position in the input file being processed. When a file is opened, the pointer is at the beginning of the file.

A control statement may process one particular module or module group or all modules that meet certain criteria, such as a particular name, type, or name prefix. To process only one particular module, a control statement must include the module name and type. To process only one particular group, the control statement must include the group name and the G option. After the control statement is executed, the pointer remains after the module or group processed.

To locate a particular module or group, the librarian searches the file starting from the current position in the file and wraps around to the beginning of the file, if necessary. For disk and format label diskette files, the librarian searches the file directory. For tape and format label diskette files, it searches the file sequentially. If the librarian cannot locate the particular module, it prints an error message on the librarian map and then processes the next librarian control statement starting from the same position within the file.

To process all modules that meet certain criteria, the control statement must include or omit certain options and parameters. Variations are listed in Table 2-2. In this case, the librarian searches the file for modules that meet these criteria. It always begins the search at the current position in the file. If the U option is not used, it ends the search at the end of the file. In this case, after the control statement is executed, the current position is then at the beginning of the file. If the U option is used, the search ends after a particular module or group and the pointer remains there for the start of the next operation.

When modules are copied to or placed in an output file, modules are always placed at the end of the file.

Should it be necessary to reset the pointer to process a particular module or range of modules in a file, there is a RES librarian control statement to do this (see 3.3.2).

Table 2-2. Control Statement Requirements for Processing Selected Modules

Modules to Be Processed	Name	Requirements Type	Options
All modules with a given name	Module name	No	-
All modules of a given type	Omit	Yes	-
All modules	Omit	No	-
All modules with a given name prefix	Module prefix	No	C
All modules with a given name prefix and type	Module prefix	Yes	C
All modules in the next group with a given name	Group name	No	G
All modules in all groups with a given name	Group name	No	GA
All groups with a given name prefix	Group prefix	No	GC
All modules from the printer position up to and including a particular module	Module name	Yes	U
All modules from the pointer position up to and including the next group with a given name	Group name	Omit	GU

2.6.11. Assigning Version Numbers to SAT Library Modules

To minimize errors in the correction process for SAT library modules, version numbers are assigned to all modules. The first time the librarian copies a module, it expands the header for that module to include a version field. This field consists of three subversion fields in the form of n/n/n where the value of n is a decimal number from 0 to 255. The initial number assigned by the librarian is 0/0/0. The assigned version number remains unchanged until you update it. To assign or change the version number of a SAT library module, use one of the following statements:

```
//PARAM VERASG=n/n/n
```

Specify in your job stream prior to the // EXEC LIBS statement. Specifies the version number n/n/n assigned to module copied or created by the librarian.

```
LIBOP VERASG=n/n/n
```

Specify prior to accessing the library module via a COP, COR, or BLK statement. Specifies the version number (n/n/n) assigned to a module copied or created by the librarian.

```
{COP}
{COR}
{BLK} ,...,VERASG=n/n/n
```

Specify when you want to assign a version number as part of the operation specified by the individual command.

You have the option of specifying one, two, or all three subversion numbers. You cannot omit one subversion number and specify the next.

When correcting a module, you can identify the specific module you want corrected by specifying its version number on the appropriate COR command statement (see 3.3.10). A comparison is made of the version numbers specified and the version numbers of the module being corrected. If they do not match, a diagnostic is printed and the correction process is terminated. Only the version numbers specified are compared; all other levels are ignored.

2.6.12. Printing Library Module Fields before Correction (// PARAM PRTCOR)

Before correcting a field in a SAT librarian object or load module, you can verify its contents by having the librarian print an image of that field. To turn on this librarian feature, specify either:

```
// PARAM PRTCOR=ON
```

or

```
LIBOP PRTCOR=ON
```

When omitted, the print-before-correction-image feature is turned off.

2.7. MIRAM Librarian

2.7.1. MIRAM Librarian Capabilities

The MIRAM librarian is used to maintain the following types of modules:

- Screen format modules
- Help screen modules
- Saved run library modules
- Menu modules

The librarian operations available for MIRAM libraries are limited to the following:

- Copy modules from one file to another
- Delete modules from a file
- Print listings of modules and file directories
- Change a module name
- Change comments in a module header record

2.7.2. MIRAM File Allocation and Management

The MIRAM librarian can only be used to manipulate files once they exist on disk, tape, diskette, or punched cards. The user may allocate a MIRAM file by using either the // EXT job control statement or the interactive services ALLOCATE command. The file type must be MI for MIRAM. A MIRAM tape file is essentially allocated when it is prepped.

Once a MIRAM file has been allocated, modules can be stored in it. The entire file can also be copied or deleted using the MIRAM librarian. The librarian can also be used to print a file directory.

2.7.3. MIRAM Module Structure

A MIRAM module consists of a set of code that is executed at run time. Each module begins with a header record that is formatted as described in Table 2-3.

Table 2-3. MIRAM Module Header Record Format

Byte Position	Contents
0-7	Module name
8-11	Module type
12-14	Creation date (yymmdd)
15-17	Creation time (hhmmss)
18-19	Number of active bytes in the last sector occupied by the module
20-23	Number of data sectors
24-27	Total number of sectors occupied by the module
28-30	Reserved for flags
31	Active flag: X'00' - Delete module X'FF' - Active module
32-33	Record size
34-68	Unused
69-98	Comment field
99-255	Unused

2.7.4. MIRAM Module Creation

Screen format modules contain formatted display screens that are used to supply input and display output at workstations for user application programs. Users create screen format modules by running the screen format generator (SFG) program.

Each time a screen is generated by SFG, two screen format modules are produced. Both modules have the same name, but one is type FC and the other is type F. The type FC module contains the screen format text that is displayed at the workstation when the screen format is used in an application. The type F module contains the control information needed to handle the flow of data passed to and from the application via the screen format.

Help screen modules are provided with the system in the system file YHELP. The user can create additional help screens using the general editor (EDT). Once created, these screens must be saved in the system file YHELP as type H B084 modules using the general editor @WRITE command.

Saved run modules are translated job control streams saved in the system file YSAVE through the job control SAVE option.

Menu modules are created by the menu processor. Refer to the *Menu Services Technical Overview* (UP-9317).

2.7.5. MIRAM Module Management

The MIRAM librarian provides control statements to copy modules, delete modules, print listings of modules, change a module name, and change a comment in a module header record. All MIRAM librarian control statements can be used for any type of module. The use of various options and parameters in the control statements allow the librarian to process one module or all modules in a file with the same name, name prefix, or type.

2.7.6. Using the MIRAM Librarian

The following is a sample job control stream used to run the MIRAM librarian. For more information on job control, see the *Job Control Language Programming Guide* (UP-9986). Figure 2-5 is the librarian map generated by this example. The librarian map lists the volume and actual file labels for files declared on the FIL statement.

```
     1        10      16                                          72
1.   // JOB MLIBRUN
2.   // DVC 20  // LFD PRNTR
3.   // DVC RES  // LBL $Y$FMT  // LFD IN
4.   // DVC 130  // VOL D07041
5.   //  EXT MI,C,2,CYL,2
6.   // LBL SCREEN  // LFD SCRN
7.   // EXEC MLIB
8.   /$
9.            FIL   F0=IN,F1=SCRN
10.           COP   F0,F,SFGSCRN,F1
11.           COP   F0,FC,SFGSCRN,F1
12.           COP.C F0,F,SFSPL,F1
13.           COP.C F0,FC,SFSPL,F1
14.           COP   F0,MENU,,F1
15.           PRT.D F1
16.           GHG   F1,F,SFSPLIO,N,SFSXXXX
17.           CHG   F1,F,SFSXXXX,C,'NAME CHANGE'
18.           DEL.C F1,MENU,OS31
19.           PRT.D F1
20.  /*
21.  /&
```

1. The // JOB statement names the job MLIBRUN.

2. This is the device assignment set for a printer that is needed to print the librarian map. If a printer is not assigned to the job, or a librarian map is not required, the // PARAM PRINT=OFF statement must be used to suppress printing of the librarian map (see 2.6.3).

3. This is a device assignment set for the system file YFMT on the SYSRES volume. This file is given the logical file descriptor of IN, which is then equated to a logical file name F0 in the FIL librarian control statement (line 9). This logical file name is then used to reference the file in all subsequent librarian control statements (lines 10 through 19).

4. through 6.
   This is the device assignment set to allocate a diskette file named SCREEN on the diskette volume D07041. This file is given the logical file descriptor of SCRN, which is then equated to the logical file name F1 in the FIL librarian control statement (line 9). This logical file name is then used to reference the file in all subsequent librarian control statements (lines 10 through 19).

7. Initiates execution of the MIRAM librarain.

8. Identifies the beginning of the librarian control statements.

9. The FIL librarian control statement assigns a logical file name to each file used in the job. These logical file names will be used exclusively to reference these files in subsequent librarian control statements.

10. through 14.
    These COP librarian control statements copy the following modules from the file YFMT to the file SCREEN:

    - The type F screen format module named SFGSCRN
    - The type FC screen format module named SFGSCRN
    - All type F screen format modules with names beginning with FSPL
    - All type FC screen format modules with names beginning with SFSPL
    - All menu modules

15. This PRT librarian control statement prints an alphabetical listing of header records for all modules now in the file SCREEN.

16. This CHG librarian control statement changes the name of the type F screen format module SFSPLIO to SFSXXXX. This module is in the file SCREEN.

17. This CHG librarian control statement inserts the comment 'NAME CHANGE' in the header record for the type F module that is now named SFSXXXX.

18. Deletes all menu modules in the file SCREEN with names beginning with the prefix OS31.

19. This PRT librarian control statement prints an alphabetical listing of all module header records in the file SCREEN. This list can be compared to the list produced by line 15 to verify that the changes and deletions specified by lines 16 through 18 were made.

20. Identifies the end of the librarian control statements.

21. Identifies the end of job.

Notes:

1. *All statements from the / / EXEC MLIB statement to the /* statement, inclusive, comprise a single job step. Therefore, other job steps could be inserted in this control stream between the device assignment statements and the / / EXEC MLIB statement or between the /* and the /& job control statements.*

2. *When a librarian control stream is created on punched cards, a / / FIN job control statement is required after the /& job control statement to end the card reader operation.*

3. *A sample session illustrating the use of the general editor (EDT) to create and run a librarian job control stream interactively at a workstation is given in 3.6.5.*

4. *The device assignment sets vary depending on the files being processed.*

5. *A device assignment set for a system file on the SYSRES volume must include an / / LBL statement.*

6. *When MIRAM files on format label diskettes are being processed, all volumes must be mounted for the duration of the job step. Only one data set label diskette volume can be processed at a time.*

```
UNISYS OS/3 MIRAM LIBRARIAN
DATE  88/05/14  TIME 18.39    JOB   MLIBRUN
                        /$
                        FIL          FO=IN,F1=SCRN
                        COP          FO,F,SFGSCRN,F1
                                                                F       SFGSCRN     80/11/14   20.36
                        COP          FO,FC,SFGSCRN,F1
                                                                FC      SFGSCRN     80/11/14   20.36
                        COP.C        FO,F,SFSPL,F1
                                                                F       SFSPLINF    00/00/00   00.17
                                                                F       SFSPLIO     80/01/20   18.30
                        COP.C        FO,FC,SFSPL,F1
                                                                FC      SFSPLINF    00/00/00   00.17
                                                                FC      SFSPLIO     80/01/20   18.30
                        COP          FO,MENU,,F1
                                                                MENU    OS301       80/08/21   14.30
                                                                MENU    OS302       80/08/21   15.12
                                                                MENU    OS314       80/08/21   14.40
                        PRT.D        F1
                                                                MENU    OS301       80/08/21   14.30
                                                                MENU    OS302       80/08/21   15.12
                                                                MENU    OS314       80/08/21   14.40
                                                                F       SFGSCRN     80/11/14   20.36
                                                                FC      SFGSCRN     80/11/14   20.36
                                                                F       SFSPLINF    00/00/00   00.17
                                                                FC      SFSPLINF    00/00/00   00.17
                                                                F       SFSPLIO     80/01/20   18.30
                                                                FC      SFSPLIO     80/01/20   18.30
                        CHG          F1,F,SFSPLIO,N,SFSXXXX
                        CHG          F1,F,SFSXXXX,C,'NAME CHANGE'
                        DEL.C        F1,MENU,OS31
                                                                MENU    OS314       80/08/21   14.40
                        PRT.D        F1
                                                                MENU    OS301       80/08/21   14.30
                                                                MENU    OS302       80/08/21   15.12
                                                                F       SFGSCRN     80/11/14   20.36
                                                                FC      SFGSCRN     80/11/14   20.36
                                                                F       SFSPLINF    00/00/00   00.17
                                                                FC      SFSPLINF    00/00/00   00.17
                                                                FC      SFSPLIO     80/01/20   18.30
                                                NAME CHANGE     F       SFSXXXX     80/01/20   18.30
                        /*
LIBRARIAN FINISHED
DATE 88/05/14   TIME 18.41
ML024  MLIB TOTAL ERRORS= 00000, UPSI= X'00'
```

Figure 2-5. Sample MIRAM Librarian Map

2.7.7. Creating a Multifile Tape (// PARAM TAPEFILES=MULTI)

The following // PARAM statement allows the librarian to output more than one file to a single-tape volume:

```
// PARAM TAPEFILES=MULTI
```

If used, this statement must be placed between the // EXEC MLIB or // EXEC LIBS statement and the /$ job control statement. If this statement is not used, only one file can be be written to a tape. This statement is not required to read a file on a multifile tape.

A multifile tape is created by copying files from another storage medium to output files on the tape. The tape must already be prepped and it may or may not already contain some files. The files must be copied in sequence as they are to be arranged on the tape. These files cannot be extended later.

The librarian job, which copies the files to the tape, must contain the // PARAM statement. Also, the // LBL job control statement for each new tape file must include a file sequence number parameter (in positional parameter 4) to indicate the position of the output file on the tape. For example, if the tape already contains two files, the // LBL statement for the first new tape file must specify a 3 for the file sequence number. If the tape is a newly prepped tape that does not yet contain any files, the file sequence number for the first new tape file must be 1.

As the files are being output to the tape, only one tape file can be open at a time. Therefore, the librarian job step should use the same logical file name (T*n*) for every tape file but redefine that logical file name each time a new file is to be processed. (See the FIL librarian control statement description for more information.)

Normally, after each file is written to the tape and then closed, the librarian would rewind the tape to the load point. Then it would reopen the tape and advance to the end of the tape again to write the next file. However, a // DD job control statement with a rewind parameter can be used in the device assignment set for each tape to eliminate unnecessary rewinding at each open and close operation.

- The statement // DD OPRW=NORWD specifies no rewind at file open.
- The statement // DD CLRW=NORWD specifies no rewind at file close.

Specifically, the device assignment sets for the tape files should specify

- CLRW=NORWD for the first tape file
- OPRW=NORWD for the last tape file
- Both OPRW=NORWD and CLRW=NORWD for all other files

The following sample job copies five files from disk to the same new tape volume:

```
// JOB MULTFILE
// DVC 20 // LFD PRNTR
// DVC 50  // VOL D00410  // LBL DISKFIL1  // LFD DISK1
// DVC 50  // VOL D00410  // LBL DISKFIL2  // LFD DISK2
// DVC 50  // VOL D00410  // LBL DISKFIL3  // LFD DISK3
// DVC 50  // VOL D00410  // LBL DISKFIL4  // LFD DISK4
// DVC 50  // VOL D00410  // LBL DISKFIL5  // LFD DISK5
// DVC 90  // VOL S01841  // DD CLRW=NORWD
// LBL TFIL1,,,,1  // LFD TAPE1
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL2,,,,2  // LFD TAPE2
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL3,,,,3  // LFD TAPE3
// DVC 90  // VOL S01841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL4,,,,4  // LFD TAPE4
// DVC 90  // VOL S01841  // DD OPRW=NORWD
// LBL TFIL5,,,,5  // LFD TAPE5
// EXEC LIBS
// PARAM TAPEFILES=MULTI
/$
        FIL   F0=DISK1,F1=TAPE1
        COP   F0,,,F1
        FIL   F0=DISK2,F1=TAPE2
        COP   F0,,,F1
        FIL   F0=DISK3,F1=TAPE3
        COP   F0,,,F1
        FIL   F0=DISK4,F1=TAPE4
        COP   F0,,,F1
        FIL   F0=DISK5,F1=TAPE5
        COP   F0,,,F1
/*
/&
```

Section 3
Librarian Control Statements

3.1. Introduction

Librarian control statements define what the librarian is to do. They name the files and modules to be processed and the librarian functions to be performed. They must be included as embedded data within each librarian job control stream that executes the SAT or MIRAM librarian. (See 3.6 for sample librarian job control streams.)

This section describes the syntax and functions for all the SAT and MIRAM librarian control statements in detail. These control statements are executed in the order that they are provided in the job control stream. Except for some related statements that must be used together, any order is permitted.

For convenient reference, some canned librarian job control streams provided with OS/3 are described in 3.5. They should be used instead of the librarian to perform library maintenance on system release volumes. Appendix A summarizes more canned job control streams that are described in other manuals.

3.2. Format Conventions

The recommended coding format for librarian control statements is:

LABEL	ΔOPERATIONΔ	OPERAND	SEQUENCE
	function [.options]	[p1][,p2][,p3][,p4][,p5]	[seq-no]

where:

function
: A mnemonic that identifies the librarian operation.

options
: A string of one or more uppercase letters that specifies the processing options to be used. A period, not a space, is used to separate the function mnemonic and the options.

Examples

```
COP.G
COP.CN
```

The options available with each librarian control statement are listed in each statement description.

p1,p2,p3,p4,p5
: A series of variable parameters that identify the files and modules to be processed. These parameters are positional, meaning that commas are required to indicate where any parameter values have been omitted within the statement. Trailing commas, however, should not be used except in the variation of the COP statement which lists the contents of an entire file without performing a copy operation.

Examples

```
DEL   D1,S,COBOL86,D2
COP.G D2,,ACCTS,D3
COM   D1,,,,D2
COP   D3,S,COBOL55
COP.D D7,
```

seq-no
: A string of up to eight letters or digits specifying the sequence number for a module record. At least one character must be a digit. If letters are used, all letters must be contiguous and precede all the digits.

All sample librarian control statements used in this guide adhere to the recommended format with the function mnemonic starting in column 10, the operands starting in column 16, and the sequence number, if used, starting in column 73. These suggested column positions do not have to be used. The only requirement is that the operation field, which consists of the function mnemonic and any options, must be preceded and followed by at least one space.

3.3. SAT Librarian Control Statement Descriptions

This section contains detailed descriptions of the SAT librarian control statements. Each statement description includes the statement function, syntax description, and syntax examples.

The statements are organized according to major librarian functions, such as listing modules or correcting modules. Related statements are described together. Some statements, such as the COP statement, can serve several functions, depending on the options and parameters used in the statement. In this situation, a separate description of the statement is given for each function. Any background information that is helpful when using a particular statement is provided in the function description.

Sample job control streams for the SAT librarian in 3.6 illustrate how the various librarian control statements work together. Information on the job control required to use the SAT librarian is given in 2.6.

Table 3-1 contains a summary of the functions associated with each SAT librarian control statement. Each statement is described in detail in the section indicated:

Table 3-1. SAT Librarian Control Statement Summary

Statement	Function	Description
BLK	Converts standard load modules to blocked format.	3.3.13
BOG	Writes a beginning-of-group record in a file.	3.3.18
COM	Compares two source modules record by record.	3.3.8
	Compares two files block by block.	3.3.9
COP	Copies modules from one file to another.	3.3.5
	Advances the pointer past a module in a file.	3.3.2
	Prints a sequential file table of contents (see also LST).	3.3.16
	Prints a file directory.	3.3.16
	Prints module listings or punches modules.	3.3.17
COR	Begins a series of module corrections.	3.3.10
DEL	Deletes modules or files.	3.3.6
ELE	Copies a card module to a file.	3.3.4
EOD	Ends a card module being copied (with ELE).	3.3.4
	Ends a series of module corrections (with COR).	3.3.10
	Ends a series of linkage editor control statements (with REPRO).	3.3.19
EOG	Writes an end-of-group record in a file.	3.3.18
ESC	Inserts a series of librarian control statements in a librarian job.	3.3.20
FIL	Assigns logical file names.	3.3.1

continued

Table 3-1. SAT Librarian Control Statement Summary (cont.)

Statement	Function	Description
LST	Prints an alphabetical file table of contents (see also COP).	3.3.16
PAC	Compresses a disk or format label diskette file.	3.3.15
PAGE	Starts a new page on the librarian map and optionally adds or changes the page heading.	3.3.3
REC	Recycles the pointer during source module corrections.	3.3.11
REN	Renames a module, group or record. Optionally changes the comment in a module header.	3.3.14
	Changes the sharability status of an object module.	3.3.14
REPRO	Inserts or deletes linkage editor control statements in an object module.	3.3.19
RES	Resets the pointer in a file.	3.3.2
SEQ	Adds sequence numbers to a source module being copied from cards to tape, disk, or diskette.	3.3.4
	Sequences or resequences a source module.	3.3.7
	Defines the sequence field for source module corrections.	3.3.11
SKI	Skips records during source module corrections.	3.3.11

3.3.1. Assigning Logical File Names (FIL)

Function

A FIL librarian control statement equates each disk, tape, and diskette file name used in an // LFD job control statement to a logical file name.

A logical file name represents a file device type code (D for disk or T for tape) and a logical file number (0 through 15). Format label diskette files must be declared as disk files. Data set label diskette files must be declared as tape files.

Up to 16 files can be declared in a single FIL statement. Commas, not spaces, are used to separate successive declarations. Disk and tape file declarations may be included in the same FIL statement.

The librarian map will list the following information for each logical file name declared in the FIL statement:

- The volume serial number (from the // VOL job control statement)
- The logical file descriptor (from the // LFD job control statement)
- The file label (from the // LBL job control statement)

The FIL statement only assigns logical file names to files; it does not open the files. A file is only opened when its logical file name is used in a subsequent librarian control statement. Up to six files can be open at one time. If more files are used, only the latest six remain open. Otherwise, a file remains open until the job step is completed or until the logical file name is assigned to another file by another FIL librarian control statement in the job step. Therefore, if a librarian job processes many files and it is not necessary to have all the files open at one time, it is a good idea to reuse the logical file names in another FIL librarian control statement. This way, the files will be closed when they are no longer needed (see Example 2).

Note: Because librarian control statements do not reference files by their actual names, the same set of librarian control statements can be used interchangeably to perform the same operations on any files. Only the // LBL and // LFD job control statements, and the FIL librarian control statements need to be changed to identify the new files.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	FIL	{Tn | Dn} =filename-1 [,..., {Tn | Dn} =filename-n]

Options

None

Keyword Parameter

Tn=filename

Equates either a tape or a data set label diskette file name with a logical file name of T0 through T15.

`Dn=filename`

Equates either a disk or format label diskette file name to a logical file name of D0 through D15.

The *filename* must match the file name used in the // LFD job control statement for the file, the system file name, or the job run file name YRUN.

Note: The filename *may contain up to eight alphanumeric characters. The first must be an alphabetic character.*

Examples

```
      1        10    16
1.  // JOB DVCTEST
    // DVC 20   // LFD PRNTR
    // DVC 90   // VOL OTALT   // LBL TPLOAD   // LFD SCRTAPE
    // DVC 91   // VOL OTALT1  // LBL TPDUMP   // LFD UPDATE
    // DVC 50   // VOL OS3ALT  // LBL DSKDMP   // LFD PROCLIB
    // DVC 140  // VOL OS3DST  // LBL DSTLOAD  // LFD DSKTLIB
    // EXEC LIBS
    /$
            FIL   T0=SCRTAPE,T1=UPDATE
            FIL   D0=PROCLIB,T2=DSKTLIB
```

The first FIL statement assigns the logical file name T0 to the tape file SCRTAPE and the logical file name T1 to the tape file UPDATE. The second FIL statement assigns the logical file name D0 to the disk file PROCLIB and the logical file name T2 to the data set label diskette file DSKTLIB.

```
      1        10    16
2.  // JOB FILTST
    // DVC 20   // LFD PRNTR
    // DVC 50   // VOL D00012  // LBL MINE  // LFD MY
    // DVC 50   // VOL D00020  // LBL YOUR  // LFD YR
    // DVC 50   // VOL D00012  // LBL HIS   // LFD HS
    // DVC 50   // VOL D00020  // LBL HERS  // LFD HR
    // EXEC LIBS
    /$
            FIL   D0=MY,D1=YR
            COP   D0,,,D1
            FIL   D0=HS,D1=HR
            COP   D0,,,D1
    /*
    /&
```

The first FIL statement assigns the logical file name D0 to the disk file MINE and the logical file name D1 to the disk file YOUR. The first COP statement copies all modules from the file MINE to the file YOUR.

The second FIL statement closes the file MINE and reuses the logical file name D0 by assigning it to the file HIS. It also closes the file YOUR and reuses the logical file name D1 by assigning it to the file HERS. The second COP statement then copies all modules from the file HIS to the file HERS.

3.3.2. Resetting the Pointer in a File

The RES and COP librarian control statements can move the pointer in a file to determine which modules will be processed by a librarian control statement. Refer to 2.6.10 for a general description of the pointer operation.

Resetting the Pointer to the Start of a File or Module (RES)

Function

The RES librarian control statement repositions the pointer to the beginning of the file or to a particular module or group. If the module or group is not found, the pointer position remains the same and a diagnostic message is printed on the librarian map.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	RES[.options]	[{lfn / YRUN}] [,{S / O / L / M}] [,name]

Options

G - The name parameter is a group name. Reset pointer to the first group encountered with the specified name.

Positional Parameter 1

lfn
Specifies the logical file name of the file in which the pointer is to be reset.

If omitted, the job run file YRUN is used.

Positional Parameter 2

S,M,O,L
Specifies the module type as either program source (S), macro/jproc source (M), object (O), or load (L).

If used, a name parameter is also required to specify a particular module name. If omitted, the pointer is reset to the beginning of the file specified in the lfn parameter. The type parameter is not valid when the G option is used.

Positional Parameter 3

name

Specifies the name of the module or module group (with G option) to which the pointer is to be reset.

When a module name is used, a type parameter is also required; otherwise the pointer is reset to the beginning of the file specified by the lfn parameter. When a group name is used, the G option must also be used and the type parameter must be omitted.

Examples

```
    1         10    16
1.|           RES   D1
2.|           RES   D3,O,EXAMPLE2
3.|           RES   T1,S,EXAMPLE3
4.|           RES   ,L,EXAMPLE4
5.|           RES.G D5,,GROUP5
```

1. Resets the pointer to the beginning of file D1.
2. Resets the pointer in file D3 to the object module named EXAMPLE2.
3. Resets the pointer in tape file T1 to the source module named EXAMPLE3.
4. Resets the pointer in the job run file YRUN to the load module EXAMPLE4.
5. Resets the pointer in file D5 to the module group named GROUP5.

Resetting the Pointer Past a Module (COP)

Function

The COP librarian control statement can reset the pointer past a module without performing any copy operation. It resets the pointer to the module following the module specified in the COP statement and prints that module's header information on the librarian map. If the module is not found, a diagnostic message is printed on the librarian map and the pointer position remains the same.

Note: *Other versions of the COP statement are available to copy modules (3.3.5), print a file table of contents (3.3.16), and list or punch modules (3.3.17).*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COP	[{lfn / YRUN}], {S / M / O / L},name

Options

None

Positional Parameter 1

lfn
: Specifies the logical file name of the file in which the pointer is to be reset.

If omitted, the job run file YRUN is used.

Positional Parameter 2

S,M,O,L
: Specifies the module type as either program source (S), macro/jproc source (M), object (O), or load (L).

name
: Specifies a module name. The pointer is advanced past this module.

Examples

```
    1         10    16
1.  |         COP   D1,S,EXAMPLE1
2.  |         COP   ,O,EXAMPLE2
```

1. Resets the pointer in file D1 to the module following the source module named EXAMPLE1.

2. Resets the pointer in the job run file YRUN to the module following the object module named EXAMPLE2.

3.3.3. Controlling Page Advancement for the Librarian Map (PAGE)

Function

The PAGE librarian control statement starts a new page on the librarian map. It may also specify a header line to be printed at the top of each new page. This header line remains in effect for the duration of the librarian job step or until it is changed by another PAGE control statement.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	PAGE	['header-line']

Options

None

Positional Parameter 1

'header-line'
Specifies the header line to be printed at the top of each succeeding page. It can contain up to 64 characters and must be enclosed in single quotation marks.

This header line remains in effect for the duration of the job or until it is changed. If omitted, the current header line, if any, remains in effect.

Examples

```
   1        10    16
1. |        PAGE
2. |        PAGE  'PAYROLL MODULES (CONTINUED)'
3. |        PAGE  ' '
```

1. Starts a new page on the librarian map, printing the current header line, if any, at the top of that page.

2. Starts a new page on the librarian map, using the new header line PAYROLL MODULES (CONTINUED) on that page and each succeeding page.

3. Starts a new page on the librarian map and ends the use of any previously specified header.

3.3.4. Adding Modules from Cards to a Disk, Tape, or Diskette File

A program module punched on cards can be copied to a file using the librarian. The ELE and EOD librarian control statements identify the beginning and end of the module. When a program source or macro/jproc source module is being copied, a SEQ statement can be used following the ELE statement to sequence the records in the module after they are copied.

Delimiter Cards (ELE and EOD)

Function

The ELE and EOD librarian control statements delimit a card module to be copied to a file. For disk or format label diskette files, if the file already contains a module with the same name and type, the old module is deleted and the new module is added to the end of the file. This effectively updates the module. In a tape or data set label diskette file, the module is always added to the end of the file; any old version is never deleted.

The librarian uses the information in the ELE statement to construct the module header record. This statement identifies the file where the module is being copied, the module name, the module type, and an optional comment.

A complete librarian job control stream may be stored as a source module. It must include all the cards required in the job from the // JOB card to the /& end-of-job card. Each EOD statement in the card module being stored should have a corresponding COR or ELE statement. The first EOD statement that is not associated with an ELE statement is treated as the end of the module being stored, unless the E option is used in the ELE statement.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	ELE[.options]	[{lfn / $SYSRUN}],{S / M / O / L},name[,comment]

LABEL	ΔOPERATIONΔ	OPERAND
unused	EOD	unused

Options

D - Print a listing of the module on the librarian map.

E - Terminate the module at the first EOD librarian control statement.

N - Do not list the module header information on the librarian map.

P - Punch a copy of the module being copied.

Positional Parameter 1

lfn
Specifies the logical file name of the file to which the module is being copied.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M,O,L
Specifies the type of module being added as either program source (S), macro/jproc source (M), object (O), or load (L).

Positional Parameter 3

name
Specifies the name to be given to the module after it is copied to the file.

For source modules, this may be the same as the name used for the card module or a new name. For object or load modules, it must be the same name used for the card module.

Positional Parameter 4

comment
Specifies up to 30 characters to be included in the module header record.

If omitted, no comment is included in the module header record.

Example 1

```
1         10    16
          ELE   D1,S,EXAMPLE1,REVISED 6/13/88
                .
                .   } Source module card deck
                .
          EOD
```

Copies a source module on cards to the end of the disk file D1, giving it the name EXAMPLE1. The module header record includes the comment REVISED 6/13/88.

Example 2

```
1         10    16
          ELE   ,L,EXAMPLE2
                .
                .   } Load module card deck
                .
          EOD
```

Copies the load module on cards to the end of the job run file YRUN, giving it the name EXAMPLE2.

Example 3

```
1         10    16
          ELE  D2,S,LIBJOB
// JOB LIBJOB
.
.    } Complete librarian job control stream to be stored
.
/&
EOD
```

Adds the complete librarian job control stream contained on cards to the end of file D2, giving it the name LIBJOB. (See 3.6.4 for a complete sample job control stream.)

Sequencing a Card Module Being Copied (SEQ)

Function

The SEQ librarian control statement can be used with the ELE control statement to add, check, or change sequence numbers in a module as it is being added to a file. This feature is available for program source and macro/jproc source modules only.

The SEQ statement must be placed immediately after the ELE statement and before the first card in the module being copied.

Note: Other variations of the SEQ statement are used to sequence or resequence a module without performing any other operation (3.3.7) or to sequence a module as it is being corrected (3.3.11).

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	SEQ	[{lfn / $YSRUN}] , {S / M} [,name] [, {column / 73}] [, {content / SAME / 00000000}] [, {increment / 1}]

Options

None

Positional Parameter 1

lfn

Identifies the file containing the module to be resequenced. This must match the lfn used in the preceding ELE librarian control statement.

Positional Parameter 2

S,M

Identifies the type of module being sequenced as either program source (S) or macro/jproc source (M).

Positional Parameter 3

name

Specifies the name of the module being sequenced.

If used, it must match the module name used in the preceding ELE librarian control statement. If omitted, the librarian will only verify existing sequence numbers in the module.

Positional Parameter 4

column

Specifies the starting column for the sequence number in each module record. If omitted, column 73 is assumed. The length of the content parameter determines the length of the sequence number field.

Positional Parameter 5

content
: Specifies the sequence number to be used for the first record in the module. This string can be from one to eight characters long. The length of this string determines the maximum length for any sequence number in the module. It may contain letters, digits, or both. If any letters are used, all the letters must be contiguous and precede any digits. For example, MA400, GRP000, and 10000001 are valid sequence numbers. However, M4A00, 600MAST, and 22AAA33 are not valid.

SAME
: Specifies that the same sequence number is to be used for the first record in the module.

00000000
: Eight zeros is the default sequence number for the first record in the module.

If this parameter is omitted, the sequence number for the first record is 00000000 (eight zeros).

Positional Parameter 6

increment
: Specifies decimal number from 0 to 255, to be used as an increment for sequencing successive records. If omitted, an increment of 1 is used.

Example 1

```
1         10    16
|
|       ELE   D1,S,EXAMPLE1
|       SEQ   D1,S,EXAMPLE1,,EXA00000,10
|             .
|             .    } Source module card deck
|             .
|       EOD
```

Copies the source module named EXAMPLE1 to file D1 and sequences (or resequences) the records using columns 73 through 80. The sequence number for the first record is EXA00000 and the increment is 10.

Example 2

```
1         10    16
|
|         ELE   D2,S,EXAMPLE2
|         SEQ   D2,S,EXAMPLE2,1,EXA00001
|               .
|               .     } Source module card deck
|               .
|         EOD
```

Copies the source module named EXAMPLE2 to file D2 and sequences (or resequences) the records using columns 1 through 8. The sequence number for the first record is EXA00001, and the increment is 1 (the default).

Example 3

```
1         10    16
|
|         ELE   D3,S,EXAMPLE3
|         SEQ   D3,S,,,EXA0000,1
|               .
|               .     } Source module card deck
|               .
|         EOD
```

Copies the source module named EXAMPLE3 to file D3 and checks its sequence numbers. The numbers should begin in column 73 and be seven characters long like EXA0000. The sequence number of the first record should be EXA0000, and the increment should be 1.

3.3.5. Copying Modules and Files (COP)

Function

The COP librarian control statement copies modules from one file to another. These files need not be stored on the same type of storage medium. Modules are obtained from an input file and are copied to the end of an output file. Therefore, after a module is copied, both files will contain a copy of the module. Also, any modules contained in the output file before the copy operation will still be there.

The options and parameters used in the COP statement determine which modules are copied. Options can also cause the copied modules to be listed on the librarian map or punched on cards.

When an entire disk or format label diskette file is copied, any deleted modules in the input file are not copied to the output file.

If the output file is a disk or format label diskette file that already contains a module of the same name and type as the one being copied, the one already in the output file is deleted and a new one is copied from the input file.

If the output file is a tape or data set label diskette, all the modules specified in the COP statement are copied to the end of the file, even if the file already contains any modules with the same name and type as any of those being copied. See 2.6.7 for a description of how the librarian accesses these duplicate modules in tape files.

Notes:

1. *When modules are being copied to a new tape file, the tape must first be prepped to initialize the new file. Otherwise, the copied modules will be placed at the end of an existing file on the tape. Also, the input and output files must be in separate tape volumes.*

2. *An ICAM symbiont should not be copied to a file that contains an active ICAM symbiont with the same name. This would cause the active ICAM symbiont to be deleted.*

3. *Single-device, multivolume tape files cannot be copied to diskette.*

4. *Other variations of the COP statement are used to perform the following operations (see the section indicated for more information):*

   - *Print header information for modules in a file (see 3.3.16).*
   - *Print listings of modules or punch copies of modules (see 3.3.17).*
   - *Reset the pointer in a file (see 3.3.2).*

5. *The SETREL and COPYREL canned job control streams are provided in the system to back up and copy the contents of the system release volume. See Appendix A.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COP[.options]	[{input-lfn / YRUN}][,{S / M / O / L}][,name],output-lfn[,VERASG=n/n/n]

Options

A - Copy all groups with the group name specified in the name parameter. (The G option must also be used.)

C - Copy all modules from the pointer position in the input file to the end of the input file whose names begin with the prefix specified in the name parameter. If both a name and type parameter are used, copy all modules of that type with that name prefix. If the G option is also used, copy all module groups with the group name prefix specified by the name parameter.

D - Print a listing of each module copied.

G - If the C option is not used, the name parameter specifies a complete group name. Copy the next group with that name. If the A option is also used, copy all groups with that group name from the pointer position to the end of the input file.

If the C option is used, the name parameter specifies a group name prefix. Copy every group with that name prefix from the pointer position to the end of the input file.

M - Perform updates only. Only copy a module from the input file if it has the same name and type as a module already in the output file.

N - Do not list header records on the librarian map.

P - Punch the modules copied.

Q - Add new modules only. Only copy a module from the input file if it does not have the same name and type as a module already in the output file.

U - Only copy modules from the pointer position in the input file up to and including the module specified by the name and type parameters. If the G option is also used, copy all modules from the pointer position up to and including the next group with the name specified in the name parameter.

Note: An error occurs if the C and U options are used together.

Positional Parameter 1

`input-lfn`
Specifies the logical file name of the input file.

If omitted, the job run file YRUN is used.

Positional Parameter 2

`S,M,O,L`
Specifies the type of module being copied as either program source (S), macro/jproc source (M), object (O), or load (L).

If omitted, all modules with the specified name are copied. If both name and type parameters are omitted, all modules from the pointer position to the end of the input file are copied. The type parameter is not valid when the G option is used.

Positional Parameter 3

name

Specifies the module name, module group name (with G option), module name prefix (with C option) or module group name prefix (with C and G options) for the modules to be copied.

This parameter is optional, unless the C or G option is specified. If omitted, all modules of the specified type from the pointer position to the end of the input file are copied. If both the name and type parameters are omitted, all modules from the pointer position to the end of the input file are copied.

Positional Parameter 4

output-lfn

Specifies the logical file name of the file to which the modules are to be copied.

Keyword Parameter

VERASG=n/n/n

Specifies the version number to be applied to the module being copied. (Should be specified as last parameter when used with positional parameters.)

Examples

```
   1        10    16
1. |        COP.D D1,S,MYMOD,D2
2. |        COP.UN D0,M,MYMOD,T3
3. |        COP.C T4,L,MY,D5,VERASG=6
4. |        RES   D6
   |        COP.C D6,S,COB,D7
5. |        COP.Q D8,,,D9
6. |        COP   D10,O,,D11
```

1. Copies the source module MYMOD from file D1 to file D2 and prints a listing of the module MYMOD.

2. Obtains all modules from the pointer position in file D0 up to and including the procedure module MYMOD and copies them to file T3. The module header information for these modules is not printed on the librarian map.

3. Finds all load modules whose names begin with MY from the pointer position to the end of file T4 and copies them to file D5. Assigns version number 6 to each module copied.

4. Resets the pointer to the beginning of file D6. Finds all source modules with the name prefix COB in file D6 and copies them to file D7.

5. Finds all modules from the current position in file D8 to the end of file D8 that do not exist in file D9 and copies them to the end of file D9.

6. Finds all object modules from the pointer position in file D10 to the end of file D10 and copies them to the end of file D11.

3.3.6. Deleting Modules and Files (DEL)

Function

The DEL librarian control statement logically deletes modules from a disk or format label diskette file. Deleted modules cannot be accessed, but they still occupy space in the file until the file is compressed by a PAC operation (see 3.3.15) or a COP operation that copies an entire file (see 3.3.5).

When the DEL statement is used to delete a load module, only the header record for the root phase is printed on the librarian map, even though all overlay phases are also always deleted.

Notes:

1. *An ICAM symbiont should not be deleted while it is actively processing.*

2. *The DEL statement cannot be used to delete modules from a tape or a data set label diskette file. The only way to eliminate unwanted modules in a tape file is to make a new tape, copying only the modules to be kept to the new tape file.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	DEL[.options]	[{lfn / $YSRUN}][.{S / O / M / L}][,name]

Options

A - Delete all groups with the group name specified by the name parameter. (The G option must also be used.)

C - Delete all modules from the current position to the end of the file whose names begin with the prefix specified by the name parameter. If the G option is also used, delete all groups whose names begin with the prefix specified by the name parameter.

G - The name specified in the name parameter is a group name or group name prefix. If the C option is not used, the name parameter is a complete group name. Delete all modules in the next group with that group name. If the A option is also used, delete all groups with that group name.

If the C option is used, the name parameter specifies a group name prefix. Delete all module groups with that name prefix from the pointer position to the end of the file.

N - Do not list module header records on the librarian map.

U - Delete all modules from the pointer position up to and including the module with the specified name and type. If the module name and type parameters are omitted, delete all modules from the pointer position to the end of the file. If the G option is also used, delete all modules from the pointer position up to and including the specified group.

Note: An error occurs if the C and U options are used together.

Positional Parameter 1

lfn
Specifies the logical file name of the disk or format label diskette file containing the modules to be deleted.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M,O,L
Specifies the type of modules to be deleted as either program source (S), macro/jproc source (M), object (O), or load (L).

If omitted, all modules with the specified name are deleted. If both the name and type parameters are omitted, all modules in the file are deleted. The type parameter is not valid when the G option is used.

Positional Parameter 3

name
Specifies the module name, module group name (with the G option), module name prefix (with the C option), or the group name prefix (with the C and G options) for the modules to be deleted.

If omitted, all modules of the specified type are deleted. If both the name and type parameters are omitted, all modules in the file are deleted. The name parameter is required when the C or G options are used.

Examples

```
     1         10    16
1.   |         DEL.D D1,S,EXAMPLE1
2.   |         DEL.P ,O
3.   |         DEL.C D2,O,EXA
4.   |         DEL.UN D0,L,MYMOD
```

1. Deletes and prints a listing of the source module EXAMPLE1 in file D1.
2. Deletes and punches all object modules in the job run file from the current position to the end of the file.
3. Deletes all object modules from the current position to the end of file D2 whose names begin with the letters EXA.
4. Deletes all modules from the current position in file D0 up to and including the load module MYMOD. Does not print the module header information for these modules on the librarian map.

3.3.7. Sequencing and Resequencing Source Modules (SEQ)

Function

The SEQ librarian control statement sequences or resequences records in a program source or a macro/jproc source module. It cannot be used for object or load modules. If the module does not contain sequence numbers, they are added as specified by the parameters and options in the SEQ statement. If the module already contains sequence numbers they are changed as specified.

Note: Other variations of the SEQ statement are used in conjunction with the ELE (see 3.3.4) and the COR (see 3.3.10) librarian control statements.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	SEQ[.options]	[{lfn / SYSRUN}], {S / M}, name [, {column / 73}] [, {content / SAME / 00000000}] [, {increment / 1}]

Options

D - Print a listing of the module on the librarian map after the module is sequenced or resequenced.

N - Do not list the module header information on the librarian map.

P - Punch the module after it is sequenced or resequenced.

Positional Parameter 1

lfn

Specifies the name of the file containing the module to be sequenced.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M

Specifies the type of module being resequenced as either program source (S) or macro/jproc source (M).

Positional Parameter 3

name

Specifies the name of the module to be sequenced or resequenced.

Positional Parameter 4

column

Specifies the starting column position for the sequence number in each module record.

If omitted, column 73 is used.

Positional Parameter 5

content

Specifies the sequence number to be assigned to the first record in the module. It can be from one to eight characters long. The number of characters used determines the maximum length for any sequence number in the module. It may contain letters, digits, or both. If any letters are used, all letters must be contiguous and precede any digits. For example, MA400 and GRP000 are valid. However, M4A00 and 600MAST are not.

SAME

Specifies that the sequence number for the first record in the module is to remain the same. This is used only if the module is being resequenced.

If this parameter is omitted, the number 00000000 (eight zeros) is used for the sequence number of the first record.

Positional Parameter 6

increment
: Specifies the increment to be used to sequence successive records. It can be any decimal number from 0 to 255.

If omitted, the value 1 is used for the increment.

Examples

```
   1         10    16
1. |         SEQ.DP D14,S,EXAMPLE1,20,LNK000,10
2. |         SEQ.N D12,S,EXAMPLE2,,SAME
```

1. Sequences the source module named EXAMPLE1 in file D14. Sequence numbers are placed in columns 20 through 25. The sequence number for the first record is LNK000 and the increment is 10. The sequenced module is listed and punched.

2. Resequences the source module EXAMPLE2 in file D12, using the same sequence number for the first record and an increment of 1 (the default). Each sequence number is in columns 73 through 80 (the default). The module header information is not printed on the librarian map.

3.3.8. Comparing Source Modules (COM)

Function

The COM librarian control statement can compare two program source or macro/jproc source modules record by record.

The two source modules must be in separate files and have the same name and type. They must also use the same column positions for the sequence number field. The SEQ librarian control statement (see 3.3.7) may be used to reposition the sequence numbers before the compare operation is done.

To perform a compare operation, the librarian first locates the modules in the two files and then starts comparing them record by record. When it finds a difference between corresponding records, it lists the two records on the librarian map. It then compares the sequence numbers of those two records. If one sequence number is lower, the librarian advances to the next record in that module. If the two sequence numbers are equal, it advances to the next record in both modules. Then it compares the new pair of records. When the librarian reaches the end of one module, it prints any remaining records in the other module.

Figure 3-1 illustrates part of a librarian map for a module compare operation. It contains the COM statement followed by the two module header records, one below the other. Next is the heading SOURCE RECORDS THAT ARE NOT EQUAL. Below that is each pair of different records, one below the other.

The first record in each pair belongs to the module whose header record is listed first. The second record belongs to the module whose header record is listed second. If one module ends before the other, an END PRIME MODULE or END OF SECONDARY MODULE message is printed followed by the remaining records in the other module.

Note: The COM statement used to compare two complete files is described in 3.3.9.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COM	[{prim-lfn / YRUN}],[{S / M}][,{n-n / 73-80}],name,sec-lfn

Options

None

Positional Parameter 1

prim-lfn
: Specifies the logical file name of the file containing the first module being compared.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M
: Specifies the type for both modules being compared as either program source (S) or macro/jproc source (M).

Both modules must be the same type. This parameter is required when comparing two modules. If omitted, the two files specified by the prim-lfn and sec-lfn are compared as described in 3.3.9.

Positional Parameter 3

n-n
: These two integers separated by a hyphen specify the column range used for sequence numbers in the records of the two source modules being compared.

Both modules must have sequence numbers in these columns. If omitted, the column range 73 through 80 is assumed.

Positional Parameter 4

name
: Specifies the name of both modules being compared.

Both modules must have the same name. This parameter is required when comparing two modules. If omitted, the two files specified by the *prim-lfn* and *sec-lfn* are compared as described in 3.3.9.

Positional Parameter 5

sec-lfn
: Specifies the logical file name of the file containing the second module being compared.

Examples

```
     1         10    16
     ┌──────────────────────────────────
1.   │         COM   D1,S,,EXAMPLE1,D2
2.   │         COM   D3,S,1-8,EXAMPLE2,D4
3.   │         COM   ,S,,EXAMPLE3,D5
```

1. Compares the source module named EXAMPLE1 in file D1 with the source module named EXAMPLE1 in file D2. The sequence numbers are in columns 73 through 80 (the default) of each source module record.

2. Compares the source module named EXAMPLE2 in file D3 with the source module named EXAMPLE2 in file D4. The sequence numbers are in columns 1 through 8 of each source module record.

3. Compares the source module named EXAMPLE3 in the YRUN file with the source module named EXAMPLE3 in file D5. The sequence numbers are in columns 73 through 80 (the default).

```
UNISYS OS/3 LIBRARIAN REVISION
                    CONTROL        COMMAND        OPERAND
                                     FIL    D1=JCSLIB1,D2=JCSLIB2,D3=JCSLIB3,D4=JCSLIB4
                                     COM    D2,,LIBSFOR,D4
SOURCE HEADER       NAME           DATE       TIME      COMMENTS

                    LIBSFOR        042274     1412
                    LIBSFOR        042274     2210
SOURCE RECORDS THAT ARE NOT EQUAL

                                   LA     R1,1                PUT ONE INTO REGISTER        ABCD0600
                                   MVI    LBSCOM,X'00'                                     ABCD0600

                          LBSCOMA3 MVI    LBSSWIT1,X'90'      RUNLIB NO TYPE CHECK         ABCD0800
                                   MVI    LBSCPRINT                                        ABCD0800

                                   CLI    0(R7),C'T'          IF EQ. TAPE ERROR            ABCD0830
                                   USING  R4,5,6                                           ABCD0830

                          *        END OF SYNTAX CHECK                                     ABCD0880
                                   USING  R4,5,6                                           ABCD0880

                                   STH    R5,LBSCOMS1+4            DISPLACEMENT INTO CODING ABCD3190
                                          MVI              LBSCPRINT,X'00'                 ABCD3190

                          LBSABUFF DC     A(LBSIMBUF+36)                                  ABCD4920
                                   SWTCO  EQU              *                               ABCD4920

                                   END OF PRIME MODULE
```

Figure 3-1. Sample Librarian Map for a Source Module Compare Operation

3.3.9. Comparing Files (COM)

Function

The COM librarian control statement can compare two library files containing any assortment of modules.

The librarian compares two files block by block in their entirety. When it finds a difference, it lists the two blocks in hexadecimal format side by side on the librarian map. It then advances each file pointer one block and does the next comparison. It continues this process until it reaches the end of one file. It then lists any remaining blocks in the other file on the librarian map. Figure 3-2 illustrates part of the librarian map for a file comparison operation. The block-by-block listing begins immediately below the COM statement.

Notes:

1. *Two files containing the same modules may not always compare exactly. To ensure a correct comparison, both files should first be compressed, or packed, using the PAC librarian control statement (see 3.3.15). This is especially true for files assembled and managed using a librarian from Release Level 7 or earlier. When the librarian from Release Level 8 forward is used, the packed and unpacked versions of two files containing the same nondeleted modules in the same sequence will match.*

2. *The COM statement can also be used to compare two source modules record by record as described in 3.3.8.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COM	[{ prim-lfn / YRUN }],,,,sec-lfn

Options

None

Positional Parameter 1

prim-lfn
Specifies the logical file name of the first file being compared.

If omitted, the job run file YRUN is assumed.

Positional Parameters 2 through 4

These three parameters should not be used when comparing two files. The four comma separators, however, are required between the prim-lfn and sec-lfn parameters. If a module name is included as positional parameter 4, the librarian also expects values for positional parameters 2 and 3 to do a module comparison as described in 3.3.8. If a module name is not included, any values in parameter positions 2 and 3 are ignored.

Positional Parameter 5

sec-lfn
: Specifies the logical file name for the second file being compared.

Examples

```
      1         10    16
1.  |           COM   D1,,,,D2
2.  |           COM   ,,,,D3
```

1. Compares files D1 and D2 block by block.

2. Compares the YRUN file and file D3 block by block.

Figure 3-2 illustrates the beginning and end of the librarian map for a block-by-block comparison of two files.

BLOCK	REC	NAME	TYPE	DATE	TIME	COMMENTS

PRIME FILE BLOCK NO. 000001 BLOCK LENGTH F7 SECOND FILE BLOCK NO. 000001 BLOCK LENGTH F7

000001F7	00D7C1C3	D2C8C503	07A40000	013501C3	E2D7C1C3	000001F7	00D406C4	C4F14040	40A40000	01054D06	C4C4F240
D240A400	0005BF07	D2C8C503	07F0F090	00000758	E3C5E2E3	4040A400	00043404	D6C4C4F3	404040A4	00000543	D4D6C4C4
C3E2F0F0	90000008	4003C9E2	E3C80740	40A40700	0C15C1E2	F4404040	A4000007	05D4D6C4	C4F54040	40A40000	0305D4D6
04D7D9C7	4040A400	000E96C7	C5E3D4D3	C9C24080	07070F7E	C4C4F640	4040A400	0008A8D4	D6C4C1F1	40404040	03000976
C7C5E3D4	D3C9C240	0870000F	6703C9C2	C8C4D940	40040000	D406C4C1	F2404040	A400000F	1F04D6C4	C1F3F0F0	F0900000
0FCCD3C9	C2E3E7E3	4040D400	00JF0007	D9C6C9D3	C5404004	0F83D4D6	C4C1F4F0	F0F09000	00134004	D6C4C1F5	40404040
J0000FCE	D9C9C2E3	E7E34040	04000010	0509C9C2	C8C4D940	00001415	04D6C4C1	F6404040	A4000016	964D06C4	C1F74040
40040000	1016D709	C9C24040	40404040	00102703	C9E2E3C8	4080D000	177EC7C5	E3D4D3C9	C2400800	00178703	C9C2C8C4
C5D3D7A4	00001305	C7C5E3D4	D3C9C240	A4000014	87D9C4C8	09404004	000017CC	D3C9C2E3	E7E34040	04000017	0007D9C6
C5D3D7F0	F0900000	2805C8C5	0307D7C1	C3D2A400	00301503	C9D3C540	40040000	17EED9C9	C2E3E7E3	40400400	00180509
C1D9D4C1	C9D5E3A4	00003140	00000000			C9C2C8C4	09404004	00001816	00000000		

PRIME FILE BLOCK NO. 000002 BLOCK LENGTH F7 SECOND FILE BLOCK NO. 000002 BLOCK LENGTH F7

000002F7	00C3C4C3	06D7E8F0	F0900000	3322C3C1	D9D4D5E3	000002F7	00D7D9C9	C2404040	40040000	18270406	C4C3F14C
F0F09000	00534101	C4C8C503	07F0F090	00008815	C1C4C4C8	4040A400	00206504	D6C4C3F2	404040A4	00002222	D4D6C4C3
C5D3D740	A4000093	B4E2E8E2	07090602	C5A40000	9581E2C6	F3404040	A4000023	22D406C4	C3F44040	40A40000	271ED406
C3E3C5E2	E340A4C0	0096A1E2	C6C3D1C3	E24040A4	00009A8E	C4C3F5F0	F0F00000	00240504	D6C4C3F6	404040A4	00002815
E2C6C3E3	E2E3F0F0	9000009F	60E2C6C3	D9E40540	40A40000	D4D6C4C3	F7F0F0F0	00000020	B404D6C4	C3F8F0F0	F0900000
9EC5C4C4	D407E2C3	D9054000	00A07BC4	E4040702	C3F0F090	34A404D6	C4C2F140	40404400	003A3904	D6C4C2F2	40404040
00D0A1AD	D709E2C3	D905F0F0	90000004	A4C85809	F0F04040	00003005	D4D6C4C2	F3F0F0F0	90000004	780406C4	C2F44040
40A40000	AF39C858	D9F0F140	4040A400	00827108	5809F0F2	41A40000	5315D406	C4C2F540	4040A400	00548F04	D6C4C2F6
40404004	00000376	C858D9F0	F0F04040	A4000085	40C85809	F0F0F090	00005622	D406C4C2	F7F0F0F0	90000076	41D406C4
F0F0F140	40A40000	B67AC858	D9F0F0F2	4040A400	00873208	C2F8F0F0	F0900000	A515D406	C4C5F140	4040A400	00868404
5809F0F0	F4404004	00003805	00000000			D6C4C5F2	40404004	00008840	00000000		

PRIME FILE BLOCK NO. 000003 BLOCK LENGTH 34 SECOND FILE BLOCK NO. 000003 BLOCK LENGTH 68

00000334	00C858D9	F0F0F540	40A40000	3987C858	D9F0F0F6	00000368	00D4D6C4	C5F34040	40A40000	8941D406	C4C5F440
4040A400	00BA6208	5809F0F0	F7404004	00008C05	C5D5C4D3	4040A400	006A9503	D6C1C4E2	40404040	00008834	D4D6C4C3
C9C24040	A1000008C	74E2E8E2	07090602	C5A40000	9581E2C6	F5F0F0F0	90000088	62D406C4	C3E7F0F0	F0900000	3F0504D6
C3E3C5E2	E340A400	0096A1E2	C6C3D1C3	E24040A4	00009A8E	C4C3F8F0	F0F09000	00C6A403	D6C1C4E2	40404040	03000C39
E2C6C3E3	E2E3F0F0	90000090	60E2C6C3	D9E40540	40A40000	C5D5C4D3	C9C24040	A1000008C	430406C4	C3E8F0F0	F0900000
9EC5C4C4	D4D7E2C3	D9054400	00A07BC4	E4040702	C3F0F090	34A404D6	C4C2F140	40404400	003A3904	D6C4C2F2	40404040
00D0A1AD	D709E2C3	D905F0F0	90000004	A4C85809	F0F04040	00003005	D406C4C2	F3F0F0F0	90000004	780406C4	C2F44040
40A40000	AF39C858	D9F0F140	4040A400	00827108	5809F0F2	41A40000	5315D406	C4C2F540	4040A400	00548ED4	D6C4C2F6
40404004	00000376	C858D9F0	F0F04040	A4000085	40C85809	F0F0F090	00005622	D406C4C2	F7F0F0F0	90000076	41D406C4
F0F0F140	40A40000	B67AC858	D9F0F0F2	4040A400	00873208	C2F8F0F0	F0900000	A515D406	C4C5F140	4040A400	00868405
05C403C9	C2404041	00005700	00000000			05C4D3C9	C2404041	00008840	00000000		

PRIME FILE BLOCK NO. 000001 BLOCK LENGTH EE SECOND FILE BLOCK NO. 000001 BLOCK LENGTH E3

000001EE	0038A4C0	00000000	00000000	00000007	C1C302C8	000001E3	0038A400	00000000	00000000	00000004	D6C4C4F1
C5D3D731	04270914	52404040	40404040	40404040	40404040	40404080	08081634	24404040	40404040	40404040	40404040
40404040	40404040	40404040	40404010	2407C1C3	D2C8C5D3	40404040	40404040	40404040	4040402F	240DF0F0	404009E4
0740E2E3	C1D9E340	F00C2509	09C2C103	D94040F1	F268F00C	D540D7D9	D6C3C5E2	E2D60940	C505C3D6	E405F3C5	02C5C440
250909C4	E2C905C7	405C68F1	F2002500	09D7D9C9	05E34005	C5D9D9D6	0940C3D6	C4C54305	05D5D538	25380401	D540C5D9
06C7C5D5	17251409	D607C5D5	404003C9	C2E3E7E3	684009C9	D9D6D940	E6C1E240	C4C5E3C5	C3E3C5C4	40C4E4D9	C2D5C740
C2E3E7E3	50172514	09D607C5	D5434003	C9C2C8C4	D9634009	C1D540C5	E7C3D740	C6E405C3	E3C9D6D5	4B40C103	D34006C6

Figure 3-2. Sample Librarian Map for a File Compare Operation (Part 1 of 3)

```
                                                                                          PAGE # 0003
BLOCK   REC          NAME       TYPE      DATE      TIME       COMMENTS
C9C2C8C4  D95D1725  14090607  C5D54040  D3C9C2D6  E4E3684D     40E3C8C5  3A253704  C5D9D9D6  D940C3D5  C4C5E240  C9D5C4C9
D9C9C2D6  E4E3500E  25010903  C10704F0  687EF3C6  7DF07D1A     C3C1E3C5  C440C2E8  40D5D5D5  D540D9C5  C6C5D940  E3D640E6
251709C4  D4E2C5D3  40D3C9C2  C8C4D968  D3C9C26B  C8C9D568     D9D6D5C7  40C9D5C6  D6D9D4C1  E3C9D6D5  00000000  00000000
40F05000  00000300  00000000  00000000                         00000000  00000000  00000000  00000000

PRIME FILE      BLOCK NO. 000002    BLOCK LENGTH  E5           SECOND FILE    BLOCK NO.  000002   BLOCK LENGTH E2

000002E5  00112508  0004C1C9  D5D306D6  D740C5D8  E400035C     000002E2  00392536  D4C9D540  E3C8C540  D9E4D540  D7D9D6C3
15251209  C4D4C9C5  D74003C9  C2C3C4D9  68C8C4D9  C2E4C60D     C5E2E206  D940E3C1  C2D3C5E2  6840C5E7  C3C5D7E3  40E3C8C5
250109C2  D40604C9  D6C5D909  D6D93C25  0109C2D7  0504C5D6     40D3C1E2  E340C5D9  D9D6D940  C3D6C4C5  23252004  E6C8C9C3
C6C9D3C5  0E250109  D3C10704  F068C8C4  D9C2E4C6  19251609     C840C9D5  C4C9C3C1  E3C5E240  C140C8C1  D9C4E6C1  D9C540C5
C4D4E2C5  D34D03C9  C2E3E7E3  68D3C9C2  68C9D568  40F05D0E     D9D9D609  7A2C2529  08D5D5D5  D57EF0F0  F0F14040  D9E4D540
25010903  C10704F0  68C8C4D9  C2E4C61A  251709C4  D4E2C5D3     D3C9C2D9  C1D9E840  C3C1D5D5  D6E340C2  C540E2D7  C5C3C9C6
40D3C9C2  D6E4E363  D3C9C268  D6E4E368  40F05D11  250800E3     C9C5C426  252308D5  D5D5D57E  F0F0F8F0  40400106  C240C4C9
C5E7E3D3  D6D6D740  C5D8E400  035C1525  1209C404  C9D5D740     D9C5C3E3  D6D9E840  C9E240C9  D540C5D9  D9D6D92A  252708D5
D3C9C2E3  E7E36BE3  E7E3C2E4  C6002501  09C2D406  D4C9D6C5     D5D5D57E  F1F0F0F0  4040C4C9  E2D240C1  C4C4D9C5  E2E240C6
D9D9D609  0C250109  C207D504  C1C4C404  D6C4C263  C8C9D568     D6D94009  C5C1C440  C9E240E9  C5D9D6D5  00000000  00000000
40F05000  00000000  00000000  00000000                         00000000  00000000  00000000  00000000

PRIME FILE      BLOCK NO. 000003    BLOCK LENGTH  F8           SECOND FILE    BLOCK NO.  000003   BLOCK LENGTH F3

000003F8  00152512  D9C4D406  E4E34003  C9C2D6E4  E36BE3E7     000003F3  002A2527  08D5D5D5  D57EF2F0  F0F04040  D1D6C240
E3C2E4C6  00250009  C207D5E3  C5E7E3D3  D6D6D710  250500C1     C4C9D9C5  C3E3D6D9  E840C1C4  C4D9C5E2  E240C9E2  40E9C5D9
C4C4D4D6  C4D2D3C5  D8E40003  5C162513  D9C4D4E2  C5D34003     D6252528  08D5D5D5  D57EF4F0  F0F04040  C4C9E2D2  40C1C4C4
C9C2D6E4  E368D3C9  C263C1C4  C4002500  D9C2D705  D4C1C9D5     D9C5E2E2  40C6D6D9  40E6D9C9  E3C540C9  E240E9C5  D9D62A25
D3D6D6D7  10250500  C5D6C6C9  D3C5D2D3  C5D8E400  035C0C25     270805D5  D5D57EF8  F0F0F040  40D106C2  40C4C9D9  C5C3E3D6
D9D9C3D3  D6E2C540  5CC1D3D3  D5252209  C5D6D10F  24C9D6C5     D9E840C1  C4C4D9C5  E2E240C9  E240E9C5  D9D62F25  2C08D5D5
D9D9D6D9  4040C3C1  D5C3C5D3  0E250500  D3C9C2E3  E7E30303     D5D57EF0  F8F0F040  40C5E7C3  D740E3C5  D9D4C9D5  C1E3C5C4
C3C4C9C2  0E250500  D3C9C2C8  C4D90303  C3C4C9C2  0E250500     40E6C9E3  C840C961  D640C5E7  C3C5D7E3  C9D6D538  253804C6
D3C9C2D6  E4E30303  C3C4C9C2  10250500  D9C9C2E3  E7E30203     D6D940C1  D3D340D6  E3C8C5D9  40C5D9D9  D6D940C3  D6C4C5E2
C5D8E400  035C0E25  0809C5D5  E3D9E840  D9C9C2E3  E7E30025     40C5D5E3  C5D9407F  C8C5D307  40C5C3D5  D5D57E40  E6C8C5D9
0109C4C3  0604E77D  C1F0F0F0  70000000                         C540D5D5  D540C9E2  00000000  00000000

PRIME FILE      BLOCK NO. 000004    BLOCK LENGTH  EA           SECOND FILE    BLOCK NO.  000004   BLOCK LENGTH DC

000004EA  00112501  D9C4C30A  04E770F2  F2F0F2F0  F0F3C37D     000004DC  0020252A  D4E3C8C5  4003C1E2  E340E3C8  D9C5C540
0F250109  C4C30804  E770F8F8  F0F1F0F1  700F2501  09C4C308     C4C9C7C9  E3E240C6  D9D6D440  E3C8C540  D9F0F040  D4C5E2E2
04E770F0  F2F0F1F7  F7700025  0109C4C3  0604E77D  C1F1F0F0     C1C7C548  38A40000  00000000  00000000  0000D406  C4C4F240
7D102505  0009C9C2  C8C40902  03C508E4  00035C0E  250809C5     40400008  08163523  40404040  40404040  40404040  40404040
D5E3D9E8  40D9C9C2  C8C4090D  250109C4  C30604E7  7DC1F0F0     40404040  4040404D  40404040  40403F24  D9F0F140  40D106C2
F07D1125  0109C4C3  0A04E770  F2F2F0F2  F0F1F0F0  700F2501     404AD106  C2D5C1D4  C54F40D5  D6E340E2  C3C8C5C4  E4D3C5C4
09C4C306  04E770F8  F8F0F1F0  F1700F25  0109C4C3  0804E77D     40D640E3  C5D9D4C9  D5C1E3C5  C440C2E8  40D9E4D5  40D7D9D6
F0F2F0F1  F7F7700D  250109C4  C30604E7  70C1F1F0  F0701025     C3C5E2E2  D6D93125  2E04C1D5  40C5D9D9  D6D940E6  C1E240C5
050009C9  C2D6E4E3  0203C508  E400035C  0E250809  C5D5E309     D5C3D6E4  D5E3C5D9  C5C440E3  C8C1E340  D9C5D5C4  C5D9C5C4
E84009C9  C2D6E4E3  00250109  C4C30604  E770C1F0  F0F07D25     40E3C8C9  E240D106  C2C5D307  40C5C3D5  D5D57E40  E6C8C5D9
0109C4C3  0604E77D  C1F0F0F0  70000000                         C540D5D5  D540C9E2  00000000  00000000

PRIME FILE      BLOCK NO. 000005    BLOCK LENGTH  F4           SECOND FILE    BLOCK NO.  000005   BLOCK LENGTH F1

000005F4  00112501  09C4C30A  04E770F2  F2F0F2F0  F0F3C37D     000005F1  003C2539  04E4D5C5  E7C5C3E4  E3C1C2D3  C54B40D7
00250109  C4C30604  E770F0C2  F0F37011  250109C4  C30A04C1     D9C5E5C9  D6E4E2D3  E840D3C9  E2E3C5C4  40C5D9D9  D6D9E240
03F34003  C9C2E4C6  C65D1125  0109C4C3  0A04E770  F0F4F0F2     C3C1E4E2  C5C440E3  C5D9D4C9  D5C1E3C9  D6D54838  A4000000
```

Figure 3-2. Sample Librarian Map for a File Compare Operation (Part 2 of 3)

```
                                                                                                    PAGE # 0057
BLOCK   REC          NAME        TYPE    DATE      TIME     COMMENTS

                                                          02920010  030AEF10  2F0A1907  000A1C20  12002500  030001F0
                                                          0A854680  C1CCD253  C2BFCABF  5410CB66  5800C87E  92201002
                                                          92001003  0AEF102F  0A190700  0A1C8B12  00800300  0002140A
                                                          8547F0C0  4018000A  10070058  10C22647  F0C22A5C  C103030A
                                                          270A1A11  2C000007  09D5E309  40404000  00000000  03000000
                                                          00000000  00000000  00000000  00000000  00000000  03000011
                                                          2C0000E2  C3D9C6C9  03404000  00000000  00000000  03000000
                                                          00000000  00000000  00000000  00000000  0000000A  00040200
                                                          78080300  02C0A100  00177C01  BE0A8502

PRIME FILE           BLOCK NO. 0000BC     BLOCK LENGTH  85     SECOND FILE     BLOCK NO.  0000C8    BLOCK LENGTH 5A

                                                          00D0C8FA  000E1200  06000000  0298A000  880102A1  001F1200
                                                          14030000  02ACA000  69010070  03D00886  71010278  01008001
                                                          01A10000  1707C712  00360900  000ABDE7  C1F95C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C40  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  405C0000  02300000
                                                          025C0000  0288001F  AB001FAF  001F83D2

PRIME FILE           BLOCK NO. 0000BC     BLOCK LENGTH  85     SECOND FILE     BLOCK NO.  0000CC    BLOCK LENGTH 54

                                                          0000CC54  00221200  11090000  08740000  02A00000  02380000
                                                          00500000  02C00030  001F0000  1F04001F  0C0E1300  03030000
                                                          00000000  0000001F  0008A803  D6C1C4E2  40404014  A1000000
                                                          03000000  00000000  00C5D5C4  03C9C240  405C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C
                                                          5C5C5C5C  5C5C5C40  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  40404040  40404040
                                                          40404040  40404040  40404040  40404040  405C0000  02300000
                                                          025C0000  0288001F  AB001FAF  001F83D2

LIBRARIAN FINISHED
DATE  88/07/08  TIME 16.18
TOTAL NUMBER OF ERRORS 00000 UPSI SETTING X'00'
```

Figure 3-2. Sample Librarian Map for a File Compare Operation (Part 3 of 3)

3.3.10. Correcting Modules (COR and EOD)

Function

The librarian can correct records within any type of module by processing correction statements that define the corrections. The COR librarian control statement identifies the beginning of a series of correction statements for a module. It includes the name of the input file containing the module to be corrected, the module type, the module name, and the output file where the corrected module is to be placed. The EOD statement identifies the end of the series of correction statements.

Instructions for correcting program source or macro/jproc source modules are given in 3.3.11. Instructions for correcting object and load modules are given in 3.3.12.

Note: *When a module is corrected or updated, the header records for the new version will specify the date and time that the corrections were made. This date and time is normally obtained from the system information block. However, the / / PARAM UPDATE statement may be used in the librarian job control stream to override the system setting and specify another date and time to be used. See 2.6.4 for more information.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COR[.options]	[{input-lfn / SYSRUN}],{S / M / O / L},name[,output-lfn][,n[/n[/n]]] [,VERASG=n/n/n]

LABEL	ΔOPERATIONΔ	OPERAND
unused	EOD	unused

Options

E - The first EOD statement identifies the end of the correction statements.

N - Do not list module header information, correction statements, or any records added, deleted, or changed on the librarian map.

P - Punch a copy of the corrected module on cards.

X - Extend the load module if any of the supplied patch addresses are beyond the end of the module. This option is only valid when a single-phase load module is being corrected.

Note: There is no D option to list the corrected module. Instead, the COP librarian control statement described in 3.3.17 can be used following the EOD statement that indicates the end of the correction cards.

Positional Parameter 1

input-lfn
Specifies the logical file name of the file containing the module being corrected.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M,O,L,
Specifies the type of module being corrected as either program source (S), macro/jproc source (M), object (O), or load (L).

Positional Parameter 3

name
Specifies the name of the module being corrected.

Positional Parameter 4

output-lfn
Specifies the logical file name of the file where the corrected module is to be placed.

This parameter may be omitted if the input file is a disk or format label diskette file. If omitted, the old version of the module is deleted and the corrected version is added to the end of the input file. If the input file is a tape or data set label diskette file, this parameter is required and must be different than the input-lfn.

Positional Parameter 5

n/n/n
Specifies the version and subversion numbers, if included, of the library module being corrected. The value of each n is any decimal number from 0 to 256. When specified, a match of the version numbers is required. Otherwise, the correction process is terminated and a diagnostic message printed. If omitted, no checking takes place before the correction is made.

Each time a correction is made by the correction (COR) function, the third subversion number of the module is incremented by 1. This provides a history of how many times the module has been corrected.

Keyword Parameter

VERASG=n/n/n
: Specifies the version number to be applied to the module being corrected. (Should be specified as last parameter when used with positional parameters.)

Example 1

```
1         10    16
          COR   D1,S,OLDEXP1,D2
                .
                .   Correction cards
                .
          EOD
```

Corrects the source module named OLDEXP1 in file D1 as specified by the correction cards and adds the corrected version to file D2.

Example 2

```
1         10    16
          COR   ,S,OLDEXP2,D3
                .
                .   Correction cards
                .
          EOD
```

Corrects the source module named OLDEXP2 in the job run file YRUN as specified by the correction cards and adds the corrected version to file D3.

Example 3

```
1         10    16
          COR   D1,S,EXAMPLE3
                .
                .   Correction cards
                .
          EOD
```

Corrects the source module named EXAMPLE3 in file D1 as specified by the correction cards. Deletes the old version EXAMPLE3 in file D1 and adds the new version to the end of file D1.

3.3.11. Correcting Source Modules

The librarian can be used to make the following kinds of corrections to program source and macro/jproc source modules:

- Replace records
- Insert records
- Delete records
- Rearrange records
- Resequence the corrected module

The changes are specified by correction cards that are placed between the COR and EOD librarian control statements. These correction statements may include any combination of the following:

- SEQ librarian control statement to identify the sequence number field in the module being corrected and, optionally, to resequence the corrected module.
- New source records to insert into the module.
- New source records to replace existing records.
- SKI librarian control statements to skip records in the original module.
- REC librarian control statements to recycle the pointer to the beginning of the original module.

These correction statements are a series of instructions on how to correct the module. The librarian corrects a module not by modifying the original but by building a new module according to this series of instructions. To do this, the librarian reads the original module record by record. It determines when to make a change by comparing the sequence number of the current record to the sequence number in the next correction statement. If the current record is not affected by that statement, the librarian adds the current record to the end of the new module that it is building. The librarian then advances to the next record in the original module and compares its sequence number to the number in the correction statement. Thus, the librarian only processes the next correction statement when it reaches the appropriate position in the original module. With this method, it may be necessary to make several passes through the original module to rearrange records. This is illustrated in the REC example later in this section.

Because record sequence numbers are critical in correction operations, a source module must be sequenced before it can be corrected. It can be resequenced as it is corrected by using a SEQ statement immediately after the COR statement (see the SEQ librarian control statement, which follows, for instructions). When the correction statements are out of sequence, they are flagged and the module is not corrected.

After processing all the correction cards, the librarian deletes the original version of the module and places the corrected version in the output file specified in the COR statement.

Specifying the Sequence Control Field (SEQ)

Function

When the SEQ librarian control statement is used immediately after the COR statement, it can serve the following two functions:

1. To identify column positions used for sequence numbers in the source module being corrected.

2. To specify the starting sequence number and increment if the corrected module is to be resequenced.

The corrected module always uses the same column positions for the sequence numbers as the original module.

The SEQ statement is not necessary if the original module contains sequence numbers in the default column positions 73 through 80 and the corrected module is not to be resequenced. If the SEQ statement is omitted, any records inserted during the correction operation will not have sequence numbers.

Note: If any records in the original module do not have sequence numbers, the module must be resequenced before the corrections are processed. To do this, use a SEQ statement before the COR statement (see 3.3.7).

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	SEQ	[{lfn / YRUN},{S / M}[,name][,{column / 73}][,{content / SAME / 00000000}] [,{increment / 1}]

Options

Ignored

Positional Parameter 1

lfn
: Specifies the logical file name of the file containing the module being corrected.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M
: Specifies the type of module being corrected as being either program source (S) or macro/jproc source (M).

Positional Parameter 3

name
: Specifies the name of the module being corrected.

If used, it causes the module to be resequenced and must be the same as the module name used in the COR statement. If omitted, the SEQ statement only identifies the characteristics of the sequence numbers in the module.

Positional Parameter 4

column
: Specifies the starting column for the sequence number in each module record. If omitted, column 73 is assumed. The length of the content parameter determines the length of the sequence number field.

Positional Parameter 5

content
: Specifies the sequence number for the first record in the corrected module. This string can be from one to eight characters long. Its length determines the maximum length for any sequence number in the module. It may contain letters, digits, or both; if any letters are used, all the letters must be contiguous and precede any digits. For example, MA400, GRP000, and 10000001 are valid sequence numbers. However, M4A00, 600MAST, or 22AAA33 are not valid.

SAME
: If the module is being resequenced, SAME specifies that the eight-character sequence number for the first record in the corrected module is to be the same as the one in the original module.

If this parameter is omitted, the sequence number used for the first record is eight zeros (00000000).

Positional Parameter 6

increment
: Specifies a decimal number from 0 to 255, to be used as an increment for the sequence numbers of successive records.

If omitted, the value 1 is assumed.

Example 1

```
1         10    16
          COR   D1,S,EXP1
          SEQ   D1,S,EXP1,1,SRC100
                .
                .
                .
          EOD
```

The source module being corrected is EXP1 in file D1. Its sequence numbers are to begin in column 1 and are six characters long. The corrected module is to be resequenced with a sequence number of SRC100 for the first record and the default increment of 1.

Example 2

```
1         10    16
          COR   D2,S,EXP2
          SEQ   D2,S,EMP2,1,SAME,10
                .
                .
                .
          EOD
```

The source module being corrected is EXP2 in file D2. Its sequence numbers are to begin in column 1 and start with the same sequence number as the first record in the original module. The increment is to be 10.

Example 3

```
1         10    16
          COR   D3,S,EXP3
          SEQ   D3,S,EXP3,,,20
                .
                .
                .
          EOD
```

The source module being corrected is EXP3 in file D3. Its sequence numbers are to begin in the default column positions 73 through 80. The corrected module is to be resequenced in columns 73 through 80 starting with the default value of 00000000 for the first record and an increment of 20.

Example 4

```
1         10    16
          COR   D4,S,EXP4
          SEQ   D4,S,EXP4
                .
                .
                .
          EOD
```

The source module being corrected is EXP4 in file D4. Its sequence numbers are described by the default parameters. Resequence the corrected module using those same parameters.

Example 5

See Example 2 in "Replacing and Inserting Records," which follows in this section.

Replacing and Inserting Records

The correction statement for a new record or a replacement record contains the actual source module record itself. To cause a record to be replaced, the sequence number used on the correction statement must be in the same column positions and have the same value as the record being replaced. When a series of records is being inserted in the same place, only the first new record needs a sequence number. This sequence number should be within the range between the two existing records. Any additional records to be inserted at that point can be placed immediately after the first new record. They must be in the correct order.

The librarian map lists insertions and replacements as follows:

- For record replacements, the complete original record is listed followed by its replacement. The original record is preceded by five minus signs (-----) and the replacement record is preceded by five plus signs (+++++).
- For record insertions, the original record preceding the inserted records and all the inserted records are listed in order. Each inserted record is preceded by five plus signs (+++++).

If any new records are out of sequence within the correction deck, they are flagged and the module is not corrected.

Notes:

1. *If the /$ or /* job control statements are among the correction statements, there must be a /$ for each /* statement so that the /* is not interpreted as the end of the librarian control statements.*

2. *If the module being corrected is a librarian control stream, the correction statements may include EOD librarian control statements as new records to be inserted. Unless the E option is used in the COR statement, the librarian does not terminate the corrections until it encounters an EOD statement that does not have a corresponding COR or ELE statement.*

Example 1

```
1         10    16                                                  72
         COR   D0,S,COBOL1
           02  ACCT-NO          PIC 9(4).                          COB01650
           02  FILLER           PIC X(4).                          COB02100
           02  ACCT-NO-OUT      PIC 9(4).                          COB02650
            MOVE ACCT-NO TO ACCT-OUT.                             COB03750
            MOVE SPACES TO PRINTLINE.                             COB03820
            WRITE PRINTLINE.                                      COB03840
            MOVE SPACES TO PRINTLINE.                             COB04020
            WRITE PRINTLINE.                                      COB04040
            MOVE SPACES TO PRINTLINE.                             COB04420
            WRITE PRINTLINE.                                      COB04440
         EOD
```

These correction statements are new records or replacement records for the source module COBOL1 in file D0. Because the sequence numbers are in the default column positions 73 through 80, no SEQ statement is required after the COR statement.

Example 2

```
1         10    16
         COR   D1,S,RPGSAMP
         SEQ   D1,S,RPGSAMP,3,010
055IF                                        1   8 PHONIN
075C                    MOVE PHONIN     PHONOT  8
095OF                       PHONOT
         EOD
```

These correction statements insert three new records in the source module RPGSAMP, which is an RPG II program in the file D1. The sequence numbers 055, 075 and 095 used in the program are in columns 3 through 5. (IF, C, and OF are the beginning of the source statements that start immediately in column 6.)

Because columns 3 through 5 are not the default column positions used for sequence numbers, a SEQ statement is needed after the COR statement to identify their location. The fourth parameter in the SEQ statement specifies that the sequence numbers start in column 3. The fifth parameter, 010, is the sequence number used for the first record in the original module; its length indicates that the sequence numbers have a maximum length of three characters.

Skipping Records (SKI)

Function

The SKI librarian control statement causes the librarian to skip records in the original module as it processes corrections. The SKI statement specifies sequence numbers of the first and last records to be skipped. When it encounters the SKI statement, the librarian advances the pointer past this range of records without copying them to the corrected module.

The SKI and REC statements may be used together to rearrange records, as described in "Rearranging Records (REC)," which follows in this section.

Format

LABEL	ΔOPERATIONΔ	OPERAND	SEQUENCE
unused	SKI[.options]	last-sequence-no	[starting-sequence-no]

Options

D - List the records skipped on the librarian map.

Positional Parameter 1

`last-sequence-no`
Specifies the sequence number of the last record in the series to be skipped.

Sequence Field Value

`starting-sequence-no`
Specifies the sequence number of the first record to be skipped.

This value must be placed in the same column positions used for the sequence number field in the original module. These are the default column positions 73 through 80 or the column positions specified in the SEQ control statement used after the COR statement (see "Specifying the Sequence Control Field (SEQ)" earlier in this section).

If omitted, the skip operation begins immediately after the last record processed by the last COR statement.

Examples

```
      1         10    16                                          73
1.    |         SKI   AAA500                                      AAA500
2.    |         SKI   COB1400                                     COB1100
3.    |         SKI   LIBS2000
```

1. Skips record AAA500.

2. Skips all records between COB1100 and COB1400, inclusive.

3. Skips all records from the current record up to and including record LIBS2000.

Figure 3-3 illustrates a) the original source module, b) the corrected source module, and c) the librarian control statements used to make the corrections.

```
1         10      16                                                  72
TESTEXAM  EQU     *                                                   LINK0100
          CR      R0.W3                                               LINK0200
          BE      LK$3PA20                                            LINK0300
          BH      LK$3PA10                                            LINK0400
          M       W2.LK$CSGSZ                                         LINK0500
          L       W2.LK$CSEGT                                         LINK0600
          IC      W3.0(W2.W3)                                         LINK0700
          N       W3.LK$CX7F                                          LINK0800
          BNZ     LK$3PA00                                            LINK0900
          LA      W3.LK$CROOT                                         LINK1000
TESTEXA1  EQU     *                                                   LINK1100
          LR      R0,W3                                               LINK1200
          LA      RRTNOD.4                                            LINK1300
          B       LK$CPOP                                             LINK1400
          BAL     R14.LB$CSTK                                         LINK1500
```

a. Source module

```
     1         10      16                                             72
          COR     DO.S.TESTEXAM
1.        CR      R1.W2                                               LINK0200
2.        XR      W2.W2                                               LINK0450
3.        SKI     LINK0800                                            LINK0800
4.        SKI     LINK1400                                            LINK1100
          EOD
```

b. Correction deck

1. Replacement record for source record with sequence number LINK0200.
2. New source record to be inserted between the source records with sequence numbers LINK0400 and LINK0500.
3. Skip instruction which skips record LINK0800, causing it to be omitted from the corrected module.
4. Skip instruction for all records between LINK1100 and LINK1400, inclusive.

```
1         10      16                                                  72
TESTEXAM  EQU     *                                                   LINK0100
          CR      R1.W2                                               LINK0200
          BE      LK$3PA20                                            LINK0300
          BH      LK$3PA10                                            LINK0400
          XR      W2.W2                                               LINK0450
          M       W2.LK$CSGSZ                                         LINK0500
          L       W2.LK$CSEGT                                         LINK0600
          IC      W3.0(W2.W3)                                         LINK0700
          BNZ     LK$3PA00                                            LINK0900
          LA      W3.LK$CROOT                                         LINK1000
          BAL     R14.LB$CSTK                                         LINK1500
```

c. Corrected source module

Figure 3-3. Sample Source Module Correction

Rearranging Records (REC)

Function

The REC librarian control statement recycles the pointer during source module corrections. It resets the pointer to the first record in the original module. The REC statement can be used with the SKI statement to rearrange records in a program source or macro/jproc source module.

The sequence number field in the REC statement specifies when the pointer is to be repositioned. When the REC statement is processed, all records from the current record up to and including the record specified in the REC statement are copied to the new module. Then the pointer is reset to the first record. If the sequence number field in the REC statement is blank, the pointer is reset immediately.

When records are rearranged, they retain their same sequence numbers in the corrected module unless there is a SEQ statement after the COR statement to resequence the corrected module (see "Specifying the Sequence Control Field (SEQ)" earlier in this section).

Note: The REC statement cannot be used to correct modules on tape or data set label diskette.

Format

LABEL	ΔOPERATIONΔ	OPERAND	SEQUENCE
unused	REC	unused	[last-sequence-no]

Options

None

Sequence Field Parameter

`last-sequence-no`
: Is the sequence number of the last record to be copied to the new module before the pointer is reset to the beginning of the original module.

This value must be placed in the same column position used for sequence numbers in the original module.

If omitted, the pointer is reset immediately when the REC statement is processed.

Example

Figure 3-4 illustrates a) the original source module, b) the corrected source module, and c) the librarian control statements used to make the corrections.

```
1         10        16                                                  73
                    ORIGINAL RECORD                                     LIBS0900
                    ORIGINAL RECORD                                     LIBS0100
                    ORIGINAL RECORD                                     LIBS0200
                    ORIGINAL RECORD                                     LIBS0300
                    ORIGINAL RECORD                                     LIBS0800
                    ORIGINAL RECORD                                     LIBS0400
                    ORIGINAL RECORD                                     LIBS0600
                    ORIGINAL RECORD                                     LIBS0700
                    ORIGINAL RECORD                                     LIBS0500
                    ORIGINAL RECORD                                     LIBS1000
```

a. Original Source Module

```
                    ORIGINAL RECORD                                     LIBS0100
          NEW RECORD                                                    LIBS0200
                    ORIGINAL RECORD                                     LIBS0300
                    ORIGINAL RECORD                                     LIBS0400
          NEW RECORD                                                    LIBS0500
          NEW RECORD                                                    LIBS0550
                    ORIGINAL RECORD                                     LIBS0600
                    ORIGINAL RECORD                                     LIBS0700
                    ORIGINAL RECORD                                     LIBS0800
                    ORIGINAL RECORD                                     LIBS0900
                    ORIGINAL RECORD                                     LIBS1000
```

b. Corrected Source Module

```
          COR   D0,S,EXAMPLE1
1.        SKI   LIBS0900
2.  NEW RECORD                                                          LIBS0200
3.        SKI   LIBS0800                                                LIBS0800
4.        SKI   LIBS0700                                                LIBS0600
5.        REC                                                           .
6.        SKI   LIBS0400                                                LIBS0900
7.  NEW RECORD                                                          LIBS0500
8.  NEW RECORD                                                          LIBS0550
9.        REC                                                           LIBS0700
10.       SKI   LIBS0300                                                LIBS0900
11.       REC                                                           LIBS0800
12.       SKI   LIBS0500                                                LIBS0100
          EOD
```

c. Correction Statements

Figure 3-4. Example Using SKI and REC Statements to Rearrange Records (Part 1 of 2)

1. Skip original record LIBS0900. Copy original record LIBS0100.
2. Use new record LIBS0200 instead of original record LIBS0200. Copy original record LIBS0300.
3. Skip original record LIBS0800. Copy original record LIBS0400.
4. Skip original records LIBS0600 and LIBS0700.
5. Recycle the pointer to the beginning of the original module.
6. Skip original records LIBS0900 through LIBS0400, inclusive.
7. Use new record LIBS0500 instead of original record LIBS0500.
8. Insert new record LIBS0550.
9. Copy all original records from current record (LIBS0600) to LIBS0700. Recycle the pointer to the beginning of the original module.
10. Skip original records LIBS0900 through LIBS0300, inclusive.
11. Copy all original records from the current record (LIBS0800) to LIBS0800, inclusive. (Only one record is copied.) Recycle the pointer to the beginning of the original source module.
12. Copy all original records from the current record (LIBS0900) up to, but not including, LIBS0100. Skip all records from LIBS0100 through LIBS0500, inclusive. Copy all remaining records from the current record (LIBS1000) to the end of the original module. (Only the record LIBS1000 is copied.)

Figure 3-4. Example Using SKI and REC Statements to Rearrange Records (Part 2 of 2)

3.3.12. Correcting Object and Load Modules

Function

The instructions and data needed to correct, or patch, an object or load module are supplied to the librarian through correction cards. These correction cards must be placed between the COR and EOD librarian control statements. Each correction card may contain text data, relocation data, or an ORG directive that is used to set or reset the location counter.

Text patches are inserted in the module being corrected just after the last current text record and just ahead of the transfer record. Then, whenever the appropriate load module is loaded into main storage, or the object module is linked, the new text is inserted in the appropriate place in the module, overlaying the old text. When patched object modules are listed, the patches are indicated by asterisks.

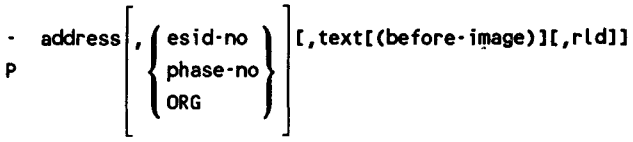
For phased load modules, the patches must be correctly sequenced by phase number.

Object module control sections and load module phase sizes may not be altered unless the module is a single-phase load module and the X option is used in the COR statement.

If the librarian detects an error within the correction cards, it does not make the correction.

Format

A correction card for an object or load module must have the following format:



Positional Parameter 1

{ - / P } address

This is the relative address to be assigned to the generated text record.

A hyphen in column 1 indicates that the address is relative to the object or load module base address. The letter P in column 1 indicates that the address is relative to the load module phase being patched.

The address value must be hexadecimal and may be positive or negative. A negative address is specified by a hyphen in column 2 followed immediately by the address. A positive address value must begin in column 2.

Positional Parameter 2

esid-no

This is the external symbol identification number (ESID) (01 through 255) for the object module being corrected. If omitted, the default value 01 is used.

phase-no

This is the phase number for the load module being corrected. It must be between 00 and 99. If omitted, the default value 00 is used.

ORG

This causes the location counter to be set or reset to the address specified by positional parameter 1. The current value of the location counter will automatically be added to all the addresses specified in subsequent correction cards. The location counter will remain set at that value until another ORG directive is used or the EOD librarian control statement is encountered. If used, the text and rld parameters must be omitted.

Positional Parameter 3

`text[(before-image)]`

Text is a contiguous string of hexadecimal digits to be inserted at the address represented by the sum of the most recent ORG directive, if any, and the relative address specified by positional parameter 1. The minimum amount of text that can be patched is 1 byte.

Text is required when an ESID or a phase number is specified. If omitted, the correction will be flagged and any data included for the rld parameter will not be used.

If ORG is specified in positional parameter 2, this parameter must be omitted.

The before-image subparameter allows you to verify proper field selection before applying the correction to that field. The verification process is a comparison of the hexadecimal string you provide via this subparameter and the code currently existing in the field being corrected. (It is not usually necessary to compare all of the text; a comparison of the first few bytes is sufficient to verify that the proper field has been selected.)

If the before-image string matches the existing code, the field is corrected with the new text provided. If the new text matches the existing code, the before-image string is ignored and no error is generated. However, if neither the before-image nor the new text matches, a diagnostic error is generated, the module is not corrected, and processing continues.

Positional Parameter 4

`rld`

Relocation data is optional. If used, it must be a string of 3-byte (6-hexadecimal-digit) substrings exactly as required. This parameter is valid for both object and load modules.

Notes:

1. *The values supplied for the address, ESID, phase number, and text parameters are automatically padded with zeros to the nearest half-byte, if necessary.*

2. *Each patch correction card generates a text record. Therefore, contiguous patch addresses on succeeding patch correction cards do not cause the text to be merged.*

Example 1

The following series of librarian control statements and correction cards are used to correct an object module named MYOBJ in file D2:

```
    1        10    16
1.  |        COR   D2,O,MYOBJ
2.  |    -C90,3,4880D074
3.  |    -1868,,0001250,011F00191F00
4.  |    -2946,255,F0F1F2
5.  |        EOD
```

1. Identifies the module being corrected as the object module named MYOBJ in file D2.

2. Patches the specified load half-word instruction into hex address 0C90. The text record CSECT base ESID is 3. No relocation data is specified.

3. Patches the specified full-word address constant into hex address 1868. The text CSECT base ESID is 1, the default. Two relocation masks are included, each with full modification to the address constant, once with ESID#1 value and once with ESID#19 value. (For RLD formats, see Figure B-2.)

4. Patches the three EBCDIC characters 0, 1, and 2 at hex address 2946. The base ESID is 255.

5. Indicates the end of the patch cards.

Example 2

The following series of librarian control statements and correction cards are used to correct a multiphase load module named MYLOAD. The original version resides in file D0; the patched version is to be placed in file D2.

```
    1        10    16
1.  |        COR   D0,L,MYLOAD,D2
2.  |   -C90,,4880D074
3.  |   P12D,1,9540C012
4.  |   -0124,ORG
5.  |   P250,3,AB
6.  |   --4B78,ORG
7.  |   -5672,4,0A1C
8.  |   -0,ORG
9.  |   -D2E,6,00012E,061F00
10. |        EOD
```

1. Identifies the module being corrected as load module MYLOAD in file D0. The corrected version of the module is to be added to file D2.
2. Applies the specified text to load module address C90 of phase 0.
3. Applies the specified text to address 12D relative to the start of phase 1.
4. Sets the location counter to 0124.
5. Applies the specified text to address 250+124 relative to the beginning of phase 3.
6. Resets the location counter to -4B78.
7. Applies the specified text to load module address 5672-4B78 in phase 4.
8. Resets the location counter to 0.
9. Applies the specified text to relative address D2E in phase 6.
10. Identifies the end of the correction cards.

3.3.13. Blocking Load Modules (BLK)

Function

The BLK librarian control statement converts a standard load module to a blocked load module. The blocked version may then be placed in a new file or may replace the unblocked version in the same file.

Blocked load modules can be loaded more efficiently than standard, unblocked load modules. For a standard load module, each phase is loaded 256 bytes at a time. For a blocked load module, the loader can read each phase (or up to an entire track) in a single I/O operation. This requires fewer disk accesses to load the entire module.

Files containing blocked load modules use three partitions, rather than two like all other SAT files. The first partition is still used for a directory. The load module records in the second partition are data records that define the boundaries of each phase in the third partition. The third partition is unstructured and contains contiguous text data, free of any control information. This text data is in sequential load order and is binary zero-filled when appropriate.

In standard load format, no text records can be overlaid; however, in blocked format, they can. For example, this overlaying would occur when the load module detects the following coding:

Location	Operation	Operand
0000	CLI	R6,X'01'
0004	BC	8,STOR1
0004	ORG	*-4
0004	BC	15,STOR2

The BC 15 overlays the BC 8. In standard load module format, the bytes of text for the BC 8,R1 continue to exist in the module although they are overlaid at load time.

The following points should be considered in determining whether to convert a standard load module to blocked format:

- Modules less than 4K bytes in length take longer to load if blocked, unless the resident loader named RESMOD.SM$LOD is used.
- Do not block assembler language program modules that have information passed from one phase to the next in a DS area. All DS areas are zero-filled.
- Do not block assembler language program modules that have address constants overlaid with text. This can occur when an unneeded address constant is used for patch space. It can also occur when an ORG directive is used to overlay an address constant with instructions.
- Patches slow down the loading of blocked load module phases more than standard load module phases.

Notes:

1. *A blocked load module can be copied to and from a tape, using a COP control statement (see 3.3.5) and can be corrected to and from tape.*
2. *If the blocked load module is output to a file that already contains a load module with that name, the old one is deleted and the new one is added to the file.*
3. *Load modules generated from ANSI 1974 COBOL source code that include the dynamic CALL or CANCEL verbs cannot be converted to blocked format.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	BLK	input-lfn,name[,output-lfn][,VERSAG=n/n/n]

Options

None

Positional Parameter 1

input-lfn
: Specifies the logical file name of the file containing the standard load module.

Positional Parameter 2

name
: Specifies the name of the standard load module.

Positional Parameter 3

output-lfn
: Specifies the logical file name of a disk or format label diskette file in which the blocked load module is to be saved.

If omitted, the blocked load module is added to the end of the input file and the standard load module is logically deleted. This parameter is required if the original module is on a tape or format label diskette.

Keyword Parameter

VERASG=n/n/n
: Specifies the version number to be applied to the blocked load module. (Should be specified as the last parameter when used with positional parameters.)

Examples

```
    1       10    16
    |       |     |
1.  |       BLK     D1,STEP2,D2
2.  |       BLK     D3,STEP3
```

1. Converts the standard load module named STEP2 in file D1 to a blocked load module and adds the blocked version to the end of file D2. File D1 retains the standard load module.

2. Converts the standard load module named STEP3 in file D3 to a blocked load module. The standard version is deleted and the blocked version is added to file D3.

3.3.14. Renaming Modules, Groups, and Records (REN)

Function

The REN librarian control statement is used to:

- Rename a single module or module group
- Rename all modules with the same name
- Rename a record in an object or load module
- Mark an object or load module as shareable or unshareable
- Add or change the comment in a module header record

When a module name is changed, any other module in the file with the new name and the same type is deleted. When a load module name is changed, the new name is reflected throughout each phase.

Note: The REN control statement cannot be used for files on tape or data set label diskettes.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	REN[.options]	[{lfn / YRUN}][,{S / M / O / L}] ,old-name[.{record-type-and-name / RON / ROFF}] [,new-name][,comment]

Options

G - Rename the next group with the group name specified by old-name.

N - Do not list module header records on the librarian map.

Positional Parameter 1

```
lfn                    !
```

Specifies the logical file name of the file that contains the modules to be processed.

If omitted, the job run library YRUN is assumed.

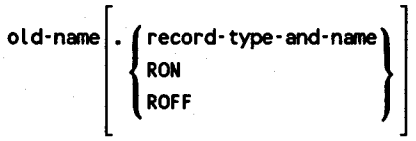
Positional Parameter 2

```
S,M,O,L
```

Specifies the type of modules to be processed as either program source (S), macro/jproc source (M), object (O), or load (L).

If omitted, all modules with the name specified by old-name are processed. The type parameter is not valid when the G option is used.

Positional Parameter 3



Old-name identifies the module or module group (with G option) to be processed. If record-type-and-name, RON, or ROFF is used, a period is needed as a separator.

The record-type-and-name is required if an object module record name or an alias phase name of a load module is being changed. This value consists of a letter for the record type plus the old record name combined in a single string. The valid record types and their meanings are as follows:

C - COM record

E - ENTRY record

N - procedure name record

P - alias-phasename of a load module

S - CSECT record

V - V-CON record

X - EXTRN record

For example, the value MASTER.XTAG5 for this parameter specifies that the EXTRN record named TAG5 in the module MASTER is to be renamed.

RON or ROFF is required after the old name if the shareability status of an object module is being changed. RON specifies shareable and ROFF specifies unshareable.

Positional Parameter 4

new-name
: Specifies the new name for the module, module group (with G option), or record. It can be up to eight characters long, except for a multiphase load module name, in which only the first six characters of the name can be changed.

This value must be omitted if the shareability status of a module is being changed or only the comment in a module header record is being added or changed.

Positional Parameter 5

comment
: This is a string of up to 30 characters to be included in the header record of the specified module or group.

If omitted, the current comment remains the same.

Examples

```
     1        10    16
1.   |        REN   D1,S,EXP1,NEWEXP1
2.   |        REN.N D2,,EXP2,NEWEXP2
3.   |        REN   ,O,EXP3,NEWEXP3
4.   |        REN   D4,O,EXP4.SCS0041,CS0042
5.   |        REN.G D5,,EXP5,NEWEXP5
6.   |        REN   D6,S,EXP6,,REVISED BY EJM
7.   |        REN   D7,L,EXP7.RON
```

1. Renames the source module EXP1 in file D1 to NEWEXP1.

2. Renames all modules of any type named EXP2 in file D2 to NEWEXP2. The header information for these modules is not printed on the librarian map.

3. Renames the object module named EXP3 in the job run library YRUN to NEWEXP3.

4. Renames the CSECT record named CS0041 in the object module named EXP4 that resides in file D4. The new name is CS0042.

5. Renames the next group named EXP5 in file D5 to NEWEXP5.

6. Adds the comment REVISED BY EJM to the header record of the source module EXP6 in file D6.

7. Makes the load module named EXP7 in file D7 shareable.

3.3.15. Compressing a File (PAC)

Function

The PAC librarian control statement physically erases modules that have been deleted and moves the remaining modules toward the beginning of the file. As a result, all unused file space is at the end of the file. The PAC statement can be used only for disk or format label diskette files.

Modules are deleted by the DEL librarian control statement or by any statement that causes an existing module to be replaced or updated. Deleted modules still occupy space in the file, but are identified as being nullified in the file directory.

Because the PAC operation permanently removes a module, it may be desirable to only use it occasionally. A version of the COP statement is available to copy all but the nullified modules from one file to another (see 3.3.5).

On the librarian map, the PAC statement is followed by the header information for all of the modules in the file. Those that were not packed are listed under the heading MODULES NOT MOVED. Any modules that were packed are listed under the heading MODULES MOVED.

Notes:

1. *The PAC statement can be used only for disk or format label diskette files. To achieve the equivalent result in a tape file or a data set label diskette file, use a COP statement to copy all but the deleted modules from the original tape to a new one (see 3.3.5).*

2. *The file being packed should be lockable, giving the current user exclusive access. For a description of the file lock facility, see the* Data Base Management System (DMS) System Support Functions Programming Guide *(UP-10870).*

3. *A file being packed cannot be updated.*

4. *When a load file is being packed, the pack operation must be completed before a program is executed from the file.*

5. *The PAC control statement should not be used for a file that contains an active ICAM symbiont.*

6. *The PACKRES canned job control stream is available to compress files on a system release volume. See 3.5.4 for information.*

7. *If the YLOD library is packed, the fast-load capability is disabled, reducing system efficiency. You must reinitialize (re-IPL) the system to regain the fast-load capability.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	PAC[.options]	[{lfn / YRUN}]

Options

N - Do not list the module header information on the librarian map.

Positional Parameter 1

lfn
Specifies the logical file name of the file that is being compressed.

If omitted, the job run file YRUN is assumed.

Examples

```
      1       10    16
      |       |     |
1.    PAC   D1
2.    PAC.N D2
```

1. Erases all logically deleted modules in file D1.

2. Erases all logically deleted modules in file D2. Module header information is not printed on the librarian map.

3.3.16. Printing a File Table of Contents

Printing an Alphabetical Table of Contents (LST)

Function

The LST librarian control statement prints an alphabetical table of contents of all modules in a file or only modules of a specified type. The list includes the module type, name, creation date, and creation time.

Note: The LISTRES and DRDP canned job control streams are available in the system to print directories of files on a system release volume. See 3.5.1, 3.5.2, and Appendix A for more information.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	LST	[{lfn / YRUN}][,{S / M / O / L}]

Options

None

Positional Parameter 1

lfn
: Specifies the logical file name of the disk or diskette file to be processed.

If omitted, the job run file YRUN is assumed.

Positional Parameter 2

S,M,O,L
: Specifies the module type to be processed as either program source (S), macro/jproc source (M), object (O), or load (L).

If omitted, all modules in the file are listed.

Examples

```
    1       10    16
1.          LST   D1,L
2.          LST   D2
3.          LST   ,S
4.          LST
```

1. Prints an alphabetical list of all load modules in file D1.
2. Prints an alphabetical list of all modules in file D2.
3. Prints an alphabetical list of all source modules in the job run file YRUN.
4. Prints an alphabetical list of all modules in the job run file YRUN.

Printing a Sequential Table of Contents (COP)

Function

A COP librarian control statement with only an input file parameter prints a sequential table of contents for a file. The list includes the type, name, creation date, and creation time for all modules in order as they occur in the file. The D option may be used to list all entries in the file directory (see 2.3.1 for a description of a disk file directory).

The table of contents begins on a new page on the librarian map.

Notes:

1. *Other uses of the COP statement are explained in the sections indicated:*
   - *Resetting the pointer in a file (3.3.2)*
   - *Copying modules and files (3.3.5)*
   - *Listing and punching modules (3.3.17)*
2. *The LISTRES and DRDP canned job control streams are available in the system to print directories of files on a system release volume. See 3.5.1, 3.5.2, and Appendix A for more information.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COP[.options]	[{input-lfn / YRUN}]

Options

D - Print all records in the file directory.

If omitted, header information for all modules in the file are listed.

Positional Parameter 1

`input-lfn`
Specifies the logical file name of the file to be processed.

If omitted, the job run file YRUN is assumed.

Examples

```
     1        10    16
     |        |     |
1.   |        COP   D1
2.   |        COP.D D2
3.   |        COP
```

1. Prints a sequential list of all modules in the file D1.
2. Prints a sequential list of all directory entries in the file D2.
3. Prints a sequential list of all modules in the job run file YRUN.

3.3.17. Listing and Punching Modules

How Modules Are Listed and Punched

Module listings are printed on the librarian map.

Source modules are listed one record per line or punched one record per card, exactly as they are coded. Each record up to 80 characters in length (including the sequence number) requires one card or one printed line. Records with more than 80 characters require more than one card.

Program source and macro/jproc source modules will be printed in hex format if the statement // PARAM PRTOBJ=ON is used between the // EXEC LIBS and /$ job control statements. This is effective for the entire librarian job step.

For object or load modules, the librarian lists or punches 80-byte records containing a 5-digit sequence number in columns 1 through 5 and object or load code in hexadecimal format in the remaining 75 columns. To do this, the librarian divides the code into 75-byte intervals. When the module is copied back to a disk, tape, or diskette file, the module is reassembled as it originally existed.

When the librarian punches a module, it does not punch the module header record.. Instead, it punches an ELE card before the first record in the module and an EOD • card after the last record in the module. The logical file name D1 is always used on the ELE card along with the name and type of the module being punched. For example, the following ELE and EOD cards would be punched for a source module named SAMPLE:

```
{ELE}
{EOD}  D1,S,SAMPLE
```

Note: *A source module punched on cards should be sequenced so that the librarian can automatically check the order of the records as it reads them. A module can be sequenced before or after it is punched using the SEQ librarian control statement described in 3.3.7.*

List and Punch Options as Secondary Operations (D and P Options)

This section explains how to list or punch modules without performing any other operation. However, all modules processed by the following librarian control statements can be listed or punched at the same time by including a list option (D) or a punch option (P) in the statement. Refer to the section indicated for more information:

COP	3.3.5
COR (punch only)	3.3.10
DEL	3.3.6
ELE	3.3.4
REPRO	3.3.19
SEQ	3.3.7

Note: *The MODLST canned job control stream is available in the system to print listings of modules in a system release volume. See 3.5.3 for information.*

Listing or Punching Modules as the Only Operation (COP)

Function

This COP librarian control statement (with no output file parameter) is used to list or punch modules in a file. No modules are copied, as when an output file parameter is included in the COP statement (see 3.3.5).

The list option (D), the punch option (P), or both may be specified in the same COP statement. Other options and parameters used in the COP statement specify the modules to be listed or punched.

Note: *The MODLST canned job control stream is available in the system to print listings of modules in a system release volume. See 3.5.3 for information.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COP[.options]	[{input-lfn / YRUN}][,{S / M / O / L}][,name]

Options

A - Process all groups in the file with the group name specified by the name parameter. The G option is also required.

C - Without the G option, the name specified in the name parameter is a module name prefix. With the G option, the name is a group name prefix.

D - Prints listings of all the modules specified.

G - Without the C option, the name parameter specifies a group name. Process all modules in the first group with that group name. If the A option is also used, process all modules in all groups with that group name.

With the C option, the name parameter specifies a group name prefix. Process all modules in all groups with that group name prefix from the pointer position to the end of the file.

N - Do not list module header information on the librarian map.

P - Punch all the modules specified.

U - Without the G option, process the modules from the pointer position in the file up to and including the specified module; both the name and type parameters are needed to specify the module. With the G option, process all modules up to and including the first module group with the specified name.

Note: *If the C and U options are used together, an error occurs.*

Positional Parameter 1

`input-lfn`
Specifies the logical file name of the file to be processed.

If omitted, the job run file YRUN is used. To list or punch all modules in a file, this parameter must be followed by a comma and all other parameters must be omitted.

Positional Parameter 2

`S,M,O,L`
Specifies the type of modules to be processed as either program source (S), macro/jproc source (M), object (O), or load (L).

If omitted, all modules with the specified name are processed. The type parameter is not valid when the G option is used.

Positional Parameter 3

name

Specifies the name of a particular module, module group (with G option), module name prefix (with C option), or group name prefix (with C and G options). It may contain up to eight characters.

If omitted, all modules of the specified type from the pointer position to the end of the file are processed. If both the name and type parameters are omitted, all modules after the pointer position are processed.

Examples

```
      1        10    16
1.    COP.P D1,O,EXAMPLE1
2.    COP.D D2,S
3.    COP.DPU D3,S,EXAMPLE3
4.    COP.CD D4,,COB
5.    COP.GP D5,,EXAMPLE5
6.    COP.D  D6,
```

1. Punches the object module named EXAMPLE1 in file D1.
2. Prints listings of all source modules from the pointer position to the end of file D2.
3. Punches and prints listings of all modules from the pointer position in file D3 up to and including the source module named EXAMPLE3.
4. Prints listings of all modules with the name prefix COB from the pointer position to the end of file D4.
5. Punches all modules in the next group named EXAMPLE5 in file D5.
6. Prints listings of all modules from the pointer position to the end of file D6. Note that the trailing comma is required.

3.3.18. Working with Module Groups

A module group is a consecutive series of modules in a file that have been placed between a beginning-of-group and an end-of-group record. These modules can be processed collectively using librarian control statements with the group option (G). Instructions for creating a module group are given in "Building Module Groups (BOG and EOG)," which follows in this section. Instructions for using the G option in librarian control statements are given in "Module Group Operations (G Option)," which also follows in this section.

Building Module Groups (BOG and EOG)

Function

The following steps are required to create a module group in a file:

1. Write the beginning-of-group record to the file (BOG control statement).
2. Add selected modules to the file.
3. Write an end-of-group record to the file (EOG control statement).

The BOG control statement must be used first to name the group and write a beginning-of-group record in the file. The group name may contain up to eight EBCDIC characters and need not be unique. This name will be independent of any module name in the file and will be used only to reference the entire group in subsequent group operations.

After the BOG statement, other librarian control statements can be used to obtain modules from other files and add them to the file after the beginning-of-group record. The output file for these statements must be the same one used in the BOG statement for the group.

After all the modules have been added to the group, an EOG control statement is needed to write an end-of-group record in the file. This completes the group creation procedure.

The following rules apply to the creation of module groups:

- A module group is always added at the end of the output file.
- A single file can contain any number of groups.
- A group can contain any number of modules of one or assorted types.
- Groups may be nested (see B.2.1).
- A disk file cannot contain two modules with the same name and type combination, even if they are in different groups.

Note: *After a group is created, the beginning-of-group and end-of-group records only affect librarian processing when the G option is used. Thus, a librarian control statement that processes all modules of a certain type will apply to all modules of that type whether they are contained within a group or not.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	BOG	group-name [, { lfn / YRUN }]

LABEL	ΔOPERATIONΔ	OPERAND
unused	EOG	group-name [, { lfn / YRUN }]

Options

None

Positional Parameter 1

group-name

Specifies the name of the module group being created. It may contain up to 8 characters. This name need not be unique.

Positional Parameter 2

lfn

Specifies the logical file name of the file in which the group is being created.

If omitted, the job run file YRUN is used.

Example 1

```
1       10    16
        BOG   GROUP1,D1
        COP   D0,,,D1
        EOG   GROUP1,D1
```

The BOG statement writes a group header record for the group named GROUP1 in file D1. The COP statement between the BOG and EOG statement copies all modules in file D0 to file D1 following the beginning-of-group record. The EOG statement writes an end-of-group record in file D1 after the last module is copied.

Example 2

```
1       10    16
        BOG   GROUP2
        ELE   ,S,SRCMOD
         .
         .
         .
        EOD
        EOG   GROUP2
```

The BOG statement writes a group header record for the group named GROUP1 in the job run file YRUN. A source module on cards is placed between the ELE and EOD librarian control statements. This module is copied to YRUN following the beginning-of-group record. The EOG statement writes an end-of-group record in YRUN after the module is copied.

Example 3

See 3.6.3 for a sample librarian job control stream which builds groups.

Module Group Operations (G Option)

Once a module group has been created, the modules in it can be processed collectively by any of the librarian control statements which allow the group (G) option. Here is a list of those statements and a escription of how they process groups:

- COP

  A COP statement with an input file parameter, a name parameter, and an output file parameter copies module groups from the input file to the output file.

  A COP statement with an input file parameter and a group name but no output file parameter lists (D option) or punches (P option) module groups.

  A COP statement with an input file parameter, a group name, and only the G option resets the pointer past the next group with that group name.

- DEL deletes all module groups.

- RES resets the pointer to the next group with the specified group name.

- REN renames groups with the specified group name.

The following syntax rules apply to these statements for group operations:

- The statement must include the group (G) option.
- The statement must include the name parameter.
- The statement must not include a type parameter.

Without the C or A option, the statement applies only to the next group in the input file with the group name specified by the name parameter. If the A option is used, all groups with that name are processed. If the C option is used, the name parameter specifies a group name prefix and the statement applies to all groups with the group name prefix from the pointer position to the end of the file.

If the G and U options are used together, all modules from the pointer position up to and including the group specified by the name parameter are processed.

The C and U options cannot be used together.

Note: *When the G option is not used, the beginning-of-group and end-of-group records do not affect librarian processing. Thus, a librarian control statement without a G option which applies to all modules with a certain name, for example, will affect all modules in the file with that name regardless of whether they are in a group or not.*

Examples

```
    1       10    16
    |       |     |
1.  COP.G D1,,GROUP1,D2
2.  COP.GD D3,,GROUP2
3.  COP.GCD D4,,SRC,D5
4.  COP.GA D6,,GROUP4,D7
5.  DEL.G D8,,GROUP5
6.  DEL.GA D9,,GROUP6
7.  DEL.GC D10,,COB6
8.  REN.G D11,,GROUPA,GROUPB,MERGED 9/9/88
9.  RES.G D12,,GROUP8
```

1. Copies the next module group named GROUP1 from file D1 to D2.
2. Prints listings of all modules in the next group named GROUP2 in file D3.
3. Finds all module groups with the group name prefix SRC from the pointer to the end-of-file D4 and copies them to file D5. Also prints listings of all the modules copied.
4. Finds all groups with the group name GROUP4 from the pointer to the end-of-file D6 and copies them to the end-of-file D7.

5. Deletes the next module group named GROUP5 in file D8.

6. Deletes all groups named GROUP6 following the pointer in file D9.

7. Deletes all groups with the group name prefix COB6 following the pointer in file D10.

8. Changes the name of the next module group named GROUPA in file D11 to GROUPB. Also changes the comment in the group header record to MERGED 9/9/88.

9. Resets the pointer in file D12 to the next module group named GROUP8.

3.3.19. Inserting Linkage Editor Control Statements in Object Modules (REPRO and EOD)

Function

The REPRO librarian control statement inserts or deletes embedded control statements in object modules.

To perform insertions only, the librarian needs the following sequence of statements:

```
REPRO lfn,module-name
.
.                 } Any linkage editor control statements
.                   to be inserted after the header record
EOD
.
.                 } Any linkage editor control statements
.                   to be inserted after the transfer record
EOD
```

The REPRO librarian control statement identifies the file containing the object module to be modified and the name of the object module itself. This is followed by any linkage editor control statements to be inserted after the object module header record. The EOD librarian control statement indicates the end of these. Following the EOD statement are any linkage editor control statements to be inserted after the object module transfer record. These are followed by another EOD librarian control statement. Both EOD statements are always required with the REPRO statement, even if linkage editor control statements are not being added in both places.

If the object module already contains a set of linkage editor control statements, the new ones are placed after the existing ones in the same order that they are supplied.

Any deletions required can be specified at the same time as insertions. This is done through the REPRO control statement, as follows:

```
REPRO lfn,module-name,#deletions,#deletions.
.
.                 } Any linkage editor control statements
.                   to be inserted after the header record
EOD
.
.                 } Any linkage editor control statements
.                   to be inserted after the transfer record
EOD
```

In the REPRO control statement, the first *#deletions* specifies the number of records to be deleted after the object module header record; the second *#deletions* specifies the number of records to be deleted after the object module transfer record. Two EOD statements are still required following the REPRO statement even if no linkage editor control statements are being inserted in the module at the same time.

The librarian always deletes the specified number of records from the end the existing records in that part of the module. For example, the following statements delete the last three linkage editor control statements after the header record and the last two linkage editor control statements after the transfer record:

```
REPRO D1,MODA,3,2
EOD
EOD
```

Deletions are always performed before insertions. Therefore, selected records can be deleted from between other records, by deleting them all and then adding some of them back again. This should be done with one REPRO statement, two EOD statements, and the linkage editor control statements being retained.

The librarian makes the corrections by building a new version of the object module. After making all the corrections, the librarian deletes the original version and adds the new version at the end of the file.

Note: *The REPRO librarian control statement cannot be used for an object module on tape or data set label diskette.*

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	REPRO[.options]	[{lfn / YRUN}] name[,#deletions][,#deletions]

Options

D - Print a listing of the entire modified object module on the librarian map.

N - Do not list module header information on the librarian map.

P - Punch the entire module as modified.

Positional Parameter 1

lfn

Specifies the logical file name of the disk file containing the object module to be modified.

If omitted, the job run file YRUN is used.

Positional Parameter 2

name

Specifies the name of the object module to be modified.

Positional Parameter 3

#deletions

This integer specifies the number of control statement records to be deleted from the end of the set of control statements which currently exist after the module header record.

Positional Parameter 4

#deletions

This integer specifies the number of control statement records to be deleted from the end of the set of control statements which currently exist after the module transfer record.

Example 1

Modify the object module named EXAMPLE in file D1 as follows: Add the control statement records INCLUDE A and INCLUDE B at the end of any control statements following the module header record. List and punch the modified object module.

```
1        10    16
|
|       REPRO.DP D1,EXAMPLE
|       INCLUDE A
|       INCLUDE B
|       EOD
|       EOD
```

Example 2

Modify the object module named EXAMPLE in file D1 as follows: Add the linkage editor control statements INCLUDE A and INCLUDE B at the end of any linkage editor control statements following the module header record. Add the linkage editor control statement INCLUDE C at the end of any linkage editor control statements following the transfer record.

```
1        10    16
         REPRO D1,EXAMPLE
         INCLUDE A
         INCLUDE B
         EOD
         INCLUDE C
         EOD
```

Example 3

Modify the object module named EXAMPLE in file D1 as follows: Add the linkage editor control statement LOADm X at the end of any linkage editor control statements following the module tramsfer record. List the modified object module.

```
1        10    16
         REPRO.D D1,EXAMPLE
         EOD
         LOADm  X
         EOD
```

Example 4

Modify the object module named EXAMPLE in file D1 as follows: Delete the last linkage editor control statement currently following the object module header record and then add the source record INCLUDE A after the header record. Also, delete the last three linkage editor control statements currently following the object module transfer record and then add the source record RES X'1000' after the transfer record. List the modified object module.

```
1        10    16
         REPRO.D D1,EXAMPLE,1,3
         INCLUDE A
         EOD
         RES  X'1000'
         EOD
```

Example 5

Modify the object module named EXAMPLE in file D1 as follows: Add the linkage editor control statement INCLUDE A at the end of any linkage editor control statements currently following the module header record. Delete the last three linkage editor control statements currently following the object module transfer record and then add the linkage editor control statement record INCLUDE B after the transfer record. List and punch the modified object module.

```
1        10    16
┌──────────────────────────────────────────────
        REPRO.DP D1,EXAMPLE,,3
        INCLUDE A
        EOD
        INCLUDE B
        EOD
```

3.3.20. Saving Librarian Control Statements on Disk or Diskette

A series of librarian control statements can be saved on disk or format label diskette and then accessed when needed by a librarian job. This eliminates the need for the user to keep rewriting frequently used librarian routines. Modules containing just librarian control statements are considered source modules. Instructions for creating and saving these modules are given in "Writing and Saving the Control Statements," following in this section. Once the control statements are saved, they can be accessed by other librarian jobs using the ESC statement described in "Using Saved Librarian Control Statements," also following in this section.

Note: This procedure is for saving only librarian control statements. The example in 3.6.4 explains how to save a complete librarian job control stream.

Writing and Saving the Control Statements

One way to create and save a series of librarian control statements on disk is to use the general editor (EDT) at a workstation. For instructions on using the general editor, see the *General Editor Operating Guide* (UP-9976). For example, the following librarian control statements can be written in the editor work space as shown (the numbers on the left are work space line numbers):

```
1.0000          COP    D1,S,COBOL4,D2
2.0000          LST    D2
3.0000@WRITE MO=LIBTEST,TYPE=S,FIL=SRCFIL
```

Lines 1 and 2 are the librarian control statements being saved. The general editor @WRITE command (line 3) saves the statements in the module named LIBTEST in the diskette file named SRCFIL. The module type must be S for source.

The series of librarian control statements can also be punched on cards and copied to diskette using the librarian. The following is a sample librarian job control stream to do this:

```
// JOB SAVE
// DVC 20  // LFD PRNTR
// DVC 130  // VOL PUBRES
// LBL SRCFIL  // LFD SRCFIL
// EXEC LIBS
/$
          FIL   D1=SRCFIL
          ELE   D1,S,LIBTEST
          COP   D1,S,COBOL4,D2   } Librarian control statements being saved
          LST   D2               }
          EOD
/*
/&

// FIN
```

In this example, the file SRCFIL has already been allocated on the format label diskette volume PUBRES. The ELE statement identifies the diskette file D1, or SRCFIL, where the librarian control statements are to be saved. It also creates and names the module LIBTEST to contain them. The module type must be S, for source. The EOD librarian control statement is associated with the ELE statement to identify the end of the statements being saved.

Using Saved Librarian Control Statements (ESC)

Function

After a series of librarian control statements have been saved on disk or diskette, they can be called by other librarian jobs through the use of the ESC librarian control statement. This statement accesses the source module containing the control statements and inserts them in the librarian job control stream where they are to be used. The following rules apply:

- The file containing the source module must be declared in a device assignment set, just like any other disk or format label diskette file.
- The ESC statement may be one of any number of statements used in a librarian job.
- A librarian job may contain more than one ESC statement.

- For each ESC statement used in a librarian job, the librarian requires 7400 (hexadecimal) bytes of main storage space to run. This must be specified in the // JOB control statement, as follows:

```
// JOB MYJOB,,,7400
```

- The ESC statement itself must be placed between the /$ and the /* job control statements.

All statements obtained from the a source module via an ESC statement are listed on the librarian map with *ESC* in the control field.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	ESC	filename,LD,modulename

Options

None

Positional Parameter 1

`filename`

Specifies the name of the file containing the source module. This must match the name used in the // LFD job control statement for that file.

Positional Parameter 2

`LD`

Indicates the control stream is in a librarian source module.

Positional Parameter 3

`modulename`

Specifies the name of the librarian source module to be processed.

Example:

The following librarian control statements have already been saved in a source module named LIBTEST in the file named SRCFIL on the diskette volume PUBRES:

```
COP   D1,S,COBOL4,D2
LST   D2
```

The following librarian job uses these saved librarian control statements:

```
     1        10    16
1.   // JOB ESCRUN,,,7400
2.   // DVC 20  //LFD PRNTR
3.   // DVC 130  // VOL PUBRES
4.   // LBL HAMMER  // LFD HAM
5.   // DVC 130  // VOL PUBRES
6.   // LBL SRCFIL  // LFD SRC
7.   // EXEC LIBS
8.   /$
9.            FIL   D1=HAM,D2=SRC
10.           ESC   SRC,LD,LIBTEST
11.  /*
12.  /&
```

1. JOB statement specifying additional bytes of main storage needed.

2. Device assignment set for a printer to produce a librarian map.

3 and 4.
   Device assignment set for file containing the diskette file HAMMER, which contains the module COBOL4.

5 and 6.
   Device assignment set for the diskette file SRCFIL. This file contains the source module LIBTEST to be accessed by the ESC statement. It is also the file where the module COBOL4 is to be copied.

7. Initiates execution of librarian.

8. Identifies the start of the librarian control statements.

9. Declares the files to be processed by the librarian job.

10. Initiates ESC processing of the saved librarian control statements.

11. Indicates the end of the librarian control statements.

12. Indicates end of job.

3.4. MIRAM Librarian Control Statement Descriptions

This section contains detailed descriptions of the MIRAM librarian control statements. Each statement description includes the statement function, syntax, and syntax examples. Information on the job control required to use the MIRAM librarian is described in 2.7. A sample job control stream in 2.7.6 illustrates how the control statements work together.

Table 3-2 contains a summary of the functions associated with each MIRAM librarian control statement. Each statement is described in detail in the subsection indicated.

Table 3-2. MIRAM Librarian Control Statement Summary

Statement	Function	Description
CHG	Changes the name or comment in a module header.	3.4.5
COP	Copies modules or files.	3.4.2
DEL	Deletes modules from files.	3.4.4
FIL	Assigns logical file names.	3.4.1
PRT	Prints a module listing or file directory.	3.4.3

3.4.1. Assigning Logical File Names (FIL)

Function

The FIL librarian control statement equates each disk, tape, and diskette file name used in an // LFD job control statement to a file with a logical file name. A logical file name for a MIRAM file must be in the range F0 through F29. The logical file name is then used exclusively to reference each file in subsequent librarian control statements.

Up to 30 files can be declared in a single FIL statement. A comma, no spaces, is used to separate successive file declarations in a single FIL statement. Disk, tape, and diskette files may be declared in the same FIL statement.

The FIL statement only assigns logical file names to files; it does not open the files. A file is only opened when its logical file name is used in a subsequent librarian control statement.

Up to six files can be open at one time. If more files are used, only the latest six remain open. Otherwise, a file remains open until the job step is completed or its logical file name is assigned to another file by another FIL librarian control statement in the job step. Therefore, if it is not necessary to have all the files open at one time during a librarian job step, it is a good idea to reuse the logical file names in another FIL librarian control statement. This way, the files will be closed when they are no longer needed.

Note: Because librarian control statements do not reference files by their actual names, the same set of librarian control statements can be used interchangeably to perform the same operations on any files; only // LFD, // LBL, and FIL statements must be changed to identify the new files.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	FIL	Fn=filename-1[,...,Fn=filename-n]

Options

None

Keyword Parameter

`Fn=filename`

Equates a MIRAM filename to a logical file name. The number *n* can range from 0 to 29.

The filename must match the file name used in the // LFD job control statement for the file, the system file name, or the job run file name YRUN.

Example

```
1        10    16
//  JOB FLTST
//  DVC 20  // LFD PRNTR
//  DVC 50  // VOL D00012  // LBL MINE  // LFD MY
//  DVC 50  // VOL D00020  // LBL YOUR  // LFD YR
//  DVC 50  // VOL D00012  // LBL HIS   // LFD HS
//  DVC 50  // VOL D00020  // LBL HERS  // LFD HR
//  EXEC MLIB
/$
          FIL   F0=MY,F1=YR
          COP   F0,,,F1
          FIL   F0=HS,F1=HR
          COP   F1,,,F1
/*
/&
```

The first FIL statement assigns the logical file name F0 to the disk file MINE and the logical file name F1 to the file YOUR. The first COP statement copies all modules from the file MINE to the file YOUR. The second FIL statement closes the file MINE and reuses the logical file name F0 by assigning it to the file HIS. The second FIL statement also closes the file YOUR and reuses the logical file name F1 by assigning it to the file HERS. The second COP statement then copies all modules from the file HIS to the file HERS.

3.4.2. Copying Modules and Files (COP)

Function

The COP librarian control statement is used to copy modules from one file to another. Options and parameters used in the COP statement determine which modules are copied. Deleted modules are never copied.

Modules are obtained from an input file and copied to the output file. When the output file already contains a module of the same name and type as the one being copied, the existing module is deleted from the output file and the new module is added.

Note: The SETREL and COPYREL canned job control streams are available to backup or copy files from a system release volume. See Appendix A for more information.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	COP[.options]	input-lfn,[module-type],[name],output-lfn

Options

C - Process only modules with the name prefix specified in the name parameter.

Positional Parameter 1

input-lfn

Specifies the logical file name of the file containing the modules to be copied.

Positional Parameter 2

module-type

Specifies the type of module being copied, as follows:

F - Type F screen format modules

FC - Type FC screen format modules

HELP - Help screen modules

J - Saved run library modules

MENU - Menu modules

If omitted, all modules with the specified name or name prefix (with C option) are copied. If both the name and type parameters are omitted, all modules in the file are copied.

When the file contains only screen format modules, the type parameter may be omitted to copy both the type F and type FC modules with the same name using one COP statement. When the file contains other types of modules, two COP statements are needed to copy two screen format modules with the same name.

Positional Parameter 3

name

Specifies a module name or a module prefix (with C option).

If omitted, all modules of the specified type are copied. If both the name and type parameters are omitted, all modules in the file are copied.

Positional Parameter 4

output-lfn

Specifies the logical file name of the file to which the modules are to copied.

Examples

```
    1       10    16
    |       |     |
1.  COP   F0,,,F12
2.  COP   F1,F,EXAMPLE2,F2
    COP   F1,FC,EXAMPLE2,F2
3.  COP   F3,,EXAMPLE3,F4
4.  COP   F5,J,EXAMPLE4,F6
5.  COP.C F7,,EXA,F8
6.  COP.C F9,J,EXA,F10
```

1. Copies all modules from file F0 to file F12.

2. This pair of statements copies the screen format modules named EXAMPLE2 from file F1 to file F2. The first copies the type F module; the second copies the type FC module. If file F1 contains only screen format modules, only one COP statement is needed to copy both the type F and type FC modules with the name EXAMPLE2. This statement must specify the name but omit the type (see Example 3 below).

3. Copies all modules named EXAMPLE3 from file F3 to F4.

4. Copies the saved run library module named EXAMPLE4 from file F5 to F6.
5. Copies all modules with the name prefix EXA from file F7 to F8.
6. Copies all saved run library modules with the prefix EXA from file F9 to F10.

3.4.3. Printing Listings of Modules and Directories (PRT)

Function

The PRT librarian control statement is used to do the following:

- Print listings of modules
- Print a file directory

Note: The LISTRES and MODLST canned job control streams are available to print listings and directories of files on a system release volume. See 3.5.1 and 3.5.3 for more information.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	PRT[.options]	lfn[,module-type][,name]

Options

C - Process only modules with the name prefix specified in the name parameter.

D - Print file directory records only. If used, the name parameter must be omitted. If not used, print module listings.

Positional Parameter 1

lfn

Specifies the logical file name of the file to be processed.

Positional Parameter 2

module-type

Specifies the type of module to be processed, as follows:

F - Type F screen format modules

FC - Type FC screen format modules

HELP - Help screen modules

J - Saved run library modules

MENU - Menu modules

If the type is omitted and the D option is used, all records in the file directory are printed. If both the type and D option are used, only module header records of that type are printed. If both the name and type parameters are omitted, all modules in the file are listed.

When the file contains only screen format modules, the type parameter may be omitted to process both the type F and type FC modules with the specified name. When the file contains other types of modules, two PRT statements are needed to process the modules of each type.

Positional Parameter 3

name

Specifies a module name or a module name prefix (with C option).

If omitted, all modules of the specified type or all modules with the specified name prefix (with C option) are listed. If both the C option and the type parameter are used, all modules of the specified type with that name prefix are listed. If both the name and type parameters are omitted, all modules in the file are listed.

This parameter must be omitted when the D option is used.

Examples

```
     1        10    16
     ┌──────────────────────────────────────────
1.   │        PRT   F0
2.   │        PRT   F2,F,EXAMPLE2
     │        PRT   F2,FC,EXAMPLE2
3.   │        PRT   F3,,EXAMPLE3
4.   │        PRT.C F4,J,EXA
5.   │        PRT.C F5,,EXA
6.   │        PRT.D F6
7.   │        PRT.D F7,MENU
```

1. Prints listings of all modules in file F0.

2. This pair of statements prints listings of the screen format modules named EXAMPLE2 in file F2. The first lists the type F module; the second lists the type FC module. If file F2 contains only screen format modules, only one PRT statement is needed to list both the type F and type FC modules named EXAMPLE2. This statement must specify the name, but omit the type (see Example 3 below).

3. Prints listings of all modules of any type named EXAMPLE3 in file F3.

4. Prints listings of all saved run library modules with the name prefix EXA in file F4.

5. Prints listings of all modules of any type with the name prefix EXA in file F5.

6. Prints all directory records in file F6.

7. Prints header records for all menu modules in file F7.

3.4.4. Deleting Modules from Files (DEL)

Function

The DEL librarian control statement is used to delete one or more modules from a file. The use of options and parameters in the DEL control statement allow the following variations:

- Delete all modules in a file.
- Delete the module with the specified name and type.
- Delete all modules with a specified name.
- Delete all modules with a specified name prefix.
- Delete all modules of a specified type.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	DEL[.options]	lfn[,module-type][,name]

Options

C - Process only modules with the name prefix specified in the name parameter.

Positional Parameter 1

lfn

Specifies the logical file name of the file containing the modules to be deleted.

Positional Parameter 2

module-type
Specifies the type of module being deleted, as follows:

F - Type F screen format modules

FC - Type FC screen format modules

HELP - Help screen modules

J - Saved run library modules

MENU - Menu modules

If omitted, all modules with the specified name or name prefix (with C option) are deleted. If both the name and type parameters are omitted, all modules in the file are deleted.

When the file contains only screen format modules, the type parameter may be omitted to delete both the type F and type FC modules with the same name. When the file contains other types of modules, two DEL statements are needed to delete the modules of each type.

Positional Parameter 3

name
Specifies a module name or a module name prefix (with C option).

If omitted, all modules of the specified type are deleted. If both the name and type parameters are omitted, all modules in the file are deleted.

Examples

```
     1        10    16
     |        |     |
1.   DEL   F0
2.   DEL   F1,F,EXAMPLE2
     DEL   F1,FC,EXAMPLE2
3.   DEL   F3,,EXAMPLE3
4.   DEL   F4,J,EXAMPLE4
5.   DEL.C F5,,EXA
6.   DEL.C F6,MENU,EXA
```

1. Deletes all modules in file F0.

2. This pair of statements deletes the screen format modules named EXAMPLE2 from file F1. The first deletes the type F module; the second deletes the type FC module. If file F1 contains only screen format modules, only one DEL statement is needed to delete both the type F and type FC

modules named EXAMPLE2. This statement must specify the module name but omit the type (see example 3 below).

3. Deletes all modules named EXAMPLE3 from file F3.

4. Deletes the saved run module named EXAMPLE4 from file F4.

5. Deletes all modules with the name prefix EXA from file F5.

6. Deletes all menu modules with the prefix EXA from file F6.

3.4.5. Changing Module Names and Comments (CHG)

Function

The CHG librarian control statement is used to change the name of an existing module or change comments in a module header record. Each CHG statement can make only one change, either a name change or a comment change, for one module.

If a module name is being changed and the file already contains a module of the specified type and the new name, an error message is printed on the librarian map and no change is made.

A new comment replaces any exisiting comment in its entirety.

Format

LABEL	ΔOPERATIONΔ	OPERAND
unused	CHG[.options]	lfn,module-type,old-name,{N,new-name / C,'comment'}

Options

None

Positional Parameter 1

`lfn`

Specifies the logical file name of the file containing the module being processed.

Positional Parameter 2

`module-type`

Specifies the type of module being processed, as follows:

F - Type F screen format modules

FC - Type FC screen format modules

HELP - Help screen modules

J - Saved run library modules

MENU - Menu modules

Positional Parameter 3

`old-name`
Specifies the name of the module being processed.

Positional Parameter 4

`N,C`
Identifies the kind of change to be made. N specifies a module name change; C specifies a comment change in the module header record. If N is used, a new-name must be supplied in positional parameter 5. If C is used, a comment must be supplied in positional parameter 5.

Positional Parameter 5

`new-name`
Specifies a new module name containing up to eight letters or digits. The first character must be a letter. If used, N must be used in positional parameter 4.

`'comment'`
Specifies a new comment of up to 30 characters to be inserted in the module header record. The comment must be enclosed in single quotation marks. If used, C must be used in positional parameter 4.

Examples

```
    1       10    16
    ┌──────────────────────────────────────────────────
1.  │       CHG   F0,F,VERSION1,N,VERSION2
    │       CHG   F0,FC,VERSION1,N,VERSION2
2.  │       CHG   F2,J,RUN6,N,RUN7
    │       CHG   F2,J,RUN7,C,'REVISED BY CGM'
3.  │       CHG   F3,J,CHS001,C,'UPDATED 2/16/82'
```

1. This pair of statements change the name of both screen format modules (type F and type FC) from VERSION1 to VERSION2 in file F0.

2. The first statement changes the name of the saved run library module in file F2 from RUN6 to RUN7. The second statement changes the comment in the header record for that same module to 'REVISED BY CGM'.

3. Changes the comment in the header record for the saved run library module named CHS001 to 'UPDATED 2/16/82'.

3.5. Canned Librarian Job Control Streams

This section describes the canned librarian job control streams provided in the system library. The job control streams reside in the system job control stream file YJCS; the associated load modules which are actually run reside in the system load file YLOD.

Each job control stream is executed by keying in its associated RV command at the system console. Table 3-3 lists each job name and the function it performs. Each function is described in detail in the section indicated. These canned librarian job control streams are summarized along with other available canned job control streams in Appendix A.

Table 3-3. Canned Librarian Job Control Streams

Job Name	Function	Description
LISTRES	Prints a directory of modules in a release volume.	3.5.1
DRDP	Prints a disk display of any disk file directory.	3.5.2
MODLIST	Prints listings of the modules in the system libraries of a release volume	3.5.3
PACKRES	Compresses all files in a release volume and prints updated directories.	3.5.4

3.5.1. Printing a Directory of a Release Volume (LISTRES)

Function

LISTRES prints a directory of all modules in a release volume or all modules in a particular file in a release volume. See Table 3-4 for a list of the modules included.

Note: LISTRES can list a directory only for the SYSRES volume, a release volume, or a backup copy of a release volume because it assumes that certain files exist in the volume being processed. If the volume specified is not a release volume and these files are not present, errors may occur.

Command Format

```
RV LISTRES[,[,F=filename][,V=vsn]]
```

F Keyword Parameter

F=filename

If used, a directory of all modules in the specified system file is listed. If omitted, a directory of the entire release volume specified is listed. Table 3-4 lists the abbreviated filenames to be use for the filename parameter. More than one filename may be specified. The filenames must be separated by commas and enclosed in parentheses (see Examples 2 and 3).

V Keyword Parameter

V=vsn

Specifies the volume serial number of the release volume to be processed. If omitted, the active system-resident pack is assumed.

Examples

```
1.  RV LISTRES,,F=OBJ,V=REL120
2.  RV LISTRES,,F=(LOD,JCS,SRC)
3.  RV LISTRES,,F=(HELP,MSG),V=REL120
4.  RV LISTRES,,V=REL120
5.  RV LISTRES
```

1. Prints directory of all modules in the file YOBJ on release volume REL120.

2. Prints directory of all modules in the files YLOD, YJCS, and YSRC on the active system-resident pack.

3. Prints directory of all modules in the files YHELP and YMSG on release volume REL120.

4. Prints a directory of all modules in all files in release volume REL120.

5. Prints a directory of all modules in all files on the active system-resident pack.

Table 3-4. Files Processed by LISTRES and PACKRES

F=filename	Actual Filename	File content
LOD	YLOD	System load modules
OBJ	YOBJ	System object modules
MAC	YMAC	System macro modules
SRC	YSRC	System source modules
JCS	YJCS	System job control streams
SMC	SMCFILE	System maintenance correction file
HELP	YHELP	System help screens
SHR	YSHR	Information on active shared code modules
SDF	YSDF	System definition file indicating microcode level of all workstations
IVP	IVPLIB	Installation verification programs
SCLOD	YSCLOD	System shared code load modules
MIC	YMIC	Microcode for loadable components
SAVE	YSAVE	System saved job run streams
DLF	YDIALOG	Dialog screens for job control and system generation
FMT	YFMT	System screen format modules
MSG	YMSG	System messages
SG$JCS	SG$JCS	SYSGEN job control streams
SG$LOD	SG$LOD	SYSGEN load modules
SG$OBJ	SG$OBJ	SYSGEN object modules
SG$MAC	SG$MAC	SYSGEN macro modules

3.5.2. Printing a Disk Display of a Disk File Directory (DRDP)

Function

DRDP prints a sequential disk display of any disk file directory.

Command Format

```
RV DRDP,,V=vsn,L=file-identifier
```

V Keyword Parameter

V=vsn
: Specifies the volume serial number of the volume containing the file to be processed.

L Keyword Parameter

L=file-identifier
: Specifies the file-identifier for that file, as recorded in the volume table of contents.

Examples

```
1. RV DRDP,,V=DSK001,L=RTCHG
2. RV DRDP,,V=D00022,L=CUST88
```

1. Prints a disk display of the directory for the file RTCHG on disk volume DSK001.

2. Prints a disk display of the directory for the file CUST88 on disk volume D00022.

Note: The maximum length for the file-identifier is 11 characters. To process a file with a file-identifier of more than 11 characters, use the following job control stream:

```
1        10    16
// JOB   DRDP
// DVC   20    // LFD PRNTR
// DVC   50    // VOL vsn  // LBL file-identifier
// LFD   LU$DTFI
// OPTION JOBDUMP
// EXEC  SULBD
// PARAM file-identifier
/&
// FIN          Required only if job is run using punched cards.
```

3.5.3. Printing Listings of Modules in the System Libraries of a Release Volume (MODLST)

Function

MODLST prints listings of all modules and macros in the five system libraries YSRC, YOBJ, YLOD, YJCS, and YMAC. Each module and macro is listed in alphanumeric sequence along with a description of its function and size.

Note: MODLST can only print listings of modules in the SYSRES volume, a release volume, or a backup copy of a release volume because it assumes that certain release files exist in the volume being processed. If the volume specified is not a release volume and these files are not present, errors may occur. To print listings of modules in any other volumes, use the SAT or MIRAM librarian.

Command Format

```
RV MODLST[,,VSN=vsn]
```

VSN Keyword Parameter

VSN=vsn
: Specifies the volume serial number for a disk where a 30-cylinder workspace is to be allocated for processing MODLST. If not specified, the disk containing the job run file YRUN is used.

Examples

```
1.  RV MODLST,,VSN=WRKDSK
2.  RV MODLST
```

1. Prints listings of all modules in YSRC, YOBJ, YLOD, YJCS, and YMAC on the active system-resident pack. A work space is allocated on the disk WRKDSK.

2. Prints listings of all modules in YSRC, YOBJ, YLOD, YJCS, and YMAC on the active system-resident pack.

3.5.4. Compressing a Release Volume and Printing Update Directories (PACKRES)

Function

PACKRES compresses and prints updated directories of the module header information for files in a release volume. The files are listed in Table 3-4.

Notes:

1. *Before PACKRES is run, the system must be loaded using the IPL procedure described in the* Operations Guide *(UP-8859).*

2. *PACKRES can only compress files in the SYSRES volume, a release volume, or a backup copy of a release volume because it assumes that certain release files exist in the volume being processed. If the volume specified is not a release volume and these files are not present, errors may occur. To compress files in any other volumes, run a librarian job using the SAT librarian PAC control statement or the MIRAM librarian COP control statement.*

3. *PACKRES must be run only in an idle system. Use the MI DA command at the system console to ensure that no other jobs or symbionts are active. If interactive services is still active, it must be shut down. Any users that are still logged on must be logged off and then a 00 IS SHUTDOWN command must be executed to remove interactive services from the system.*

4. *The system must be re-IPLed after PACKRES completes to ensure that all system file pointers are reset.*

Command Format

```
RV PACKRES[,[,F=filename][,V=vsn]]
```

F Keyword Parameter

`F=filename`

If used, all modules in the specified system file are processed. If omitted, all files in the the specified release volume are processed. Table 3-4 lists the abbreviated filenames for the filename parameter. More than one filename may be specified. The filenames must be separated by commas and enclosed in parentheses (see Examples 2 and 3 under "Examples").

V Keyword Parameter

`V=vsn`

Specifies the volume serial number of the release volume to be processed. If omitted, the active system-resident pack is assumed.

Examples

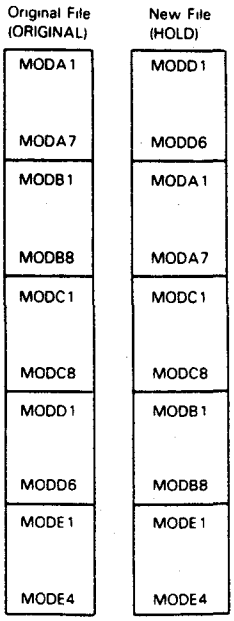
```
1.  RV PACKRES,,F=OBJ,V=REL120
2.  RV PACKRES,,F=(LOD,JCS,SRC)
3.  RV PACKRES,,F=(HELP,FMT),V=REL120
4.  RV PACKRES,,V=REL120
5.  RV PACKRES
```

1. Compresses the file YOBJ on release volume REL120.
2. Compresses the files YLOD, YJCS, and YSRC on the active system-resident pack.
3. Compresses the files YHELP and YFMT on release volume REL120.
4. Compresses all files in release volume REL120.
5. Compresses all files on the active system-resident pack.

3.6. SAT Librarian Programming Examples

3.6.1. Rearranging Modules in a Disk File

This job rearranges modules in a disk file. It copies modules from the original file into a new file in the new sequence as shown in the following listing. The names such as MODA1 and MODA7 are the names of the first and last modules in each series of consecutive modules that are copied with each COP statement. After all the modules are copied to the new file, the original file is scratched and the new file is renamed as the original. Figure 3-5 illustrates the librarian map for this job.



Job Control Stream

```
1.  // JOB SHUFFLE
2.  // DVC 20  // LFD PRNTR
3.  // DVC 50  // VOL D00410
4.  // LBL ORIGINAL  // LFD RG
5.  // DVC 50  // VOL D00410
6.  // EXT ST,,1,BLK,(256,4000)
7.  // LBL HOLD  // LFD HD
8.  // EXEC LIBS
9.  /$
10.           FIL   D1=RG,D2=HD
11.           COP   D1
12.           RES   D1,S,MODD1
13.           COP.U D1,S,MODD6,D2
14.           RES   D1,S,MODA1
15.           COP.U D1,O,MODA7,D2
16.           RES   D1,S,MODC1
17.           COP.U D1,L,MODC8,D2
18.           RES   D1,S,MODB1
19.           COP.U D1,L,MODB8,D2
20.           RES   D1,S,MODE1
21.           COP.U D1,S,MODE4,D2
22.           COP   D2
23. /*
24. // SKIP END,11111111
25. // SCR RG
26. // REN HD,ORIGINAL
27. //END NOP
28. /&
```

1. Identifies the job.

2. Assigns a printer to the job.

3. Identifies the logical unit number 50 for the disk volume D00410, which contains the original file.

4. Declares the file name ORIGINAL and the logical file descriptor RG for the original file.

5. Identifies the logical unit number 50 for the disk volume D00410 to be used for the new file.

6. Allocates file space for the new file.

7. Declares the file name HOLD and the logical file descriptor HD for the new file.

8. Initiates execution of the librarian.

9. Indicates the start of the librarian control statements.

10. Assigns a type code and a logical file number to the files in the job. Thus, in the control statements that follow, D1 refers to ORIGINAL and D2 refers to HOLD.

11. Prints a sequential list of the modules in the file ORIGINAL before the modules are rearranged.

12. through 21.
    Copy modules from the file ORIGINAL to the file HOLD, moving a series of consecutive modules at a time. Each RES statement sets the pointer in the file ORIGINAL to the first module in the series. Then a COP statement copies all modules from the pointer to the module named in the COP statement.

22. Prints a sequential list of the modules in the file HOLD.

23. Identifies the end of the librarian control statements.

24. and 27.
    Skip the scratch and rename operation if any errors occur in the job.

25. Scratches the file ORIGINAL.

26. Renames the file HOLD to ORIGINAL.

28. Indicates the end of job.

Note: To save both the original file and the new file, omit lines 24 through 27.

```
UNISYS OS/3 LIBRARIAN                                                PAGE # 0001
DATE 88/07/06  TIME 11.58                                                VFR820401

BLOCK   REC          NAME       TYPE    DATE      TIME     COMMENTS

.. COMMAND .....     FIL        D1=RG,D2=HD

                     D 1 - VSN IS 000410, LFD IS  RG      , FILE LABEL IS ORIGINAL
                     D 2 - VSN IS 000410, LFD IS  HD      , FILE LABEL IS HOLD

.. COMMAND .....     COP        D1
```

Figure 3-5. Librarian Map for Repositioning Modules (Part 1 of 5)

```
                                                                    PAGE # 0002
BLOCK   REC          NAME       TYPE     DATE       TIME      COMMENTS

                                  TABLE OF CONTENTS

          SOURCE     MODA1      81/04/27   09.14
          SOURCE     MODA2      81/04/27   09.22
          LOAD       MODA3000   81/05/06   11.10
          LOAD       MODA4000   81/05/08   10.44
          SOURCE     MODA5      81/05/08   11.06
          SOURCE     MODA6      81/05/12   13.45
          OBJECT     MODA7      81/05/12   13.48
          SOURCE     MODB1      81/05/12   13.59
          SOURCE     MODB2      81/06/01   12.33
          LOAD       MODB3000   81/06/01   12.50
          SOURCE     MODB4      81/06/05   12.45
          SOURCE     MODB5      81/06/29   12.38
          LOAD       MODB6000   81/07/06   14.52
          LOAD       MODB7000   81/07/10   14.14
          LOAD       MODB8000   81/08/14   13.31
          SOURCE     MODC1      81/08/14   13.35
          SOURCE     MODC2      81/08/21   10.51
          SOURCE     MODC3      81/08/24   10.43
          SOURCE     MODC4      81/08/24   10.47
          LOAD       MODC5000   81/08/24   10.54
          SOURCE     MODC6      81/09/01   10.28
          LOAD       MODC7000   81/09/11   08.30
          LOAD       MODC9000   00/00/00   00.08
          SOURCE     MODD1      80/08/08   16.34
          SOURCE     MODD2      80/08/08   16.35
          SOURCE     MODD3      80/08/08   16.37
          SOURCE     MODD4      80/08/08   16.39
          SOURCE     MODD5      80/08/08   16.41
          SOURCE     MODD6      80/08/08   16.43
          SOURCE     MODE1      80/08/08   16.44
          SOURCE     MODE2      80/08/08   16.45
          SOURCE     MODE3      80/08/08   16.45
          SOURCE     MODE4      80/08/08   16.47

                   BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000

.. COMMAND .........   RES        D1,S,MODD1

.. COMMAND .........   COP.U      D1,S,MODD6,D2

000001  005            MODD1      SOR     80/08/08   16.34
000004  052            MODD2      SOR     80/08/08   16.35
000005  067            MODD3      SOR     80/08/08   16.37
000007  005            MODD4      SOR     80/08/08   16.39
000008  005            MODD5      SOR     80/08/08   16.41
```

Figure 3-5. Librarian Map for Repositioning Modules (Part 2 of 5)

```
                                                                    PAGE # 0003
BLOCK   REC          NAME        TYPE    DATE      TIME     COMMENTS

000008  171          MODD6       SOR     80/08/08  16.43

.. COMMAND .........  RES          D1,S,MODA1

.. COMMAND .........  COP.U        D1,O,MODA7,D2

000009  118          MODA1       SOR     81/04/27  09.14
000014  031          MODA2       SOR     81/04/27  09.22
000015  179          MODA3000    LOD     81/05/06  11.10
000019  077          MODA4000    LOD     81/05/08  10.44
000020  021          MODA5       SOR     81/05/08  11.06
000022  150          MODA6       SOR     81/05/12  13.45
000023  126          MODA7       OBJ     81/05/12  13.48

.. COMMAND .........  RES          D1,S,MODC1

.. COMMAND .........  COP.U        D1,L,MODC8,D2

000032  005          MODC1       SOR     81/08/14  13.35
000034  034          MODC2       SOR     81/08/21  10.51
000035  034          MODC3       SOR     81/08/24  10.43
000039  030          MODC4       SOR     81/08/24  10.47
000040  005          MODC5000    LOD     81/08/24  10.54
000043  021          MODC6       SOR     81/09/01  10.28
000044  187          MODC7000    LOD     81/09/11  08.30
000052  164          MODC8000    LOD     00/00/00  00.08

.. COMMAND .........  RES          D1,S,MODB1

.. COMMAND .........  COP.U        D1,L,MODB8,D2

000058  057          MODB1       SOR     81/05/12  13.59
000061  005          MODB2       SOR     81/06/01  12.33
000074  123          MODB3000    LOD     81/06/01  12.50
000093  021          MODB4       SOR     81/06/05  12.45
000084  142          MODB5       SOR     81/06/29  12.38
000086  034          MODB6000    LOD     81/07/06  14.52
000118  365          MODB7000    LOD     81/07/10  14.14
000174  021          MODB8000    LOD     81/08/14  13.31

.. COMMAND .........  RES          D1,S,MODE1

.. COMMAND .........  COP.U        D1,S,MODE4,D2

000182  180          MODE1       SOR     80/08/08  16.44
000184  076          MODE2       SOR     80/08/08  16.45
000185  065          MODE3       SOR     80/08/08  16.45
000186  149          MODE4       SOR     80/08/08  16.47
```

Figure 3-5. Librarian Map for Repositioning Modules (Part 3 of 5)

```
BLOCK    REC          NAME      TYPE     DATE      TIME      COMMENTS                    PAGE # 0004

.. COMMAND .........  COP       D2
```

Figure 3-5. Librarian Map for Repositioning Modules (Part 4 of 5)

```
BLOCK   REC        NAME       TYPE     DATE      TIME      COMMENTS                    PAGE # 0005

                                    TABLE OF CONTENTS

          SOURCE    MODD1      80/08/08  16.34
          SOURCE    MODD2      80/08/08  16.35
          SOURCE    MODD3      80/08/08  16.37
          SOURCE    MODD4      80/08/08  16.39
          SOURCE    MODD5      80/08/08  16.41
          SOURCE    MODD6      80/08/08  16.43
          SOURCE    MODA1      81/04/27  09.14
          SOURCE    MODA2      81/04/27  09.22
          LOAD      MODA3000   81/05/06  11.10
          LOAD      MODA4000   81/05/08  10.44
          SOURCE    MODA5      81/05/08  11.06
          SOURCE    MODA6      81/05/12  13.45
          OBJECT    MODA7      81/05/12  13.48
          SOURCE    MODC1      81/08/14  13.35
          SOURCE    MODC2      81/08/21  10.51
          SOURCE    MODC3      81/08/24  10.43
          SOURCE    MODC4      81/08/24  10.47
          LOAD      MODC5000   81/08/24  10.54
          SOURCE    MODC6      81/09/01  10.28
          LOAD      MODC7000   81/09/11  08.30
          LOAD      MODC8000   00/00/00  00.08
          SOURCE    MODP1      81/05/12  13.59
          SOURCE    MODP2      81/06/01  12.33
          LOAD      MODP3000   81/06/01  12.50
          SOURCE    MODP4      81/06/05  12.45
          SOURCE    MODP5      81/06/29  12.38
          LOAD      MODP6000   81/07/06  14.52
          LOAD      MODP7000   81/07/10  14.14
          LOAD      MODP8000   81/09/14  13.31
          SOURCE    MODE1      80/08/08  16.44
          SOURCE    MODE2      80/08/08  16.45
          SOURCE    MODE3      80/08/08  16.45
          SOURCE    MODE4      80/08/08  16.47

                    BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


LIBRARIAN FINISHED
DATE  88/07/06  TIME 11.58
TOTAL NUMBER OF ERRORS 00000 UPSI SETTING X'00'
```

Figure 3-5. Librarian Map for Repositioning Modules (Part 5 of 5)

3.6.2. Sorting Modules into Separate Files by Type

This job sorts modules from one file into separate files for each module type. This job was run interactively, so the files for each module type were allocated with an interactive services ALLOCATE command before the job was run. Figure 3-6 illustrates the librarian map for this job. Note that the last COP statement caused an error because there were no macro/jproc source modules in the original file. However, the job terminated normally.

Job Control Stream

```
1.  // JOB TYPSRT
2.  // DVC 20  // LFD PRNTR
3.  // DVC 50  // VOL D00410
4.  // LBL ORIGINAL  // LFD RG
5.  // DVC 50  // VOL D00410
6.  // LBL ALLSRC  // LFD SC
7.  // DVC 50  // VOL D00410
8.  // LBL ALLOBJ  // LFD OB
9.  // DVC 50  // VOL D00410
10. // LBL ALLLOD  // LFD LD
11. // DVC 50  // VOL D00410
12. // LBL ALLMAC  // LFD MC
13. // EXEC LIBS
14. /$
15.           FIL   D1=RG,D2=SC,D3=OB,D4=LD,D5=MC
16.           COP   D1
17.           COP   D1,S,,D2
18.           COP   D1,O,,D3
19.           COP   D1,L,,D4
20.           COP   D1,M,,D5
21. /*
22. /&
```

1. Identifies the job.

2. Assigns a printer to the job.

3. Identifies the logical unit number 50 for the disk volume D00410 that contains the original file.

4. Declares the file ORIGINAL and the logical file descriptor RG for the original file.

5. through 12.
   Are device assignment statements for the files for each module type. All are on the disk volume D00410 with the logical unit number 50. The files are named ALLSRC, ALLOBJ, ALLLOD, and ALLMAC and are assigned the logical file descriptors SC, OB, LD, and MC, respectively.

13. Initiates execution of the librarian.

14. Indicates the start of the librarian control statements.

15. Assigns a type code and logical file number to the five files used in the job. Thus, in the control statements that follow, D1 refers to ORIGINAL, D2 to ALLSCR, D3 to ALLOBJ, D4 to ALLLOD, and D5 to ALLMAC.

16. Prints a table of contents of all modules in the file ORIGINAL.

17. Copies all source modules from the file ORIGINAL to the file ALLSCR and prints a list of all of the modules copied.

18. Copies all object modules from the file ORIGINAL to the file ALLOBJ and prints a list of all of the modules copied.

19. Copies all load modules from the file ORIGINAL to the file ALLLOD and prints a list of all of the modules copied.

20. Copies all macro/jproc modules from the file ORIGINAL to the file ALLMAC and prints a list of all of the modules copied.

21. Indicates the end of the librarian control statements.

22. Indicates the end of job.

```
                                                                    PAGE # 0001
UNISYS OS/3 LIBRARIAN                                                VER820401
DATE 88/07/08  TIME 15.39

BLOCK    REC          NAME       TYPE    DATE      TIME      COMMENTS


.. COMMAND .........  FIL          D1=RG,D2=SC,D3=OB,D4=LD,D5=MC

                      D 1 - VSN IS D00410, LFD IS  RG      , FILE LABEL IS ORIGINAL
                      D 2 - VSN IS D00410, LFD IS  SC      , FILE LABEL IS ALLSRC
                      D 3 - VSN IS D00410, LFD IS  OB      , FILE LABEL IS ALLOBJ
                      D 4 - VSN IS D00410, LFD IS  LD      , FILE LABEL IS ALLLOD
                      D 5 - VSN IS D00410, LFD IS  MC      , FILE LABEL IS ALLMAC

.. COMMAND .........  COP          D1
```

Figure 3-6. Librarian Map for Sorting Modules by Type (Part 1 of 3)

```
BLOCK   REC          NAME       TYPE     DATE       TIME      COMMENTS                    PAGE # 0002

                                        TABLE OF CONTENTS

          SOURCE     MODD1      80/08/08   16.34
          SOURCE     MODD2      80/08/08   16.35
          SOURCE     MODD3      80/09/08   16.37
          SOURCE     MODD4      80/08/08   16.39
          SOURCE     MODD5      80/08/08   16.41
          SOURCE     MODD6      80/08/08   16.43
          SOURCE     MODA1      81/04/27   09.14
          SOURCE     MODA2      81/04/27   09.22
          LOAD       MODA3000   81/05/06   11.10
          LOAD       MODA4000   81/05/08   10.44
          SOURCE     MODA5      81/05/08   11.06
          SOURCE     MODA6      81/05/12   13.45
          OBJECT     MODA7      81/05/12   13.48
          SOURCE     MODC1      81/08/14   13.35
          SOURCE     MODC2      81/08/21   10.51
          SOURCE     MODC3      81/08/24   10.43
          SOURCE     MODC4      81/08/24   10.47
          LOAD       MODC5000   81/08/24   10.54
          SOURCE     MODC6      81/09/01   10.28
          LOAD       MODC7000   81/09/11   08.30
          LOAD       MODC9000   00/00/00   00.08
          SOURCE     MODB1      81/05/12   13.59
          SOURCE     MODB2      81/06/01   12.33
          LOAD       MODB3000   81/06/01   12.50
          SOURCE     MODB4      81/06/05   12.45
          SOURCE     MODB5      81/06/29   12.38
          LOAD       MODB6000   81/07/06   14.52
          LOAD       MODB7000   81/07/10   14.14
          LOAD       MODB8000   81/08/14   13.31
          SOURCE     MODE1      80/08/08   16.44
          SOURCE     MODE2      80/08/08   16.45
          SOURCE     MODE3      80/08/08   16.45
          SOURCE     MODE4      80/08/08   16.47

                     BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000

.. COMMAND .........   COP        D1,S,,D2

000001   005           MODD1      SOR    80/08/08   16.34
000004   052           MODD2      SOR    80/08/08   16.35
000005   067           MODD3      SOR    80/08/08   16.37
000007   005           MODD4      SOR    80/08/08   16.39
000008   005           MODD5      SOR    80/08/08   16.41
000008   171           MODD6      SOR    80/08/08   16.43
000009   118           MODA1      SOR    81/04/27   09.14
000014   031           MODA2      SOR    81/04/27   09.22
```

Figure 3-6. Librarian Map for Sorting Modules by Type (Part 2 of 3)

```
                                                                                PAGE # 0003
BLOCK   REC          NAME        TYPE     DATE       TIME      COMMENTS

000015  179          MODA5       SOR      81/05/08   11.06
000018  072          MODA6       SOR      81/05/12   13.45
000019  071          MODC1       SOR      81/08/14   13.35
000021  080          MODC2       SOR      81/08/21   10.51
000022  074          MODC3       SOR      81/08/24   10.43
000026  095          MODC4       SOR      81/08/24   10.47
000027  062          MODC6       SOR      81/09/01   10.28
000029  005          MODB1       SOR      81/05/12   13.59
000031  135          MODB2       SOR      81/06/01   12.33
000045  005          MODB4       SOR      81/06/05   12.45
000046  091          MODB5       SOR      81/06/29   12.38
000048  005          MODE1       SOR      80/08/08   16.44
000049  135          MODE2       SOR      80/08/08   16.45
000050  130          MODE3       SOR      80/08/08   16.45
000052  005          MODE4       SOR      80/08/08   16.47

.. COMMAND .........  COP         D1,O,,D3

000001  005          MODA7       OBJ      81/05/12   13.48

.. COMMAND .........  COP         D1,L,,D4

000001  005          MODA3000    LOD      81/05/06   11.10
000004  077          MODA4000    LOD      81/05/08   10.44
000005  021          MODC5000    LOD      81/08/24   10.54
000008  176          MODC7000    LOD      81/09/11   08.30
000016  164          MODC8000    LOD      00/00/00   00.08
000022  057          MODB3000    LOD      81/06/01   12.50
000030  021          MODB6000    LOD      81/07/06   14.52
000062  065          MODB7000    LOD      81/07/10   14.14
000118  021          MODa8000    LOD      81/08/14   13.31

.. COMMAND .........  COP         D1,M,,D5

            8060*****NOTHING FOUND


LIBRARIAN FINISHED
DATE 88/07/08  TIME 15.39
TOTAL NUMBER OF ERRORS 00001 UPSI SETTING X'40'
```

Figure 3-6. Librarian Map for Sorting Modules by Type (Part 3 of 3)

3.6.3. Building Module Groups

This job builds two module groups by copying modules from other files. All the files have already been allocated. Figure 3-7 illustrates the librarian map for this job.

Job Control Stream

```
1.  // JOB GROUP
2.  // DVC 20  // LFD PRNTR
3.  // DVC 50  // VOL D00410  // LBL ORIGINAL  // LFD RG
4.  // DVC 50  // VOL D00410  // LBL ALLLOD    // LFD LD
5.  // DVC 50  // VOL D00410  // LBL ALLSRC    // LFD SC
6.  // DVC 50  // VOL D00410  // LBL MIXED     // LFD MX
7.  // EXEC LIBS
8.  /$
9.          FIL    D1=RG,D2=LD,D3=SC,D4=MX
10.         BOG    GROUPMIX,D4
11.         COP    D1,S,MODA1,D4
12.         COP    D3,S,MODC3,D4
13.         COP    D1,O,MODA7,D4
14.         EOG    GROUPMIX,D4
15.         COP    D4
16.         BOG    LOADS,D1
17.         RES    D2,L,MODC5000
18.         COP.U  D2,L,MODC8000,D1
19.         EOG    LOADS,D1
20.         COP    D1
21. /*
22. /&
```

1. Identifies the job.

2. Assigns a printer to the job.

3. through 6.
   Declare files for the job. All are on the disk volume D00410 with the logical unit number 50. Their names are ORIGINAL, ALLLOD, ALLSRC, and MIXED with logical file descriptors RG, LD, SC, and MX, respectively.

7. Initiates execution of the librarian.

8. Indicates the start of librarian control statements.

9. Assigns a type and logical file number to each of the files used in the job. Thus, in the control statements that follow, D1 refers to the file ORIGINAL, D2 to ALLLOD, D3 to ALLSRC, and D4 to MIXED.

10. through 14.
    Build the group GROUPMIX in the file MIXED. The BOG statement writes the beginning-of-group record in the file MIXED. Line 11 copies the source module MODA1 from the file ORIGINAL; line 14 copies the source module MODC3 from the file ALLSRC; and line 15 copies the object module MODA7 from the file ORIGINAL. The EOG statement writes the end-of-group record in the file MIXED.

15. Prints a table of contents for the file GROUPMIX.

16. Writes the beginning-of-group record for the group LOADS in the file ORIGINAL.

17. Sets the pointer in the file ALLLOD to the load module MODC5.

18. Copies all modules from MODC5 to MODC8 in the file ALLLOD to the group LOADS in the file ORIGINAL.

19. Writes an end-of-group record for the group LOADS in the file ORIGINAL.

20. Prints a table of contents for the file ORIGINAL.

21. Indicates the end of the librarian control statements.

22. Indicates the end of job.

```
UNISYS OS/3 LIBRARIAN                                          PAGE # 0001
DATE  88/07/08  TIME 15.42                                     VER820401

BLOCK   REC          NAME       TYPE    DATE       TIME      COMMENTS


.. COMMAND .....     FIL         D1=RG,D2=LD,D3=SC,D4=MX

                     D 1 - VSN IS D00410, LFD IS  RG       , FILE LABEL IS ORIGINAL
                     D 2 - VSN IS D00410, LFD IS  LD       , FILE LABEL IS ALLLOD
                     D 3 - VSN IS D00410, LFD IS  SC       , FILE LABEL IS ALLSRC
                     D 4 - VSN IS D00410, LFD IS  MX       , FILE LABEL IS MIXED

.. COMMAND .....     BOG         GROUPMIX,D4

000001  005          GROUPMIX    BOG

.. COMMAND .....     COP         D1,S,MODA1,D4

000001  045          MODA1       SOR     81/04/27   09.14

.. COMMAND .....     COP         D3,S,MODC3,D4

000005  191          MODC3       SOR     81/08/24   10.43

.. COMMAND .....     COP         D1,O,MODA7,D4

000009  188          MODA7       OBJ     81/05/12   13.48

.. COMMAND .....     EOG         GROUPMIX,D4

000017  228          GROUPMIX    EOG

.. COMMAND .....     COP         D4
```

Figure 3-7. Librarian Map for Building Module Groups (Part 1 of 3)

```
BLOCK   REC          NAME       TYPE    DATE       TIME      COMMENTS                    PAGE # 0002

                                        TABLE OF CONTENTS

  BOG     GROUPMIX
          SOURCE     MODA1      81/04/27   09.14
          SOURCE     MODC3      81/08/24   10.43
          OBJECT     MODA7      81/05/12   13.48
  EOG     GROUPMIX

                   BLOCKS REMAINING    DIRECTORY 000000 PRIME 000001 THIRD 000000  UNUSED 000000


.. COMMAND .........  BOG          LOADS,D1

000187  058          LOADS        BOG

.. COMMAND .........  RES          D2,L,MODC5

.. COMMAND .........  COP.U        D2,L,MODC8,D1

000187  098          MODC5000     LOD     81/08/24   10.54
000191  005          MODC7000     LOD     81/09/11   08.30
000198  164          MODC8000     LOD     00/00/00   00.08

.. COMMAND .........  EOG          LOADS,D1

000204  057          LOADS        EOG

.. COMMAND .........  COP          D1
```

Figure 3-7. Librarian Map for Building Module Groups (Part 2 of 3)

```
                                                                                PAGE # 0003
BLOCK   REC          NAME       TYPE     DATE       TIME      COMMENTS

                                        TABLE OF CONTENTS

        SOURCE     MODD1       80/08/08   16.34
        SOURCE     MODD2       80/08/08   16.35
        SOURCE     MODD3       80/08/08   16.37
        SOURCE     MODD4       80/08/08   16.39
        SOURCE     MODD5       80/08/08   16.41
        SOURCE     MODD6       80/08/08   16.43
        SOURCE     MODA1       81/04/27   09.14
        SOURCE     MODA2       81/04/27   09.22
        LOAD       MODA3000    81/05/06   11.10
        LOAD       MODA4000    81/05/08   10.44
        SOURCE     MODA5       81/05/08   11.06
        SOURCE     MODA6       81/05/12   13.45
        OBJECT     MODA7       81/05/12   13.48
        SOURCE     MODC1       81/08/14   13.35
        SOURCE     MODC2       81/08/21   10.51
        SOURCE     MODC3       81/08/24   10.43
        SOURCE     MODC4       81/08/24   10.47
        SOURCE     MODC6       81/09/01   10.28
        SOURCE     MODB1       81/05/12   13.59
        SOURCE     MODB2       81/06/01   12.33
        LOAD       MODB3000    81/06/01   12.50
        SOURCE     MODB4       81/06/05   12.45
        SOURCE     MODB5       81/06/29   12.38
        LOAD       MODB6000    81/07/06   14.52
        LOAD       MODB7000    81/07/10   14.14
        LOAD       MODB8000    81/08/14   13.31
        SOURCE     MODE1       80/08/08   16.44
        SOURCE     MODE2       80/08/08   16.45
        SOURCE     MODE3       80/08/08   16.45
        SOURCE     MODE4       80/08/08   16.47
BOG     LOADS
        LOAD       MODC5000    81/09/24   10.54
        LOAD       MODC7000    81/09/11   08.30
        LOAD       MODC3000    00/00/00   00.08
EOG     LOADS

                   BLOCKS REMAINING     DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


LIBRARIAN FINISHED
DATE 88/07/08  TIME 15.42
TOTAL NUMBER OF ERRORS 00000 UPSI SETTING X'00'
```

Figure 3-7. Librarian Map for Building Module Groups (Part 3 of 3)

3.6.4. Copying Cards to a Disk File

This job copies cards to a module in a disk file. In this case, the cards contain the sample job control stream explained in 3.6.2, except that the job is called SRTFLS. Figure 3-8 illustrates the librarian map for this job.

Job Control Stream

```
1.  // JOB SAVEDECK
2.  // DVC 20  // LFD PRNTR
3.  // DVC 50  // VOL D00410
4.  // LBL YOURSRC  // LFD YR
5.  // EXEC LIBS
6.  /$
7.           FIL   D1=YR
8.           ELE.D D1,S,SRTFLS,SORTS MODULES BY TYPE
9.  // JOB SRTFLS
10. // DVC 20  // LFD PRNTR
11. // DVC 50  // VOL D00410
12. // LBL ORIGINAL  // LFD RG
13. // DVC 50  // VOL D00410
14. // LBL ALLSRC  // LFD SC
15. // DVC 50  // VOL D00410
16. // LBL ALLOBJ  // LFD OB
17. // DVC 50  // VOL D00410
18. // LBL ALLLOD  // LFD LD
19. // DVC 50  // VOL D00410
20. // LBL ALLMAC  // LFD MC
21. // EXEC LIBS
22. /$
23.          FIL   D1=RG,D2=SC,D3=OB,D4=LD,D5=MC
24.          COP   D1
25.          COP   D1,S,,D2
26.          COP   D1,O,,D3
27.          COP   D1,L,,D4
28.          COP   D1,M,,D5
29. /*
30. /&
31.          EOD
32. /*
33. /&
34. // FIN
```

1. Identifies the job.
2. Assigns a printer to the job.
3. Identifies the logical unit number 50 for the disk volume D00410, which contains the file where the module is to be stored.

4. Declares the file YOURSRC and the logical file descriptor YR for the file where the module is to be stored.

5. Initiates execution of the librarian.

6. Indicates the start of the librarian control statements.

7. Assigns a type code and logical file number to the file YOURSC. Thus, D1 in line 8 refers to the file YOURSRC.

8. Indicates the beginning of the cards. It also supplies a name SRTFLS for the module once it is added to the disk file YOURSRC. The comment SORTS MODULES BY TYPE will be included in the module header record. The D option causes the records in the module to be listed on the librarian map.

9. through 30.
   Are the cards in the librarian control stream being saved.

31. Indicates the end of the cards.

32. Indicates the end of the library control statements for the job.

33. Indicates the end of job.

34. Ends card reader operation.

```
UNISYS OS/3 LIBRARIAN                                                    PAGE # 0001
DATE 88/07/08  TIME 15.53                                                VER820401

BLOCK   REC          NAME       TYPE    DATE       TIME      COMMENTS


.. COMMAND .........  FIL         D1=YR

                      D 1 - VSN IS D00410, LFD IS  YR      , FILE LABEL IS YOURSRC

.. COMMAND .........  ELE.D       D1,S,SRTFLS,SORTS MODULES BY TYPE

000001  005          SRTFLS      SOR    82/07/08   15.53     SORTS MODULES BY TYPE
000001  063          // JOB SRTFLS
000001  083          // DVC 20  // LFD PRNTR
000001  113          // DVC 50  // VOL D00410
000001  144          // LBL ORIGINAL  // LFD RG
000001  177          // DVC 50  // VOL D00410
000001  208          // LBL ALLSRC  // LFD SC
000002  005          // DVC 50  // VOL D00410
000002  036          // LBL ALLOBJ  // LFD OB
000002  067          // DVC 50  // VOL D00410
000002  098          // LBL ALLLOD  // LFD LD
000002  129          // DVC 50  // VOL D00410
000002  160          // LBL ALLMAC  // LFD MC
000002  191          // EXEC LIBS
000002  210          /&
000003  005                   FIL   D1=RG,D2=SC,D3=OB,D4=LD,D5=MC
000003  046                   COP   D1
000003  060                   COP   D1,S,,D2
000003  080                   COP   D1,O,,D3
000003  100                   COP   D1,L,,D4
000003  120                   COP   D1,M,,D5
000003  140          /*
000003  149          /&
```

Figure 3-8. Librarian Map for Copying Cards to a Disk File

3.6.5. Creating and Running a Librarian Job Interactively

The following sample session illustrates how to create a librarian job control stream interactively at a workstation, save it on disk, and then run the librarian job.

1.

```
 000000      SSSS          /       333
00000000    SSSSSS         //      33333
00    00    SS   SS       ///     33   33
00    00    SS            ///         33
00    00     SS          ///         33
00    00      SS        ///         333
00    00       SS      ///         33
00    00    SS   SS   ///         33   33
00000000    SSSSSS    //          3333333
 000000      SSSS     /             3333

          INTERACTIVE OPERATING SYSTEM
           DEPRESS TRANSMIT FOR LOGON
```

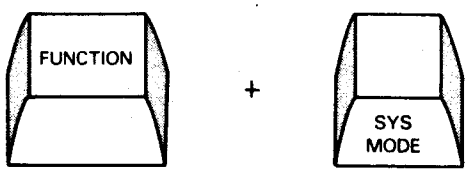
2a.

```
                OS/3 INTERACTIVE SERVICES
LOGON IDENTIFICATION:   USER-ID              >______<
                        ACCOUNT NUMBER       >____<
                        PASSWORD             >______<

OPTIONS:                EXECUTION PROFILE    >_______ <
                        BULLETIN             >YES <
                        LOG                  >YES <
                        NEW PASSWORD         >______<
```

2b.

```
LOGON USER1,BULLETIN=NO
 000000      SSSS          /       333
00000000    SSSSSS         //      33333
00    00    SS   SS       ///     33   33
00    00    SS            ///         33
00    00     SS          ///         33
00    00      SS        ///         333
00    00       SS      ///         33
00    00    SS   SS   ///         33   33
00000000    SSSSSS    //          3333333
 000000      SSSS     /             3333

          INTERACTIVE OPERATING SYSTEM
           DEPRESS TRANSMIT FOR LOGON
```

3.



4.
```
EDT
```

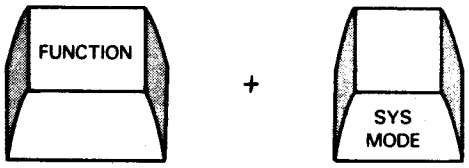
5.
```
OS/3 EDITOR READY (VERSION XXX)
1.0000// JOB SAVSRC
2.0000// DVC 20  // LFD PRNTR
3.0000// DVC 50  // VOL PUBDSK
4.0000// LBL ORIGINAL  // LFD RG
5.0000// DVC 50  // VOL PUBDSK
6.0000// LBL ALLSRC  // LFD SC
7.0000// DVC 50  // VOL PUBDSK
8.0000// EXEC LIBS
9.0000/$
10.0000         FIL   D1=RG,D2=SC
11.0000         COP   D1
12.0000         COP   D1,S,,D2
13.0000         COP   D2
14.0000/*
15.0000/&
```

6.
```
16.0000@WRITE MO=SAVSRC,TYPE=S,FIL=MYJOB,VSN=DSK001,SIZE=10,SAT=YES
```

7.
```
17.0000@DELETE
```

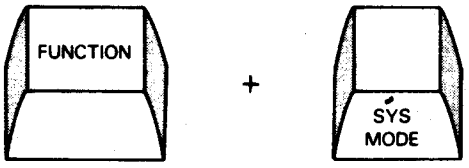
8.
```
18.0000@HALT
```

9.



10.
```
RV SAVSRC:(MYJOBS,DSK001)
```

11.



12.
```
LOGOFF
```

13.
```
IS73 LOGOFF ACCEPTED AT 14:34:00 ON 88/09/12
```

1. Start with the idle workstation display.
2. Log on at the workstation in either of two ways:
   a. Press the XMIT key to display the logon screen, fill in the entries, and press the XMIT key again.
   b. Key in the LOGON command and press the XMIT key.
3. Place workstation in system mode. Hold down the FUNCTION key and, while holding it down, press the SYS MODE key.
4. Activate the general editor (EDT). Key in EDT on the top line of the screen and press the XMIT key.
5. The first workspace line number is displayed. Key in the job control stream.
6. Key in an EDT WRITE command to save the job control stream as a source module in a SAT file. Here, the module name is SAVSRC; its type is S for source. SAT=YES saves it in a SAT file. The file is named MYJOBS and is located on the disk volume named DSK001. The SIZE parameter allocates 10 cylinders for the new file. If the file MYJOBS already exists on DSK001, the SIZE parameter must be omitted.
7. After the module is saved, key in an @DELETE command to clear the workspace.
8. Key in an @HALT command to terminate EDT. The workstation returns to workstation mode.
9. Place the workstation in system mode. Press the FUNCTION key and, while holding it down, press the SYS MODE key.
10. Key in RV command to run the job. and press the XMIT key. The workstation returns to workstation mode.
11. When a job completion message is displayed, place the workstation in system mode. Press the FUNCTION key and, while holding it down, press the SYS MODE key.
12. Log off of the workstation. Key in the LOGOFF command and press the XMIT key.
13. A successful logoff message is displayed.

Section 4
Functional Characteristics

4.1. Introduction

The OS/3 linkage editor is a multiphase load module that resides in the system load library file (YLOD). It relies on allocation of both system disk and main storage free space for intermediate processing efficiency as well as one or more system or user object libraries for obtaining the desired object code. The load modules the linkage editor produces subsequently can be stored in either the system load library file or a user load library file. All reentrant load modules must be stored in the system load library file (YLOD). The system access technique (SAT) is used in the management of all files accessed by the linkage editor. Figure 4-1 illustrates the functional relationships between the linkage editor, SAT, and the various interfacing files.

The linkage editor relies on an input control stream for acquisition of control statement directives and parameter information. In the absence of such a control stream, the linkage editor defaults to the appropriate system job run library (YRUN) file to obtain the modules to be used in constructing the load modules. The program name of the linkage editor is LNKEDT.

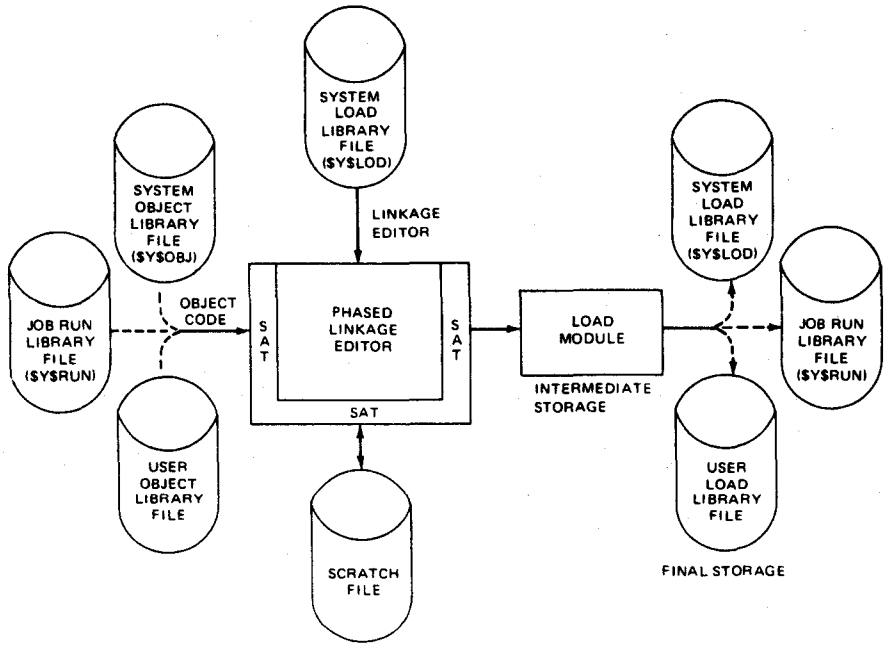


Figure 4-1. Functional Relationship between the Linkage Editor, SAT, and Related Files

4.1.1. The SAT Interface

The linkage editor uses the system access technique for the following purposes:

- Reading system and user libraries to obtain the necessary object modules to be included in the load module.
- Managing an intermediate scratch file during the link-edit process.
- Creating the final output load module or modules.

Because the linkage editor is producing standard load modules (as needed by the system) and is accessing standard object modules (as produced by the various language processors), all of which are contained in disk program library files, the SAT file definitions (DTFs) describe three partitions for each file. The first partition is the directory partition used as an index into the second and third partitions, the data partitions of the library file. Because the output of the linkage editor is essentially a library file (load module type), it too contains both a directory partition and two data partitions (the directory is composed primarily of phase header pointers and the third partition is empty).

The load modules produced by the linkage editor contain phase header records (one for each phase segment); text/RLD records comprising the actual instructions, constants, and relocation information making up each individual phase; transfer records (one at the conclusion of each phase); ISD records and, optionally, shared code records. Each phase header contains an appropriate entry in the load library directory (first partition), which identifies the load module and the particular phase and points to the relative position within the data body of the library file (second partition). For each phase header entered in the body of the load module (second partition), the necessary directory entries also are entered in the directory index (first partition) with the relative pointer obtained from the necessary DTF description for the second partition. Both the directory and prime data areas of the file contain blocked logical records.

The object modules that the linkage editor uses in its construction of the load module are also retrieved through the SAT. These files are accessed through a file directory scan (first partition) in an attempt to locate the necessary module header, control section definition, or entry symbol definition. (Object library file directories are supplemented with named CSECTs, COMs, and ENTRYs to facilitate this search.) When the desired directory item is found, the relative directory pointer is extracted from the record and copied into the second partition definition of the DTF to obtain the needed modules or control sections from the body of the module.

The intermediate scratch file used by the linkage editor in the course of the link-edit job and seen only by the linkage editor for its processing, also is accessed via SAT. This file is scratched when the link-edit job is finished.

4.1.2. Temporary Storage Usage

The linkage editor uses open storage space at the end of its longest path for the management of temporary reference tables and buffers. This area is reserved by the linkage editor when it is first loaded into main storage. In a minimum storage system, the reserved area is always equal to the size of the maximum storage partition allowed any user program. In larger system configurations, this reserved area is dynamically expanded at linkage editor execution time.

If any job step in a job containing a linkage editor job step requires an amount of storage in excess of the minimum linkage editor storage requirements, the linkage editor uses the excess for reference table and buffer expansion. Because job control assigns storage requirements for a given job based on the largest requirement for any job step, when not specified explicitly on the // JOB control statement, the linkage editor adjusts its pointers (as originally established) to accommodate the availability of the extra tabling space (if any extra exists). The linkage editor determines the amount of storage space allocated for the job (of which it is a step) by examining the job prologue. The speed of the link-edit is proportional to the amount of main storage space available to the linkage editor.

If a given link-edit job involves a number of definitions with table entries exceeding the current internal storage space available for that job step, the linkage editor expands the reference table by using data blocks on the temporary output medium via the DTF partition assigned to the intermediate file. Thus, table overflow does not cause the link-edit job to be aborted.

4.2. Linkage Editor Inputs and Outputs

The linkage editor uses as input both job control stream data in the form of linkage editor control statements and object module elements and produces, as output, one or more load modules and a link-edit map for each load module (see Figure 4-2). Under normal operating conditions (no normal linkage editor options suppressed or superseded), the linkage editor uses the control statements contained in the job control stream to construct one or more load modules per job. These statements, which comprise the primary control stream input to the linkage editor, specify the object modules to be included in the load modules, the structure of the load modules to be produced, and the linkage editor options to be in effect during construction of the load modules.

The object modules referenced in the primary control stream also may have linkage editor control statements embedded in them. When they do, these control statements are effectively inserted in the primary control stream at the point the object module containing them was referenced, and thus become part of the primary control stream input.

The control statements contained in the primary control stream input also may identify source modules containing linkage editor control statements to be processed during the job step. When such statements are processed, the control statements they reference also are effectively inserted in the primary control stream input just like embedded control statements. These statements, however, have special processing significance attached to them and are thus referred to as source module control stream statements in this guide to differentiate them from the other two types of primary control stream statements (job control data and embedded control statements).

The load modules produced by the linkage editor normally are output to the system job run library file (YRUN). The user, however, may direct his output to be stored in any library file he wishes, or he may elect to suppress this output entirely through the linkage editor control statement options available to him. The same is true for the link-edit map, which normally is output by the linkage editor for each load module it produces, whether or not the load module output is suppressed. This output also can be suppressed by the user through a linkage editor control statement option. Each link-edit map describes the control statements used to construct its associated load module, the symbols, or labels, detected in the object modules included in the load module, the structure of the load module produced, and any processing errors that might have been detected during construction of the load module. Detailed descriptions and illustrations of the map components are presented in Section 7.

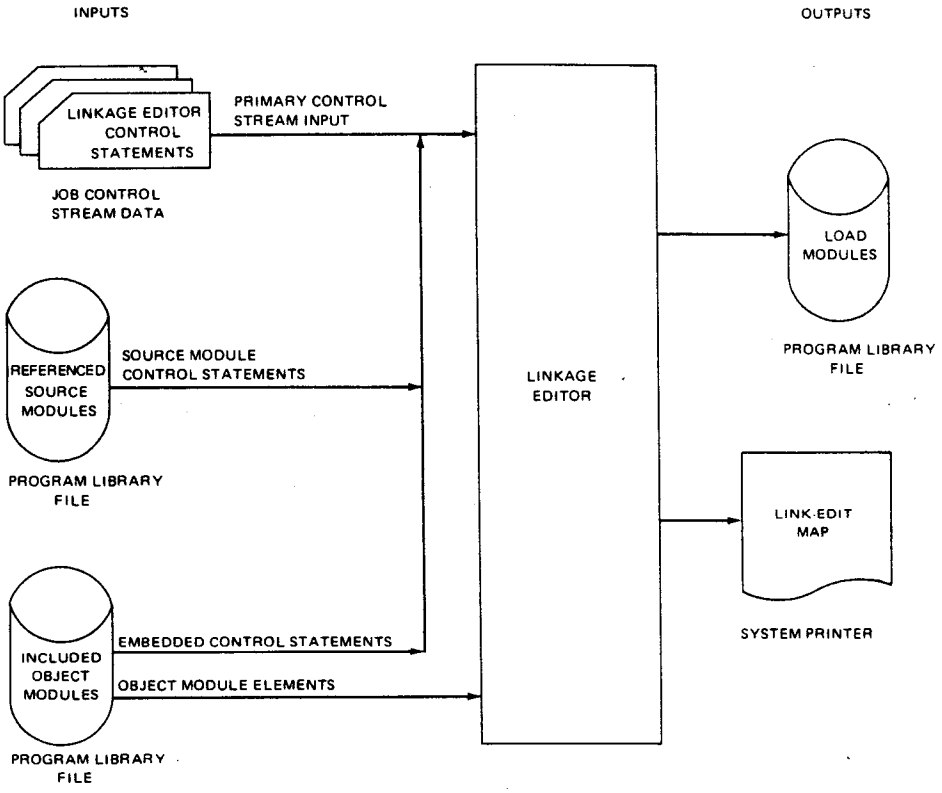


Figure 4-2. Linkage Editor Inputs and Outputs

4.3. Control Statement Functions

Control statements are available to the user to direct the linkage editor in construction of a load module. These statements allow the user to generate a load module that is structured in accordance with his program requirements, as these requirements exist in the object elements that are to make up the load module. Linkage editor control statements normally appear as data in a job control stream following an execute linkage editor job control statement (// EXEC LNKEDT). As previously indicated, however, linkage editor control statements also can be embedded in object modules or contained in source modules referenced by linkage editor control statements. Therefore, if no control statements are present in the job control stream following a // EXEC LNKEDT control statement, the linkage editor constructs a load module by using all the object modules currently contained in the job's job run library file (YRUN). If none of the object modules contained in this file contain embedded linkage editor control statements, the linkage editor constructs only one nonreentrant single-phase load module, named LNKLOD, consisting of all the object modules contained in the YRUN library file; otherwise, if so directed by embedded control statements, the linkage editor may produce multiple load modules, reentrant or otherwise, with or without multiphase and multiregion structures.

The name and basic function of each of the linkage editor control statements are described in the following paragraphs. A more detailed description of these control statements is presented in Section 6.

- *// PARAM and LINKOP Statements.* These statements specify the linkage editor processing options that are to be in effect during construction of a load module. These options include:
  - Determining file name assumption to be used by the INCLUDE mechanisms of the linkage editor when such file names are not explicitly designated by the user
  - Disallowing the automatic inclusion of object modules in the load module by the linkage editor
  - Disallowing the automatic overlay mechanism of the linkage editor from being included in the load module
  - Disallowing the promotion of common storage sections by the linkage editor
  - Selecting the output file in which the load module created by the linkage editor is to be stored
  - Terminating a link-edit job if a processing error is detected
  - Selecting the components of the link-edit map to be output for a given link-edit job
  - Inserting comments in the phase header records produced by the linkage editor

- Building reentrant load modules
- Ignoring the reentrancy of object modules
- Suppressing the production of ISD records

- *LOADM - Begin Load Module.* This statement requests the linkage editor to begin construction of an executable load module. This statement initiates the creation of the start of the root phase segment and specifies a name for the load module. It normally is the first control statement in each link-edit job.

- *INCLUDE - Include Object Code.* This control statement instructs the linkage editor to include an entire object module or specific object module sections in the load module being constructed. This statement specifies the name of the object module and module sections, if applicable, that are required to be in the load module segment currently under construction. It may also identify the file in which the specified module is located.

- *OVERLAY - Begin Overlay Phase.* This statement directs the linkage editor to begin construction of a new load module phase separate from the initial phase and any other previously defined phase. Any and all INCLUDE requests for object code following an OVERLAY statement are included in the phase initiated by the OVERLAY statement up until the detection of another OVERLAY or REGION control statement or termination of the immediate link-edit job. The phase name assigned to the overlay phase is a combination of the name assigned to the load module plus a two-digit number from 00 to 99, which indicates the order in which the various phases of a load module were declared to the linkage editor. A unique six-character alias-phasename also may be assigned to each overlay phase.

- *REGION - Begin New Region.* This statement causes a new phase to be created in a new region of the load module. This statement has all the attributes of the OVERLAY statement, and, in addition, initiates the construction of a new load module region.

- *ENTER - Define Phase Execution Entrance.* This statement specifies the start-of-execution point for the phase currently under construction in a load module. This is the point to which control is optionally transferred once the phase has been loaded in main storage. The transfer point is optionally assigned by the linkage editor if no such statement is supplied for a phase. This statement is normally the last specified for each phase defined in a load module.

- *EQU - Define Label.* This statement is used to define an otherwise undefined label in a load module. The normal method of defining and satisfying cross-references by the linkage editor is via the proper INCLUDE directives and external symbol dictionary (ESD) records in object code included in the load module. This statement, however, allows the user to equate two symbols, a symbol and a value, or a value only, that could not otherwise be resolved in the link-edit run. If one symbol is being equated to another, the equating symbol must have been previously defined.

- *MOD - Modify Location Counter.* This statement is used to modify the current program counter that the linkage editor maintains during construction of a load module. This statement permits the user to accomplish boundary adjustments at link-edit time outside the realm of the makeup of modules and code being included.

- *RES - Reserve Storage.* This statement directs the linkage editor to reserve additional storage at the end of the longest path in the load module being constructed. The additional storage requested is included in the overall length of the load module as recorded in the load module header record, and may be referenced by the object code included in the load module.

4.4. Object Module Format

The format of the object modules produced by all the language processors and used as input by the linkage editor is illustrated in Figure 4-3. As shown, an object module consists of:

- Object module header record that uniquely identifies the object module.

- One or more control section records, each of which defines the symbolic name, external symbol identification (ESID), and length of each control section (CSECT and COM) comprising the object module.

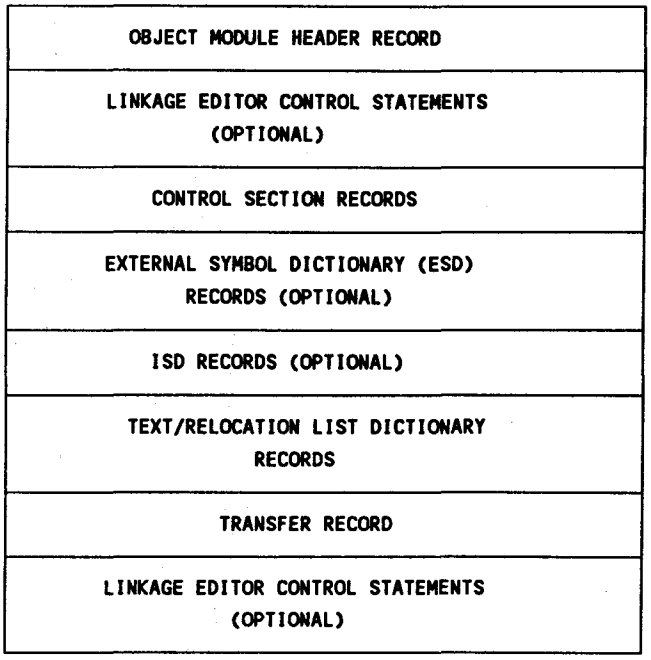
- Possibly one or more external symbol dictionary (ESD) records, each of which is used by the linkage editor in satisfying cross-references (ENTRY, EXTRN, and V-CON references) between object modules during the link-edit operation.

- Possibly one or more internal symbol dictionary (ISD) records used to describe the internal symbols of the user program.

- One or more text and relocation list dictionary (RLD) records containing the data and instructions making up the object code, together with relocation masks used by the linkage editor to modify specific areas of the object code at link-edit time.

- Transfer (TRF) record that may optionally indicate a start-of-execution address for the object module. The transfer record is normally the last record in any set of relocatable object code produced by a language processor.

- One or more linkage editor control statements that may be embedded in an object module. Embedded control statements may appear only immediately following the object module header record or transfer record (see Figure 4-3).



Note:

Control section and ESD type records must have unique names within a given object module. For example, an entry point and a CSECT must not have the same name and exist in the same object module.

Figure 4-3. OS/3 Object Module Format

4.5. Load Module Format

The format of the load modules produced by the linkage editor, as they are stored in a library file, is illustrated in Figure 4-4. As shown, all load modules consist of at least one phase called the root phase. Multiphase and multiregion load modules consist of a root phase plus one or more additional phases. (A root phase is required in every load module because it contains the information necessary to allocate the load module space in main storage prior to its execution by the system.) The number of phases and regions comprising a load module is specified by the user in the linkage editor control stream that produces the load module. A load module may consist of up to 100 phases and up to 10 regions.

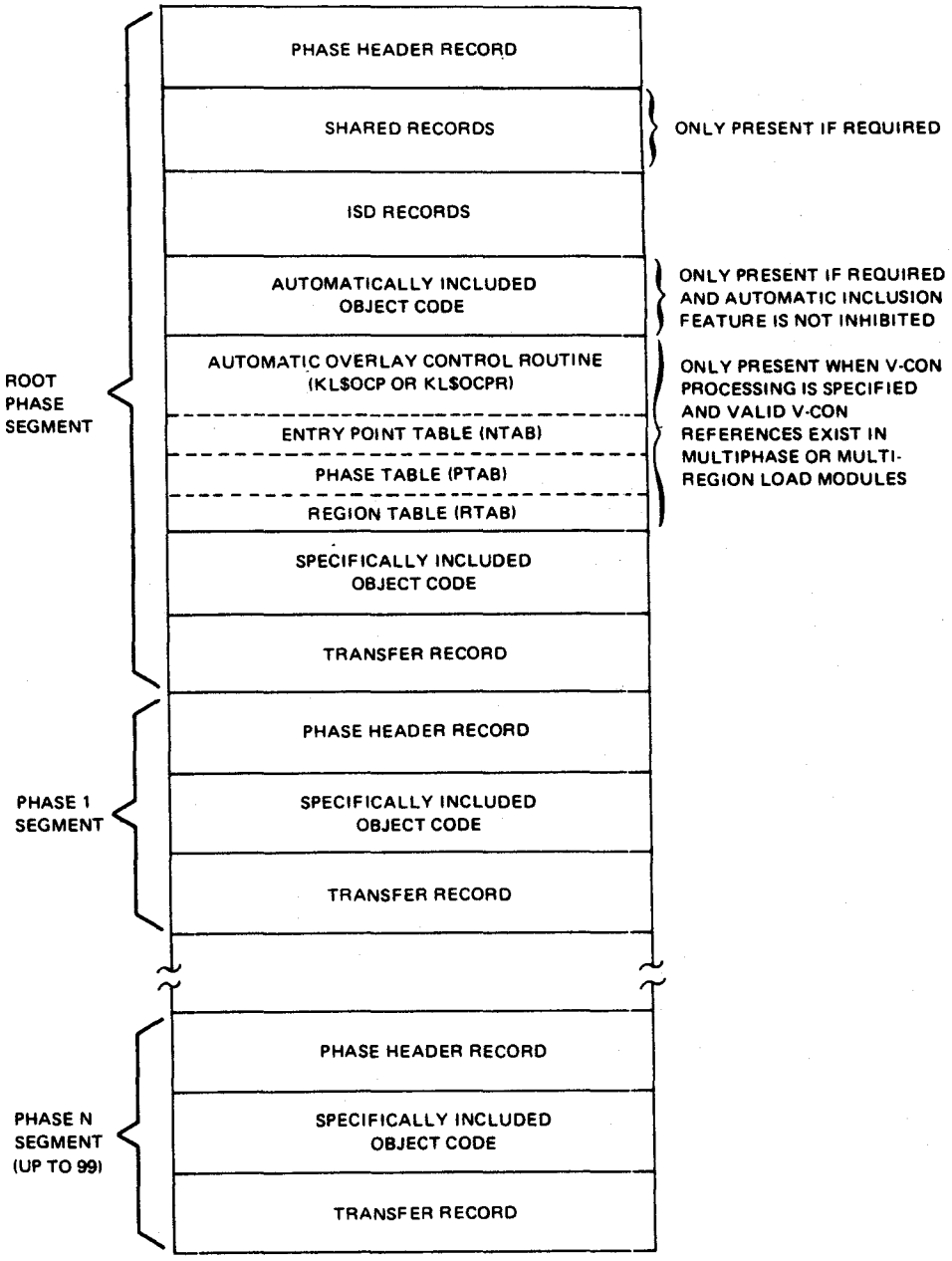


Figure 4-4. OS/3 Load Module Format

Each phase of a load module consists of at least the following:

- A phase header record that identifies:
  - Name of the phase
  - Size of the phase in bytes and its origin
  - Amount of main storage required to load the longest path of the load module
- At least one (and probably more) text/relocation (RLD) record that includes both data and instructions comprising the load module phase and possible relocation masks to be used to relocate the text at execution time.
- A transfer record identifying the end of the phase, and containing a transfer address at which point phase execution is to optionally begin.

In addition to these records, the root phase of a load module may contain:

- Any number of shared records. A reentrant load module will have one or more SENTRY records indicating the availability of ENTRY points in that module. A nonreentrant load module will not have any shared records unless it references reentrant code. In this case, it will have a resource record and a SEXTRN record for each reference that was resolved to a reentrant object module.
  - Resource records contain the name of the reentrant module referenced.
  - SEXTRN records identify the EXTRNs that were resolved to the resource.
- Any number of ISD records. The ISD records can be one of the following:
  - CSECT/COM ISD records identifying control and common sections included by the linkage editor.
  - ISD records produced as a result of being generated by any of the language processors within object modules.
- Any automatically included object module sections that are needed to satisfy any cross-references not defined in any specifically included object module section.
- An automatic overlay control routine if the load module produced requires the automatic loading of its phases. Two versions of this routine exist: one handles single-region structures (SL$OCP) and the other handles multiregion structures (KL$OCPR).
- An entry point table (NTAB), used in conjunction with the automatic overlay control routine, to provide for automatic loading of phases. This table contains the addresses of referenced ENTRY points and an index into the phase table (PTAB) which identifies the phases containing the ENTRY points.

- A phase table (PTAB), used in conjunction with the automatic overlay control routine, to provide for automatic loading of phases. This table contains the overlay structure information required to load the various phases of the load module.

- A region table (RTAB), used in conjunction with the automatic overlay control routine, to control automatic loading of the various regions that may be in the load module. This table describes the region structure of the load module and is present only if the KL$OCPR automatic overlay control routine, with multiregion control, is present.

When a load module is loaded in main storage for program execution, only its instructions and data are loaded; phase header, shared records, ISD records, and transfer records are not included. These records are used only to:

- Relate the amount of main storage required to store the load module and the particular phase being loaded, including any common storage areas and reserved storage areas required by the load module program

- Identify the phase load address

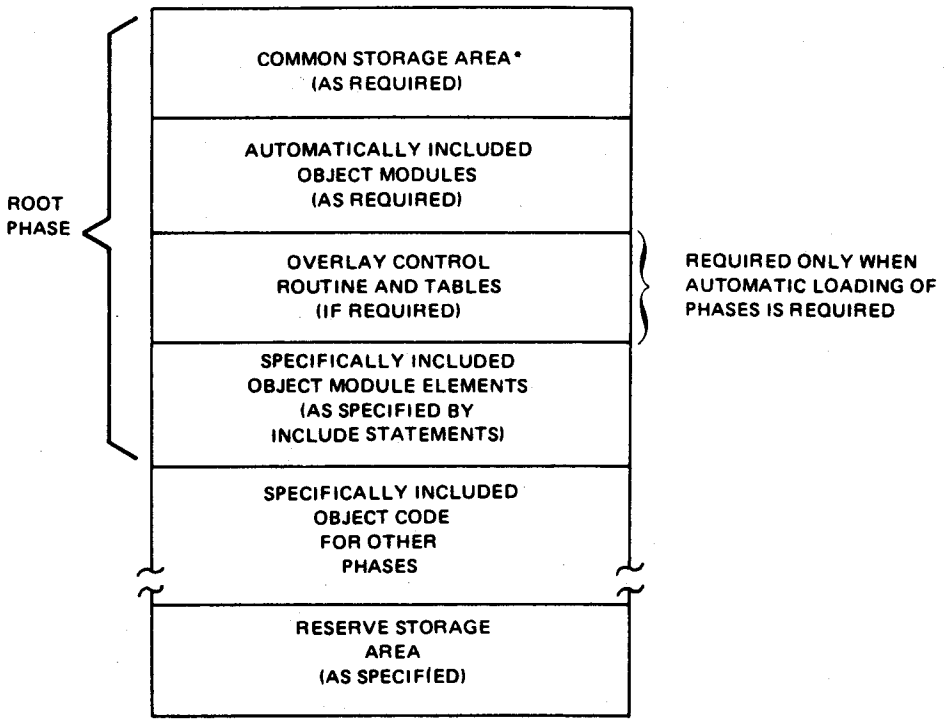
- Identify the program starting address

- Identify the resources (reentrant load modules) that are needed and SEXTRNs to be satisfied at execution time

- Identify the entry points that are available in reentrant load modules at execution time

- Give JOBDUMP the necessary information to print a dump of segmented CSECTs of the program and internal symbols of the user program

Figure 4-5 illustrates the format of a typical load module as it might appear in main storage. As shown, it might have a common storage area and a reserve storage area. These areas are storage areas that the linkage editor reserves for the load module program in accordance with the common storage area definitions contained in the object module code included in the load module and the reserve storage (RES) control statements contained in the linkage editor control stream that created the load module. The size of the common storage area reserved for a load module is the sum of all the common storage areas requested and described by both the specifically included and automatically included object modules in the load module. Common storage areas may be labeled or unlabeled in the object module code. The size of the reserve storage area reserved for a load module is the sum of all the reserve storage areas specified by the user in all RES control statements.



*Common storage areas also may be promoted to specific phases other than the root phase by the linkage editor (see 4.7.3).

Figure 4-5. Typical Load Module Format when Loaded in Main Storage

4.6. Load Module Structure

The structure of a load module is that form a load module takes when it is loaded in main storage by the program loader routine of the supervisor. This structure is defined by the phase and region data contained in the load module as placed there by the linkage editor in accordance with the control stream directives by the user. Three basic load module structures are capable of being constructed by the linkage editor:

- Single-phase load modules
- Multiphase load modules
- Multiregion load modules

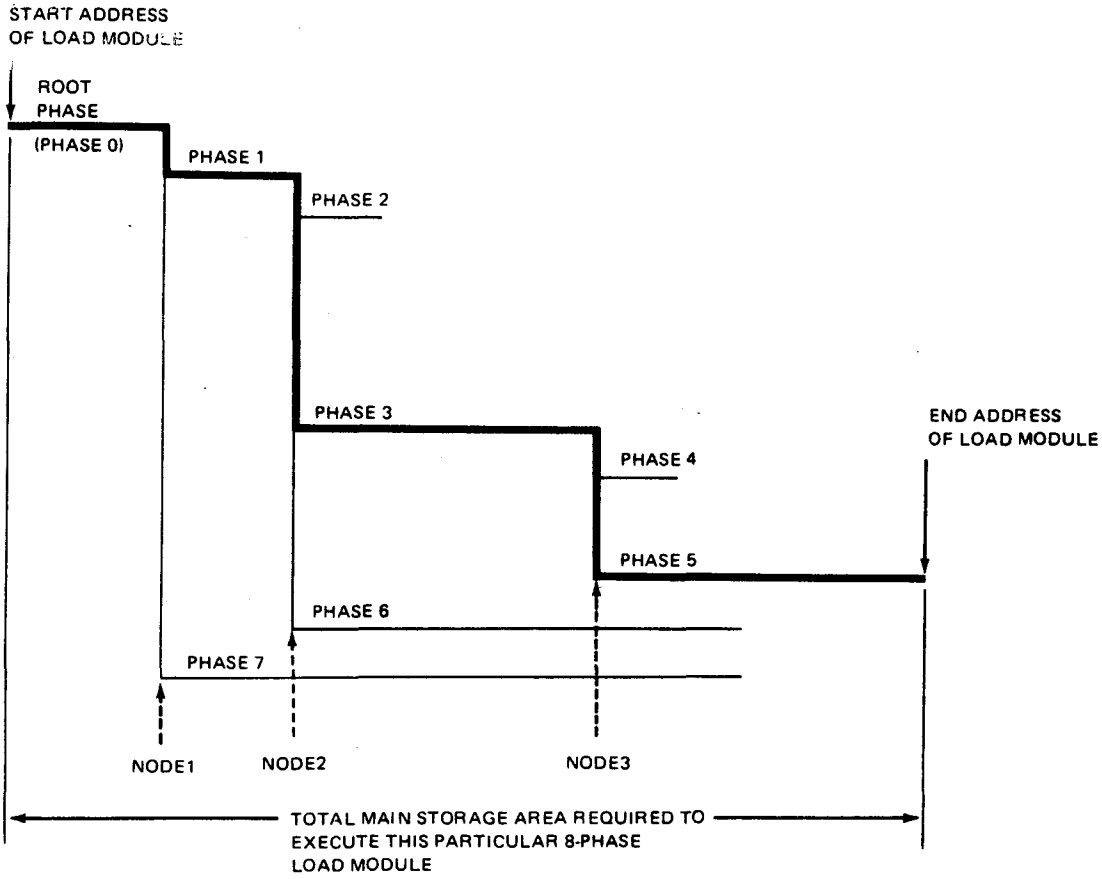
4.6.1. Single-Phase Load Modules

The storage structure of a single-phase load module can be represented by a single horizontal line whose length is relative to the amount of serial storage locations required to store the load module in main storage. All load modules generated by the linkage editor start out as single-phase load modules and are created only as multiphase or multiregion load modules if so directed through OVERLAY and REGION control statements in the input control stream. Thus, single-phase load modules are modules that consist solely of a root phase; multiphase and multiregion load modules consist of a root phase plus one or more additional phases. Reentrant load modules must be single-phase load modules.

4.6.2. Multiphase Load Modules

Multiphase load modules are constructed by a programmer to minimize the main storage requirements of a program. Multiphase load modules contain multiple phases, which can overlay each other in main storage. The structure of a typical multiphase load module is illustrated in Figure 4-6. As shown, multiphase load modules are represented by multiple horizontal lines, each of which represents a particular program phase and its relative length. The main storage location at which a phase is loaded is called a node point. Node points are shown as vertical lines whose lengths have no significance. All the phases in a multiphase load module excluding the root phase are loaded in main storage at a node point. Node points and phases are defined by the user through the linkage editor OVERLAY and REGION control statements.

The INCLUDE statements following an OVERLAY or REGION control statement identify the object module elements that are to comprise the phase. Ignoring the root phase, the number of phases in a multiphase load module coincides with the number of OVERLAY and REGION control statements present in the control stream that caused its generation. The root phase of all load modules is initiated with the initiation of the load module, normally in response to a LOADM control statement.

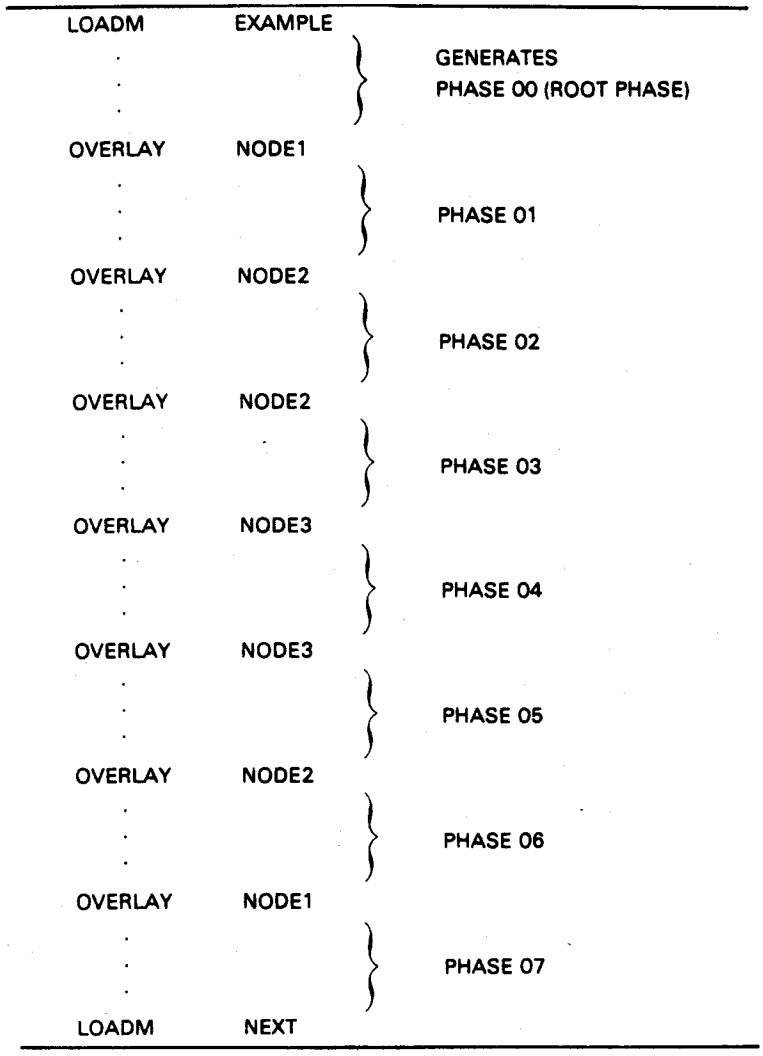


NOTES:

1. Heavy lines indicate an example of a path.
2. The end address of the load module is the last addressable location of the longest path in the load module. This address is always defined and may be declared as an external reference (EXTRN) in any object module included in the program. The entry name assigned this end address is KE$ALP.

Figure 4-6. Typical Multiphase Load Module Structure

The OVERLAY control statements required to produce the multiphase load module shown in Figure 4-6 are illustrated in Figure 4-7.



Note: The ellipses represent INCLUDE statements for object modules to be included in a particular phase. These statements must follow the proper LOADM or OVERLAY statement.

Figure 4-7. Typical Multiphase Load Module Control Stream

Phase Definitions

Phases are defined by the user beginning with an OVERLAY or REGION control statement (root phases excepted), and are usually followed by one or more INCLUDE control statements that identify the object module elements that are to comprise the phase. A load module phase may be thought of as a program segment that can

perform one or more specific processing tasks. The order in which phases are defined by the user does not dictate the order in which the phases in a load module must be executed; the logic of the program determines the sequence of execution of the phases contained in a load module and, likewise, the number of times any specific phase will be loaded and executed. In some programs, not all phases are executed each time the program is executed, again, depending on the logic of the program.

Thus, the order in which phases are defined, which is also the order in which they are numbered by the linkage editor, does not necessarily have any bearing on the order in which the phases of a load module are executed.

The node point assigned to each phase usually does, however, determine where the phase will be loaded in main storage. Thus, a program's logic must be considered before one phase of a load module is assigned the same origin as another phase in the load module, because these phases never can be in main storage simultaneously.

Phase Names

Phase names are used to identify the various phases of a load module when they are to be loaded in main storage for program execution. Programmers who wish to do their own program loading, rather than have the automatic overlay-region control mechanism of the linkage editor embedded in their load module, must reference a phase name in a FETCH or LOAD macro instruction whenever a phase is to be loaded for execution.

The linkage editor automatically assigns a name to each phase of a multiphase load module based on the name assigned to the load module either by the programmer or the linkage editor. Phase names consist of eight alphanumeric characters. The first six characters are the same as the load module name and the last two characters are the decimal number of the phase (00 through 99). Phase numbers are assigned to each phase consecutively, in the order in which the phases are defined by OVERLAY and REGION control statements.

An alias phase name also may be assigned to each phase by the programmer through the OVERLAY or REGION control statement that causes its generation. The assignment of alias phase names allows the programmer to reference the phases of a load module in his subroutines or phrases, without knowing the order in which the phases will be defined in the linkage editor control stream. The linkage editor overlay control mechanism always refers to the linkage-editor-generated phase names.

Node Points and Paths

The starting address of each phase in a load module is called a node point. Node points are defined by the user as a symbol in the operand field of the OVERLAY or REGION control statements that are used to define each phase. The node point of the root phase is the name assigned to the load module. The starting address of one phase is normally the terminating address of a previous phase (it also could be a relative definition in the current path). The same node point may be the assigned origin of more than one phase in a load module; however, when an OVERLAY statement that

refers to a node point previously defined is detected by the linkage editor, all intervening node points are eliminated. For example, in Figures 4-6 and 4-7, once phase 6 is defined, no additional phases may be defined starting at NODE3. Also, once phase 7 is defined, no additional phases may be defined starting at NODE2. If phase 7 were followed by an OVERLAY NODE2 statement, the linkage editor would construct a new NODE2 node point at the end of phase 7 for the new phase being defined.

References to a given label in the load module may be traced along a path from one phase back to the root phase. Reference x is said to be on the path of phase y if they are in the same phase, or if they are on the same path and reference x is closer to the root phase than phase y. The root phase is usually on the path of every other phase in a load module, except when it is overlaid by a subsequent phase. In Figure 4-6, phases 0, 1, 3, and 5 are on the path of phase 5, as indicated by the heavy line; however, phase 5 is not on the path of phases 0, 1, or 3. Phase 1 is not on the path of phase 7, and phase 7 is not on the path of phase 1. The maximum number of phases allowed on any path is 14.

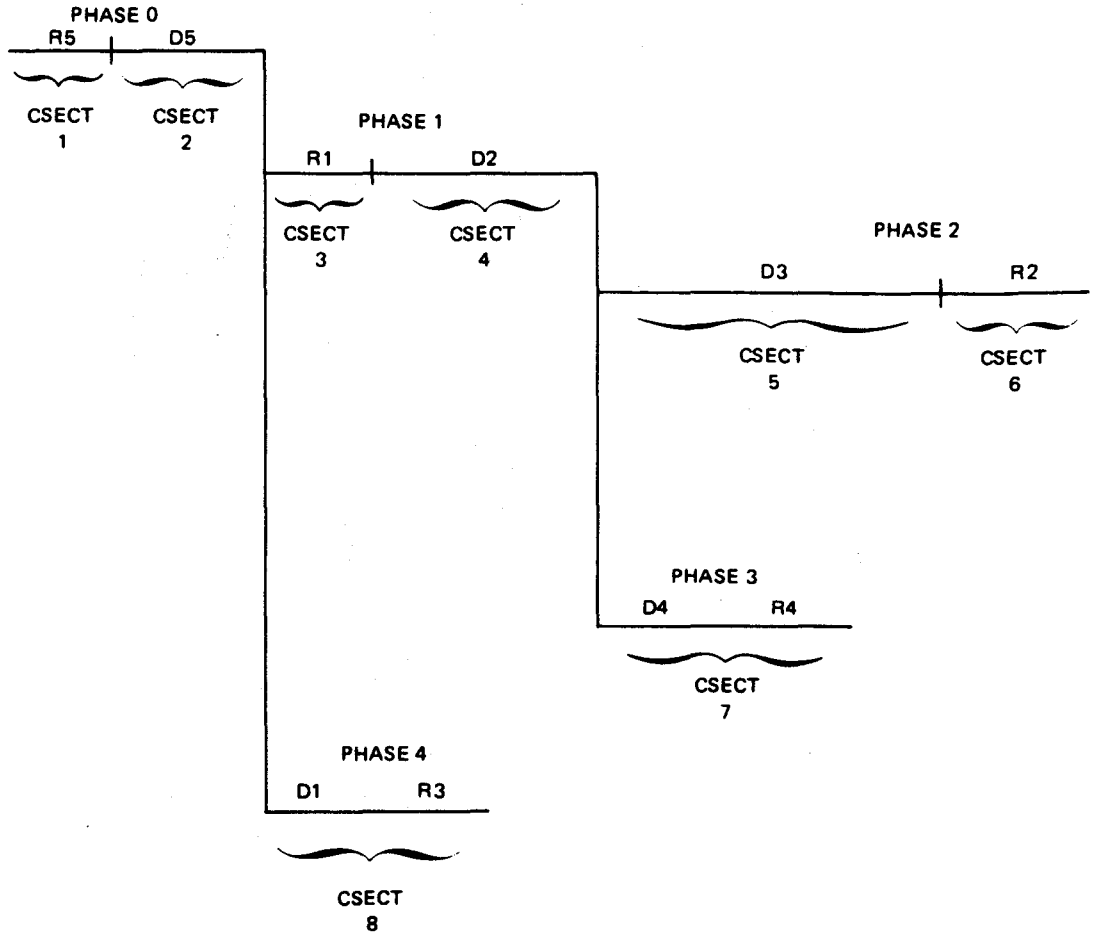
The path concept of a load module must be understood before the user can appreciate how the linkage editor satisfies references between various load module phases.

Communications between Phases

Communications between phases is accomplished by an external reference in one phase whose name matches a definition (CSECT or ENTRY point) in another phase. The language processors may produce two types of external references. The first is a standard (type A) address constant which, at execution time, will contain the linkage-editor-assigned address of the associated definition. The second is a type V address constant, which, in effect, requests that an automatic load mechanism ensure that the definition being referenced is resident at the time the reference is made. A linkage editor option allows the user to convert, at link-edit time, all type V EXTRNs to standard address constants.

The phase structure of a particular load module determines another characteristic of a given reference. If the definition to which a particular EXTRN refers is on the path of the reference, the reference is said to be inclusive. All inclusive references are treated as standard address constants by the linkage editor regardless of the EXTRN type generated by the language processor. Conversely, if the definition that satisifies a particular reference is not on the path of the reference, the reference is said to be exclusive. It is these exclusive references that can be controlled by the user through the linkage editor (V/NOV) option. Figure 4-8 illustrates both inclusive and exclusive references.

If a type V exclusive reference is made, the linkage editor puts the address of the definition in a table it generates (NTAB) in the root phase of the load module. The reference address is then made to point to an automatic overlay control routine. This control routine ensures that the required path is loaded before transferring control to the required definition. All other references require that the user ensure that the required definition is loaded, when referenced, through the supervisor FETCH and LOAD macroinstructions contained in the text of his program.



LEGEND:

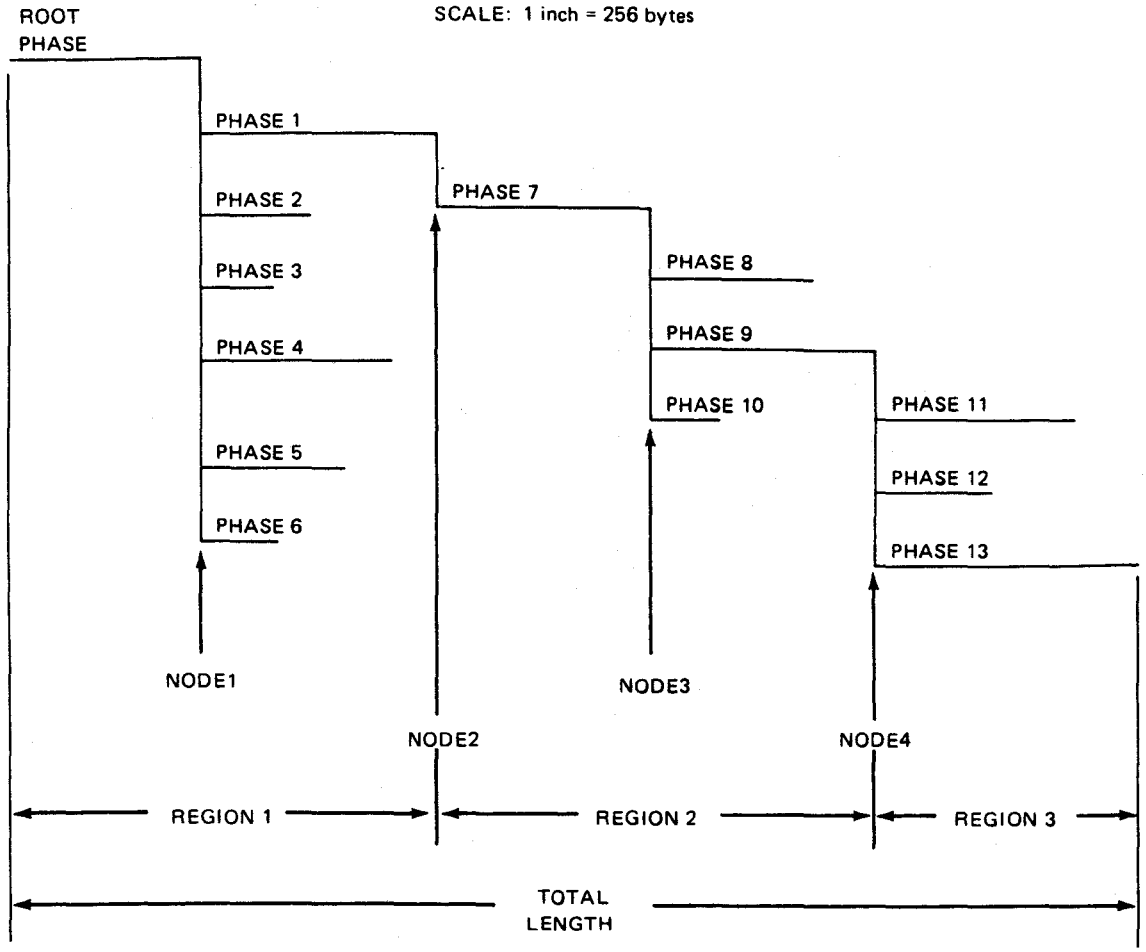
D Definition
R Reference

Inclusive References	Exclusive References
R2 to D2, D3, or D5	R1 to D1, D3, or D4
R1 to D2 or D5	R2 to D1 or D4
R3 to D1 or D5	R3 to D2, D3, or D4
R4 to D2, D4, or D5	R4 to D1 or D3
R5 to D5	R5 to D1, D2, D3, or D4

Figure 4-8. Examples of Inclusive and Exclusive References

4.6.3. Multiregion Load Modules

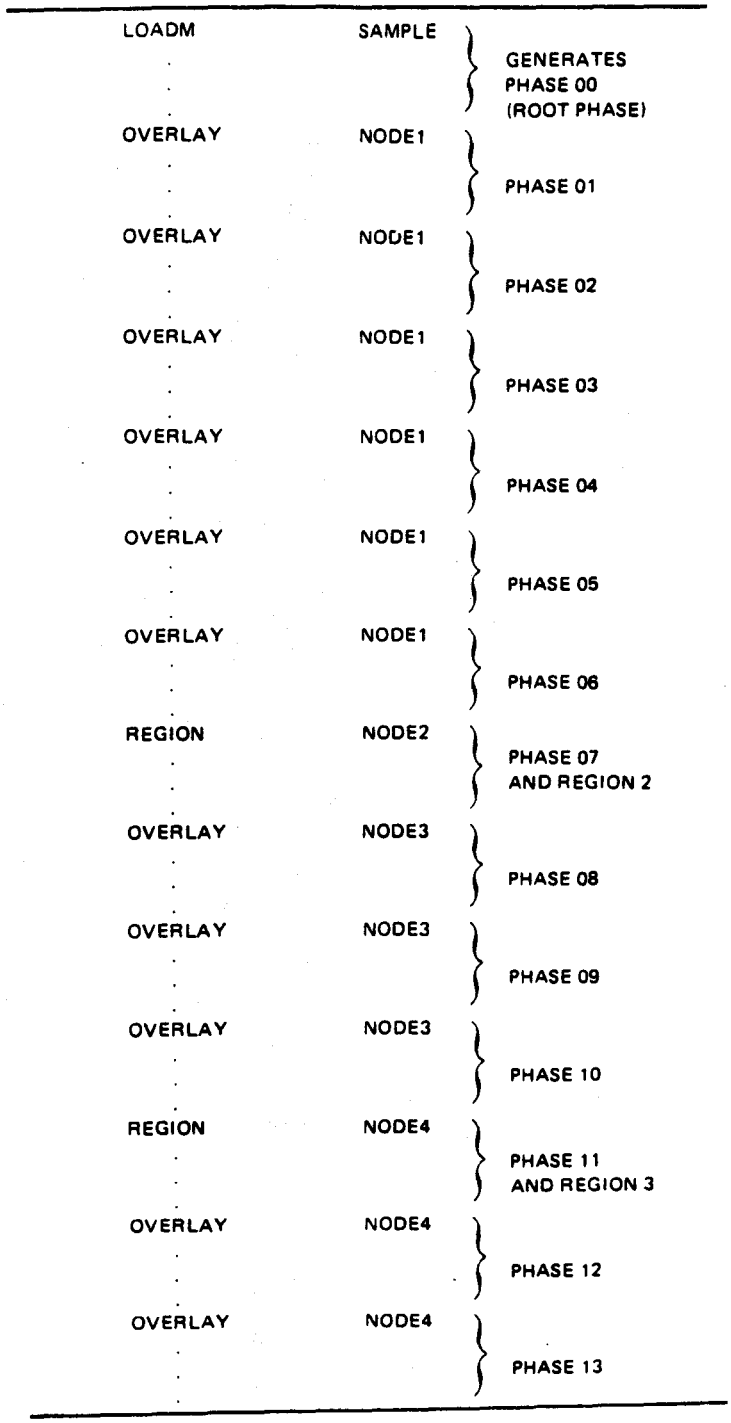
Multiregion load modules are basically the same as multiphase load modules, except that a multiregion load module is constructed so that the origin of the first phase of each region is at the end of the longest path defined in the previous region, rather than at the end of the phase previously defined. This feature prevents the phases in one region from overlaying any portion of a phase in any other region. The structure of a typical multiregion load module is illustrated in Figure 4-9. As shown, the load module consists of 14 phases divided into three regions. The OVERLAY and REGION control statements required to produce this structure are illustrated in Figure 4-10. If this same load module were defined by using only OVERLAY control statements, its structure would be as illustrated in Figure 4-11.



NOTES:

1. The origin of each region is the end of the longest path of the previous region.
2. The root phase is in the path of every region.
3. Up to 10 regions are permitted per load module.

Figure 4-9. Program SAMPLE as a Multiregion Load Module



NOTE:

The ellipses represent INCLUDE statements for object modules to be included in a particular phase. These statements must follow the LOADM, OVERLAY, and REGION control statements.

Figure 4-10. Control Stream Coding Required to Construct the Multiregion Load Module SAMPLE

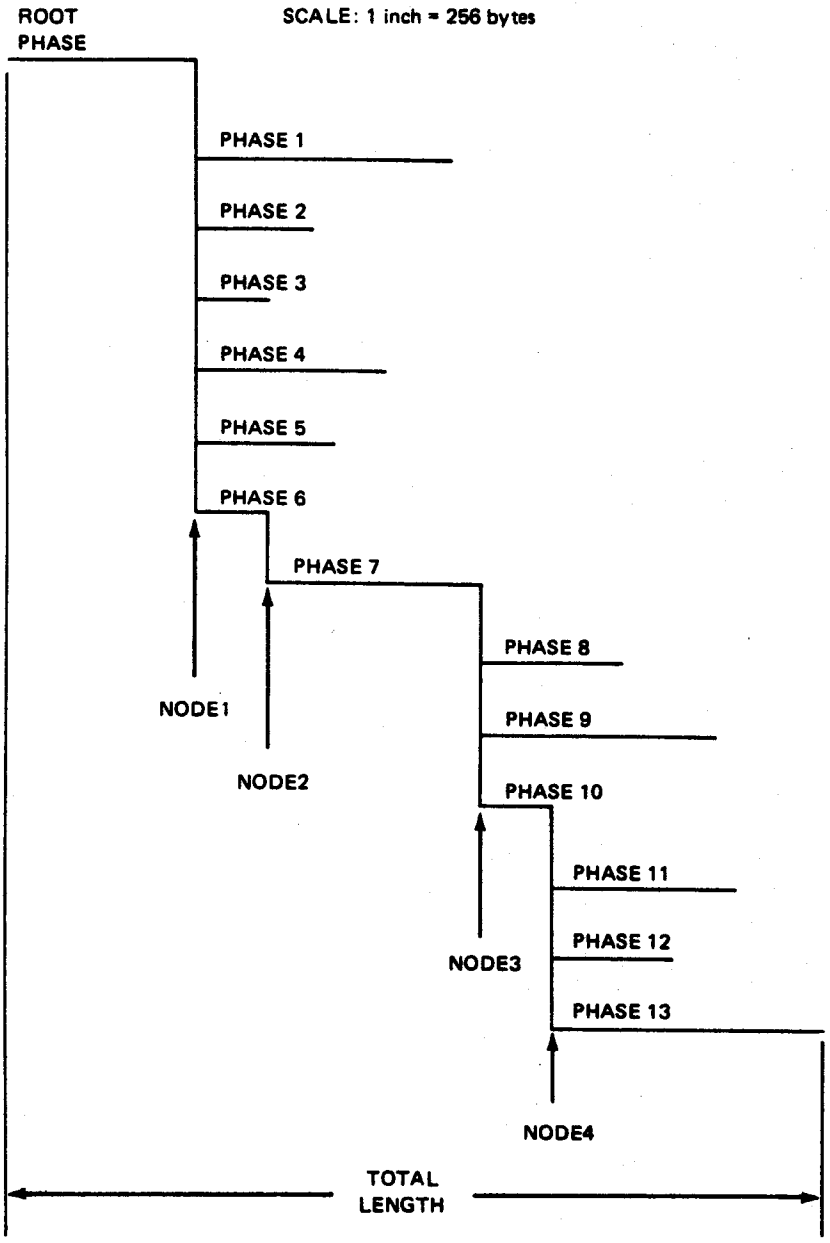


Figure 4-11. Program SAMPLE as a Multiphase Load Module

As can be seen by comparing Figure 4-9 with Figure 4-11, a load module constructed as a multiregion load module normally requires more main storage space for execution than the same program configured as a multiphase load module. Multiregion structures are most useful, however, when a need exists for a phase to reside in an area where it will not be overlaid by other phases that may not be directly associated with it. Also, it is sometimes possible to realize an actual saving in main storage space with a multiregion construction when one or more control sections are required for two or more distinctly separate phases, but are not required by any other phases. In this case, these CSECTs could be placed in a separate region, rather than being embedded in a phase common to the phases requiring them. Placing them in a common phase could unnecessarily affect the origins of succeeding phases even though the majority of these phases do not require these CSECTs. The opposite is true when these CSECTs are placed in a separate region; however, inasmuch as region origins always are assigned at the end of the longest path of the preceding region, unnecessary placement of CSECTs in separate regions may have an adverse effect on the overall length of the load module.

Regions are declared by the programmer with the REGION control statement in much the same way a phase is declared with an OVERLAY control statement. Both statements initiate construction of a new phase at some symbolic starting address, or node point, specified in the control statement. The only difference between the two statements is the way they cause the linkage editor to assign an origin to the phase being created. Region nodes are always logical and never reference a previously defined symbol because a new path is about to be constructed.

4.7. Linkage Editor Operation

When the linkage editor is called upon to perform its intended functions via a // EXEC LNKEDT job control statement, it searches the appropriate control stream data set for control statement data. If control statements specifying the object modules to be included in the load module are found, the linkage editor links these modules together in accordance with the control statements that affect the structure of the load module that also may be in the control stream. If no control statements that identify the object modules to be included are found, the linkage editor links together all the object modules currently in the job's run library file, as previously described.

Under normal operating conditions (no linkage editor capabilities suppressed or altered), the linkage editor performs the following operations for each link-edit job:

- Automatically includes any object modules needed to satisfy any references not defined in the object modules specifically included in the load module by the user
- Automatically deletes control sections redundantly included in any non-unique path of a load module
- Determines in what phase each common storage area is best located in the load module

- Automatically includes an automatic overlay control routine and its associated control tables in the load module if it is required for execution of the program
- Resolves multiple definition problems in the load module
- Includes partial object module elements (CSECTs), if specified by the user, in the load module
- Detects object modules that are marked reentrant, deletes their text, and creates appropriate shared records to indicate the reentrant code requirement
- Relocates ISD records detected in object modules and passes them to the load modules being created
- Produces CSECT/COM ISD records based on the CSECT/COM records being included

The processing involved in performing these operations is described in the remainder of this section.

4.7.1. Automatic Inclusion Processing

The linkage editor is designed to automatically include object modules in a load module when such modules are needed to satisfy references in the object modules specifically included in the load module by the user. When a reference is found in a specifically included object module for which no definition exists, the linkage editor searches up to two object module library files looking for an object module containing the definition.

The library files to be searched are specified by the user through the ALIB and RLIB keyword parameters of the // PARAM or LINKOP linkage editor control statements. If the definition is found, the entire object module containing the definition is automatically included in the load module being constructed. If the required definition is not found, all references to the undefined label are flagged as errors in the link-edit map. All automatically included object modules are placed in the root phase of the load module being constructed.

To facilitate use of the automatic inclusion feature and avoid redundant searches of object module files, library directories contain a cross-reference index that points to label definitions (CSECTs, COMs, and ENTRYs) in the object modules. These file directories are scanned for the required reference definitions, rather than the object module elements. The search for label definitions continues within the directories until all labels are defined or until the entire directory is scanned once without creating new, undefined references. All references that cannot be satisfied are flagged as errors in the link-edit map.

New, undefined references sometimes appear within an automatically included object module or in the object module elements introduced by INCLUDE control statements embedded in an automatically included object module. These embedded INCLUDE statements are processed in exactly the same way as INCLUDE statements embedded in specifically included object modules. Thus, an INCLUDE statement embedded in an automatically included object module could request the partial inclusion of an object module as well as the inclusion of a complete object module.

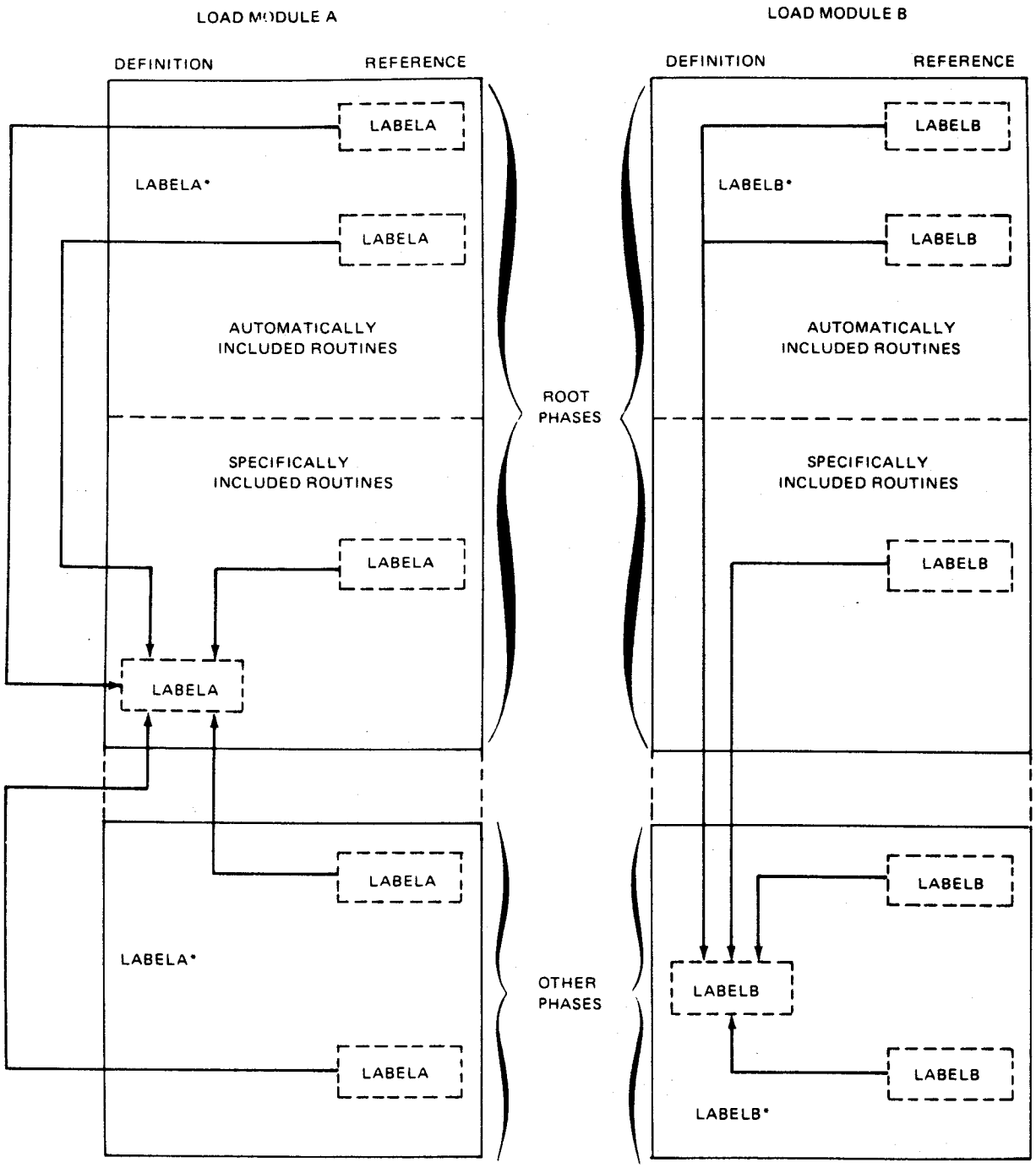
All object module elements included in a load module as a result of an INCLUDE statement being embedded in an automatically included object module are placed in the root phase of the load module as part of the automatically included object modules.

If a label is defined in an automatically included object module, that definition is used to satisfy all references to that label appearing in the load module. If a label is defined in a specifically included object module and a second definition of the same label is contained in an automatically included object module, the second definition is ignored by the linkage editor. If a label is defined in two or more specifically included object modules that are in the same phase, the first definition obtained by the linkage editor is used to satisfy all references to the label. These three referencing techniques are illustrated in Figure 4-12.

If an automatically included object module contains a control section whose name is already defined by a label in a previously included object module, that control section is not included in the load module even if that control section contains an ENTRY point definition that may be required in the load module.

Another function of the automatic inclusion feature is to construct a single-phase load module from all the object modules in the system job run library (YRUN) when the linkage editor is executed and no control statements defining the load module to be constructed are provided by the user.

The automatic inclusion feature can be inhibited by the user through the NOAUTO keyword of a // PARAM or LINKOP control statement. This capability is provided to enable the user to check the completeness of the definitions within his own program. In this instance, all references to undefined labels are flagged in the link-edit map without an attempt by the linkage editor to satisfy them.



*These definitions are deleted by the linkage editor.

NOTE:

The arrows indicate which definition is used to satisfy each reference.

Figure 4-12. Referencing Label Definitions in a Load Module

4.7.2. Automatic Deletion Processing

The linkage editor automatically deletes control sections and ENTRY points previously defined in the phase currently under construction, or in a phase that is on the path of the current phase. When a newly included object module is found to contain a CSECT, or an ENTRY point with the same name as a CSECT or an ENTRY point previously defined, the linkage editor deletes the second definition from the phase under construction. Unlabeled control sections are not automatically deleted because each of them is considered a unique entity for lack of a name. Shared definitions are considered to be in the root phase, so any subsequent duplicate definitions will always be deleted. However, nonshared definitions that are accepted before a shared definition is encountered are not deleted. Also, if such definitions are themselves in the root phase, they will cause the automatic deletion of all subsequent definitions, shared or unshared.

The only exception to this rule for automatic deletion is when multiple CSECTs are found to have the same name as a labeled common storage area. In this instance, the CSECTs are treated as block data subprograms and are used to initialize the associated common storage area. These multiple CSECTs, however, must be included in the load module after the inclusion of the related common section; otherwise, the second and subsequent CSECTs could be deleted automatically. Common (COM) sections in reentrant object modules are treated as DSECTs. In nonreentrant object modules, unnamed COM sections are never deleted. Deletion of named COM sections is subject to the rule that block data/common relations are not permitted across nonreentrant/reentrant code boundaries. Thus, if a shared CSECT has been accepted and, subsequently, a nonshared COM with the same name is encountered, the COM section is deleted. On the other hand, if a nonshared COM section has been accepted and a subsequent shared CSECT with the same name is found, the CSECT is deleted.

Thus, if a second definition for a name already defined on the path of the current phase is detected, the second definition is ignored. A single-phase program, therefore, will contain only one definition for each label that does not match a common section name. Only one definition is accepted, even if the subsequent definition is for an absolute entry. All absolute entry items are treated as low-priority items and are always deleted whenever at least one duplicate definition exists in the load module.

4.7.3. Common Storage Processing

The linkage editor constructs a common storage area for each unique COM section contained in a load module. COM sections with the same name, however, are assumed to be referring to the same common storage area; therefore, the linkage editor creates a single common storage area for these COM sections. The size of each storage area is equal to the largest size requested by all object module elements referring to a COM section. Thus, if one object module indicates that COM section A requires 256 bytes of main storage, and another object module indicates that COM section A requires 1,024 bytes of main storage, the linkage editor creates a single common storage area of 1,024 bytes to satisfy all object module references to COM section A. Blank (unlabeled) common storage is allocated in the same way.

The linkage editor assumes that the following rules concerning COM section specification have been adhered to by the object module element programmer.

- An ENTRY point in a load module cannot bear the same name as a labeled common storage area included in the load module.
- When a phase containing a CSECT with the same name as a common storage area is loaded for execution, that section is treated as a block data subprogram and is loaded in all or a portion of the common storage area with the same name. (Block data subprograms can be used to initialize common storage areas. Blank common storage areas cannot be initialized during loading unless the text following the common storage area declaration is for that COM ESD.)
- All COM sections defined in an object module are processed by the linkage editor, even if only a partial INCLUDE of an object module is taking place and the included object module element does not refer to any common storage area.
- COM sections encountered in reentrant object modules are treated as DSECTs. They do not result in any space allocation during the creation of a reentrant load module.
- Automatic deletion of nonshared common sections is subject to the rules established for deletion processing.

In the past, most linkage editors placed all common storage areas in the root phase of all load modules, regardless of the structure of the phases requesting access to the common storage area. This allocation scheme forces all phases of a load module to include the space required by all common storage areas in their main storage space requirements, even if they do not use any common storage space. The OS/3 linkage editor, however, is designed to promote the allocation of common storage areas. The promotion of a common storage area refers to its placement within a nonroot phase segment of a load module. The OS/3 linkage editor allocates common storage areas only as required by the structure of the phases referencing them.

Figure 4-13 illustrates a load module in which a common section was promoted. As shown, common storage areas A, B, and C are located in the root phase because they are referenced by phases whose only common phase is the root phase. But, because common storage area D is referenced only in phases 2 and 3, it was promoted to the phase 1 segment because this segment also is common to both phases. In this way, phases 4, 5, and 6 do not need to pay for the storage space required by common section D because it will be loaded only when phase 1 is loaded. Thus, this promotion technique allows a load module to require less main storage space then would otherwise be required. It should be noted, however, that common storage area D is overlaid each time phase 4 is loaded, and the use of common storage area D must be planned carefully by the programmer to ensure that required common storage area data is not overlaid before that data is processed. Nonroot phase common sections normally are feasible only when phased programs are executed in a straightforward fashion. For example, if phases 2 and 3 of Figure 4-13 were executed consecutively, followed by the execution of phases 4, 5, and 6, in that order, and the program was concluded without ever having to repeat either phases 2 or 3, common section D would

probably fit very nicely in phase 1. On the other hand, if phase 2 were executed and stored data in common section D for use by the phase 3 program segment, but the phase 4 program segment was loaded next, the common storage data for the phase 3 program would be overlaid and its data rendered useless to the subsequent execution of the phase 3 program segment.

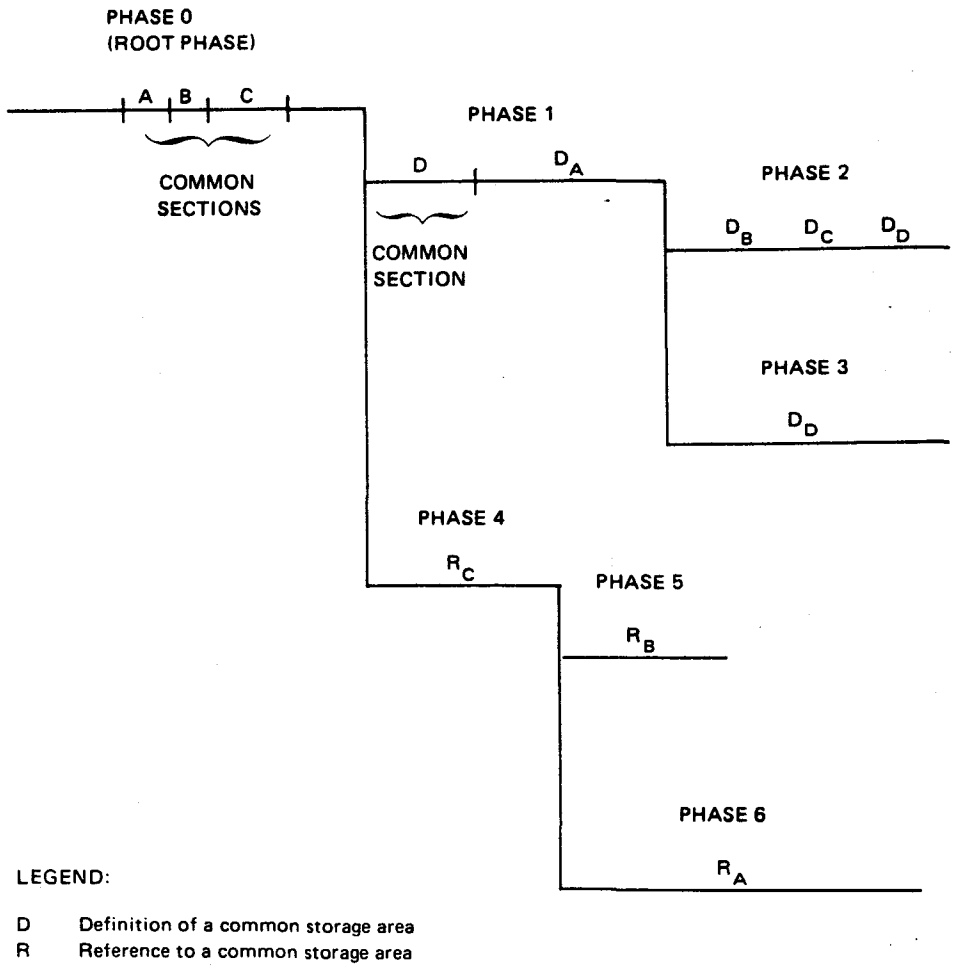


Figure 4-13. Example of Common Storage Promotion Scheme

This capability of the linkage editor can be enabled and disabled through the PROM and NOPROM keyword parameters of the // PARAM and LINKOP control statements. Thus, to inhibit the promotion and force root phase assginment of common storage areas, the user need only specify the NOPROM keyword in his input control stream to the linkage editor; otherwise, the promotion of common storage is enabled.

4.7.4. Automatic Overlay Control Processing

The linkage editor can automatically load the various phases of a multiphase or multiregion load module in main storage in accordance with the execution requirements of the program. Normally, the loading and execution of the phases of a program is done by the programmer through the object code in each phase of the load module. If the programmer wishes, however, he may leave this responsibility up to the linkage editor for the minor cost of the storage space required to include the automatic overlay control routine and its associated control tables in his load module. The exact amount of storage varies with the number of control table entries required to describe the load module.

When the programmer wishes to have his program phases loaded automatically, he must reference the ENTRYs he wishes to transfer control to with V-CON (type V address constants) references. Then, when he proceeds to link-edit his program as either a multiphase or multiregion load module, the linkage editor decides whether the automatic loading of phases is required based on

- The types of EXTRNs (type A or type V) contained in the object modules being included in the load module
- The types of references (inclusive or exclusive) in the load module
- The V-CON processing option, if in effect (V-CON processing is normally enabled)

If the V-CON processing option is in effect and at least one exclusive V-CON reference is in the object code being included in the load module, the linkage editor constructs a set of tables depicting the load module structure and includes them in addition to an automatic overlay control routine in the root phase of the load module. Also, the linkage editor modifies the V-CON references in the load module to make them reference the automatic overlay control routine. The automatic load routine (KL$OCP or KL$OCPR) that is copied into the load module depends on the number of regions comprising the load module. The KL$OCPR routine is capable of handling the automatic overlay requirements of a multiregion load module; KL$OCP is not.

When a load module containing the automatic overlay facilities is executed, its root phase is loaded in main storage, as is any other load module, by the supervisor, and program control is transferred to the program ENTRY point for the phase. As program execution proceeds, V-CON references cause the needed phases to be loaded, thus satisfying the type V address constant that referenced the label of the required entry point. A branch instruction is executed through register 15, forcing transfer of control to the overlay control routine because the type V address constant was previously modified by the linkage editor to reference the routine. The automatic overlay routine then checks to see if the required phase is in main storage. If it is, the overlay control routine branches directly to the proper address. If it is not, the automatic load routine

- Determines the phases currently in main storage
- Loads the required phase, and any phase on the path of the required phase, into main storage

- Records the new path as loaded
- Branches to the proper address

The overlay control routine executes LOAD macroinstructions as needed to obtain the required phases and records this information in its control tables. The path information is examined by subsequent invocations of the automatic overlay control mechanism and is altered if additional phases are loaded. Attempts made by the problem program to load phases directly do not cause an update of the path information; therefore, you would not issue any LOAD, LOADR, or FETCH macroinstructions in your problem program using the automatic overlay control routine.

The type V address constant in register 15 for the automatic loading of overlays has the following characteristics:

- It always occupies 4 bytes; the entire word is reserved for use by the linkage editor. Byte 0 of this word contains a number (zero or greater) that is an index into the list of NTAB entries. Bytes 1 through 3 consist of the address of the entry point table; however, a V-CON may have all zeros.
- It is intended for branching only and may not be used for addressing data. Data references do not cause the referenced phase to be loaded automatically, because transfer of control is essential for the process to be effective.
- When a type V address constant addresses a definition that is on the same path as the reference, the constant is treated as though it were a constant of type A. If such a constant is in register 15, the problem program branches directly to the required location, rather than transferring control to the overlay control mechanism.

The data required by the automatic overlay control routine to perform these functions is contained in its associated control tables. Brief descriptions of the components of the automatic overlay mechanism follow.

Overlay Control Routine

The overlay control routine is a program used for the automatic loading of segmented program structures. It is responsible for loading the program segments (phases) and maintaining the tree structure of the program. It supplies the needed interface and controls to allow the user to use segmentation without difficulty. It also causes automatic loading of the needed program phases as determined by exclusive V-CON references. An automatic ENTRY point (KL$OCP) always is supplied to allow references to the standard automatic overlay control routine. A second routine (KL$OCPR) is used if the load module structure comprises multiple regions. The control routine is always placed in the root phase of a load module and is entered through the entry point table.

Entry Point Table (NTAB)

The NTAB table also is always in the root phase. It front-ends the overlay control routine and contains such control information as the path being automatically loaded, where the phase is being loaded (or its relative position in main storage), and a list of entry points corresponding to the V-CON definitions in the problem program. The NTAB determines the phase to be loaded when a V-CON reference refers to a phase not in the same path as the reference. An NTAB entry is not created for any V-CON symbol already present in a phase higher in the current path (closer to the root phase). An automatic entry point (KL$NTB) always is supplied to allow references to NTAB.

Phase Table (PTAB)

The phase table contains information concerning the relationships between the phases of the load module. Only one phase table is built for any multiphase load module that requires automatic loading, and it is always in the root phase. An automatic entry point (KL$PTB) always is supplied to allow references to PTAB.

Region Table (RTAB)

The region table is generated only for load module structures using V-CON references in which multiple regions are involved. It describes the number of regions making up the load module and the highest phase contained in each region. The RTAB is used in conjunction with NTAB and PTAB, plus the region overlay control routine (KL$OCPR), to manage V-CON references in load modules with two or more regions. An automatic entry point (KL$RTB) always is supplied to allow references to RTAB.

4.7.5. Multidefinition Resolution Processing

The multidefinition resolution processing performed by the linkage editor depends on the type of definition being referenced.

Standard (Non-V-CON) References

If the user references a definition with a standard reference and the definition being referenced is not on the path of the reference, he is required to issue the appropriate supervisor FETCH and LOAD macroinstructions to ensure that the phase containing the proper definition is loaded when the definitions it contains are referenced. (The linkage editor allows standard EXTRN-ENTRY relationships to occur across phases without any special processing when V-CONs are not involved.)

Because of the automatic deletion mechanism of the linkage editor, only one definition will ever be present on each path of a load module. If a definition is present in the root phase, no identical definitions will be in other phases unless the path involved overlays the root phase entirely.

If there is only a single definition for a reference, the linkage editor obtains the appropriate phase number and value for the definition and applies it accordingly. But, when a reference is marked as being multiply defined, each phase number assigned to each definition is, in conjunction with the segment table, used to determine the relationship between definitions and references. The linkage editor then satisfies the reference in accordance with the following logic:

a. If a definition that satisfies a reference is in the same phase as the reference, the linkage editor uses it to satisfy the reference.

b. If a definition that satisfies a reference is on the path of the reference, the linkage editor uses it to satisfy the reference.

c. The linkage editor determines if the reference is on the path of one or more definitions that can satisfy the reference. If it is, the linkage editor chooses the first definition following the reference to satisfy the reference.

d. The linkage editor chooses the first definition following the reference (higher phase number), regardless of path associations, to satisfy the reference if one exists.

e. The linkage editor uses the first backward definition (lower phase number) to satisfy the reference.

These five reference-definition relationships are illustrated in Figure 4-14. Further, wherever the linkage editor must satisfy a reference in accordance with c, d, or e, it is indicated on the link-edit map.

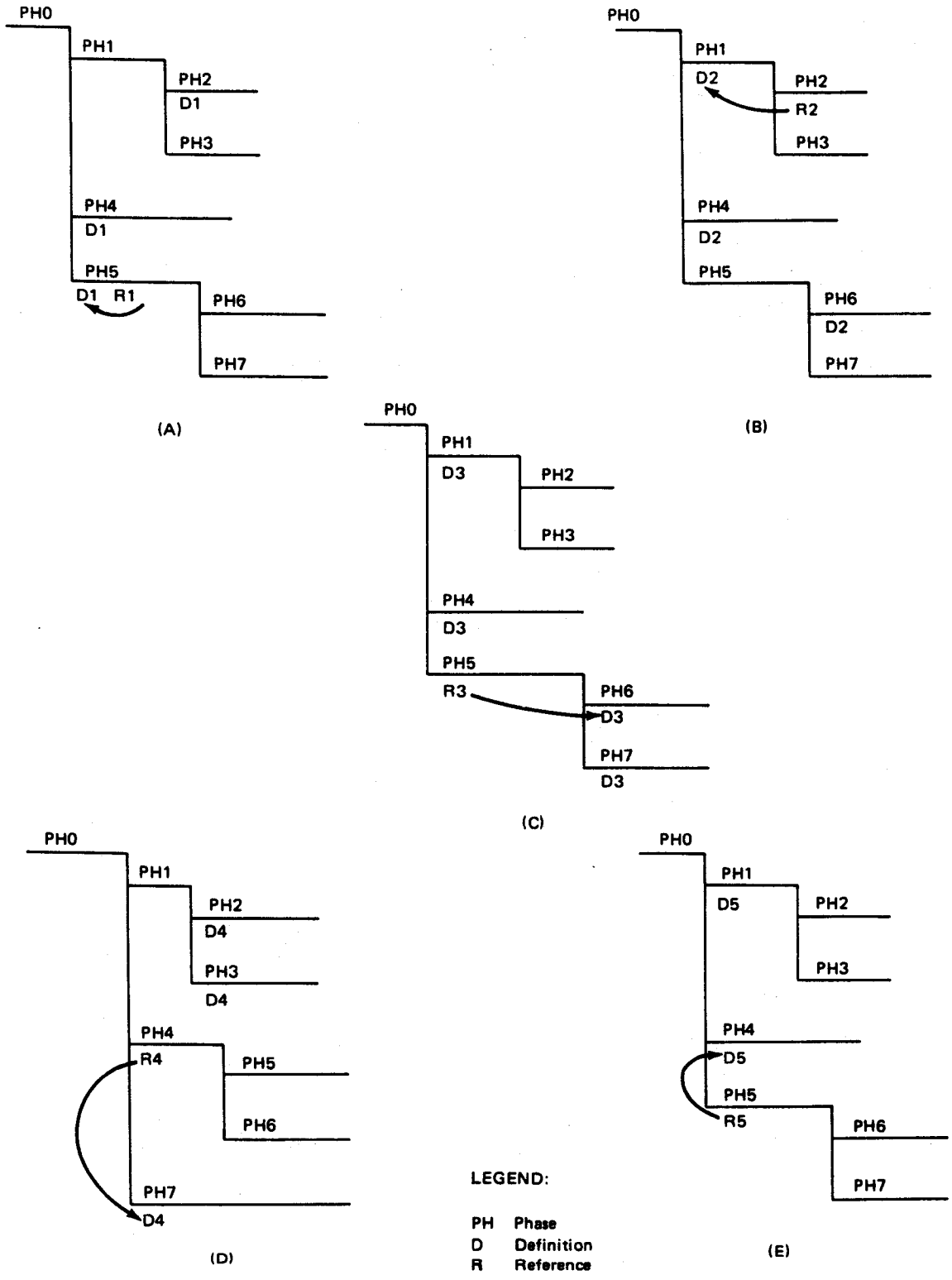


Figure 4-14. Multidefinition Resolution without V-CON References

V-CON References

The resolution processing for V-CON references is much the same as that for standard references, except that

- A V-CON reference may be converted to an A-CON reference.
- A V-CON reference forces control to the automatic overlay control mechanism.
- Exclusive V-CON references do not initiate diagnostic messages.
- A V-CON reference resolved to a shared definition is converted to an S-CON (shared constant).

Thus, at resolution time, multiple definitions for any referenced symbol are handled as follows:

1. If a definition is on the path of the reference, that definition is selected by the linkage editor to satisfy the reference. (Only one inclusive definition is possible.)
2. Otherwise (definition is exclusive of the reference), the first forward definition is chosen by the linkage editor if one exists. If none exist, the first backward definition is chosen.

When an object module incorporating V-CONs is being included, V-CON processing will include the following. Whenever a V-CON reference occurs, the linkage editor keeps track of the first such reference since the last definition. For multidefined V-CONs, separate NTAB entries may be constructed, as references are assigned to different definitions of the same symbol. This selection of definitions is a function of where the appropriate references occur with respect to path relationships concerning those definitions.

Multidefinition V-CON processing occurs

a. If a V-CON definition is on the path of the V-CON reference (or references), that reference will be treated as a direct A-CON reference and will be flagged accordingly.

b. If a V-CON reference is on the path of a V-CON definition, it is a valid V-CON reference.

c. If a V-CON reference is on the path of multiple definitions, the automatic deletion mechanism will eliminate the subsequent definitions (where such definitions are contained in the same path or phase) and the first definition will be used to satisfy that reference.

d. If V-CON references and multiple definitions exist on separate paths, the first forward definition will be used if there is one; otherwise, the first backward definition is used.

e. If V-CON references and multiple definitions are on separate paths but in the same direction (forward or backward), the first definition encountered is used.

These five reference/definition relationships are illustrated in Figure 4-15.

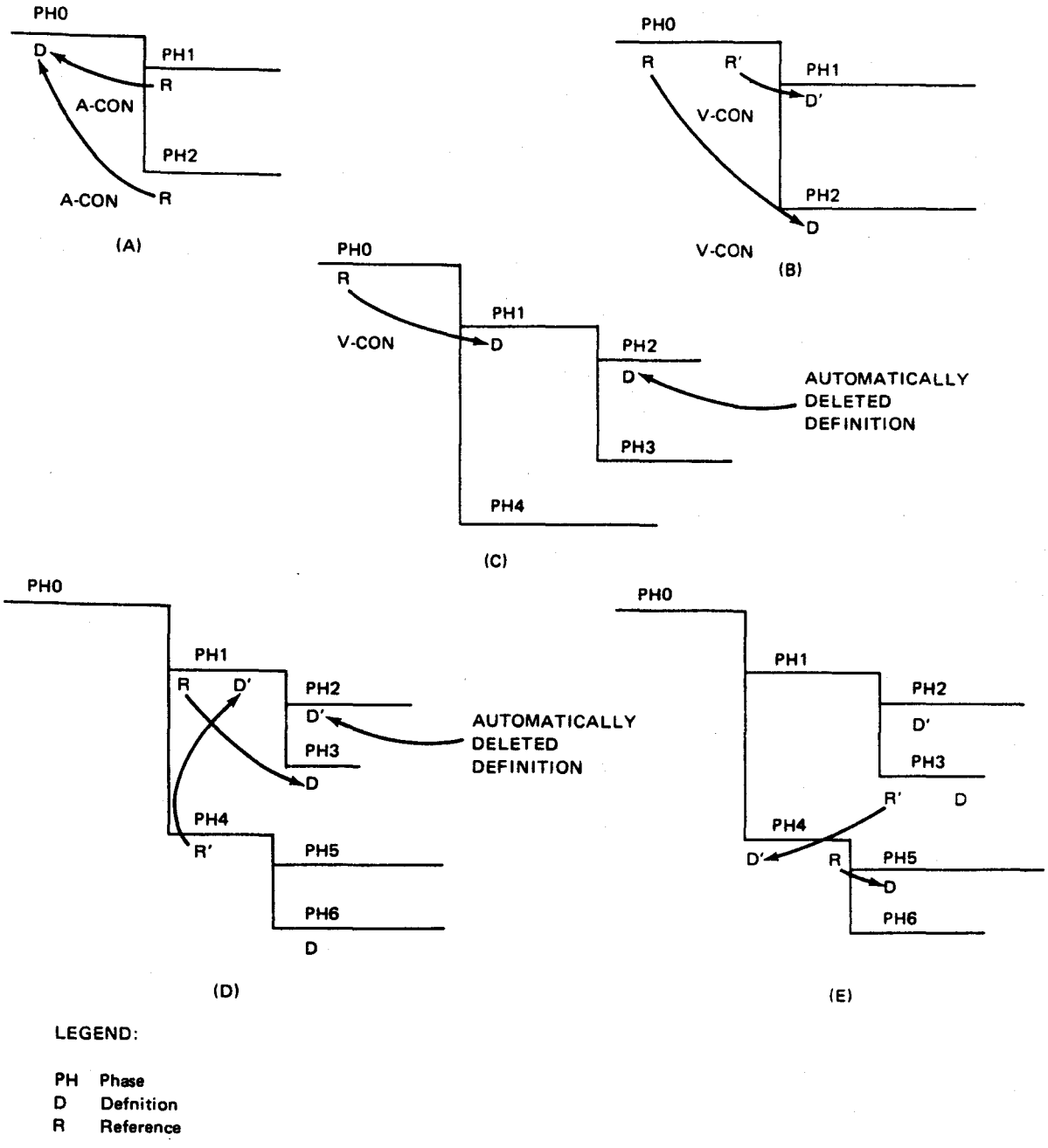


Figure 4-15. Multidefinition Resolution with V-CON References

4.7.6. Partial INCLUDE Processing

The linkage editor can include only specific CSECTs of an object module in a load module, if the user so desires. These CSECTs are specified by the user in INCLUDE control statements that contain a CSECT name list as its second operand, as well as the name of an object module as its first operand. Up to nine control sections can be specified in any one INCLUDE statement.

When CSECTs are specifically referenced in an INCLUDE statement, they are inserted in the load module in the order that they appear in the object module, regardless of the order in which they are specified in the INCLUDE statement. The rearrangement of CSECTs in a load module, however, can be accomplished by multiple INCLUDE statements, each of which references only a particular CSECT. In this way, the INCLUDE statements can be arranged to cause the resulting load module CSECT structure to assume any form desired. The smallest segment of object code that may be included in a load module is a CSECT.

Whenever an object module is accessed for a partial inclusion of its elements, any control statements that may be embedded in the object module are processed by the linkage editor as standard, embedded control statements, and any common sections defined in the object module are included in the load module.

4.7.7. Shared Code (Reentrant Code) Processing

Shared or reentrant code is a piece of code that is not self-modifying. Thus, a number of programs can call upon a single copy of this code and run concurrently. Sharing code effectively reduces the main storage requirements of the system. Consider two programs, X and Y, that call upon a reentrant routine, Z. It is obvious from Figure 4-16 that sharing Z allows both programs to be executed in less main storage space than would otherwise be required.

The librarian is capable of marking object modules reentrant by setting a flag in the object module header record. Unless otherwise marked, object modules are nonreentrant. The flag can be turned on or off by the librarian RENAME control statement.

It is the responsibility of the linkage editor to detect reentrant object modules, delete their text, and create the appropriate interfaces to enable linkages to be established at execution time. This feature can be enabled or disabled through options on // PARAM or LINKOP cards.

Since the reentrant code itself does not appear in the load module produced, it must be link-edited separately. This is done in a special mode of the linkage editor through // PARAM-LINKOP cards.

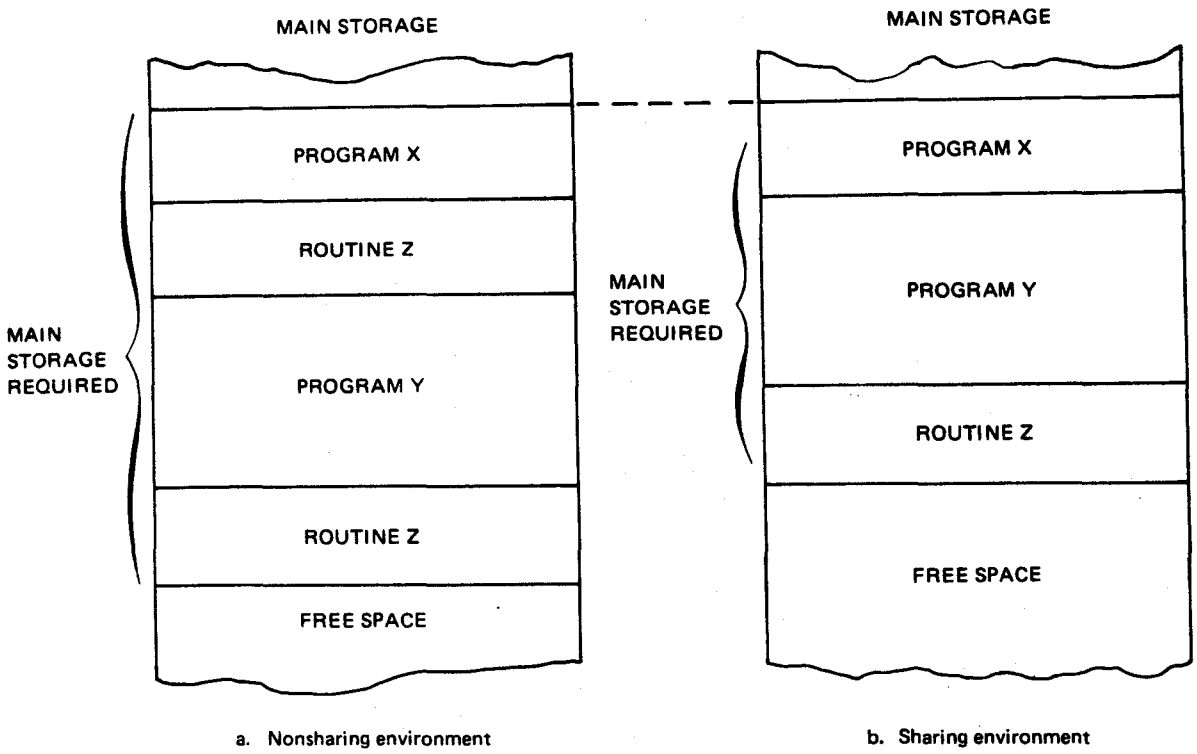


Figure 4-16. Effect of Shared Code on Main Storage Requirements

Thus, there are three processing modes controlled by // PARAM-LINKOP options:

1. Normal link-edit with no recognition of reentrant object modules (NOSHARE, NORNT options).

2. Link-edit with sharing capability enabled (SHARE option).

3. Link-edit of a reentrant module by itself (RNT option).

Share Facility

The linkage editor detects reentrant object modules under either of the following conditions:

1. An INCLUDE control statement was supplied that referenced a specific object module which, when located, was found to be marked reentrant.

2. As a result of an unresolved EXTRN reference in the user module being link-edited, the automatic INCLUDE mechanism was triggered. When the definition was detected, the object module containing the definition was found to be marked reentrant.

Provided that the share processing facility is enabled, the linkage editor, upon encountering the reentrant object module, performs the following:

1. The text from the object module (or part, if partial INCLUDE) is dropped and does not contribute towards the load module being produced.

2. The CSECT and COM lengths in the object module are not reflected in the load module size.

3. COM sections and ISD records are ignored.

4. All ESD items (except COMs) are entered into the linkage editor's internal symbol table and are processed normally.

5. Shared definitions (definitions encountered in the reentrant object module) are treated as if they occurred in the root phase and are subject to automatic deletion processing rules. This includes block data/common conflicts between reentrant and nonreentrant code.

6. A special record (the resource record) is created in the load module. This record contains the name of the object module. At program execution time, it informs job control of the requirement for a resource, namely the reentrant module.

Satisfying shared resource requirements is a part of the total job scheduling process and is performed before actual program execution begins.

Linkage in Shared-Code Environment

In the course of a link-edit with the share facility enabled, the linkage editor distinguishes between the different types of EXTRNs as shown in Figure 4-17. As is obvious from Figure 4-17, reentrant code cannot call nonreentrant code.

Each SEXTRN causes the creation of a SEXTRN record in the load module. This record contains the SEXTRN name and a unique number assigned to the SEXTRN called the SINDEX number. The SEXTRN record is used to link the load module with the reentrant code at execution time.

EXTRN Found in ↓	Resolved to →	Nonreentrant Object Module	Reentrant Object Module
Nonreentrant object module		Allowed normal EXTRN	Allowed SEXTRN (shared EXTRN)
Reentrant object module		Not allowed unless the definition is absolute	Allowed provided requirements in 4.7.7 are met

Figure 4-17. EXTRN Resolution Processing in Shared-Code Environment

Shared Constants

Type A or type V address constants associated with SEXTRNs are known as shared constants (S-CONs). You can transfer control to shared code by loading such a constant into register 15 and branching to it. Only type 1 linkages are allowed for calls from nonreentrant to reentrant code. Register 13 must be loaded with the address of an 18 full-word save area.

The shared constant in register 15 has the following characteristics:

1. It always occupies 4 bytes; the entire word is reserved for use by the linkage editor. Byte 0 of this word contains an index (called the SINDEX (shared INDEX) number), which is a unique number for every SEXTRN. Bytes 1 through 3 consist of the address of the preamble SEXTRN processor. This address is negative since the SEXTRN processor resides in the prologue.

2. It is intended for branching only and may not be used for addressing data. The SEXTRN processor, after receiving control, converts the SINDEX number into an absolute address and branches to it. It uses the save area and user registers for its own housekeeping, and then restores them before branching to reentrant code, which must have its own mechanism for saving registers if it needs to.

3. It must be coded as a symbol, not as an expression. For example, V(X) and A(Y) (where Y is declared as an EXTRN) are valid shared references, V(X+4) is not.

Link-Editing Reentrant Code

In order for reentrant code to be loaded, it must be link-edited. This is done in a special mode of the linkage editor (RNT option on // PARAM-LINKOP cards). The link-edit of reentrant code has the following special characteristics:

1. No overlay structures are permitted.

2. The load module name may be up to eight characters long.

3. COM sections are treated as DSECTs. No storage allocation is performed for them.

4. For each relative definition (CSECT, ENTRY, linkage editor EQU), a SENTRY record is created in the load module. This record informs job control at execution time that a definition is available in the module and can be used for establishing linkages. No SENTRY record is produced for absolute ENTRYs and EQUs. The relative definitions are known as SENTRY (shared ENTRY) items.

5. Only specific INCLUDEs are meaningful and the text encountered during the specific INCLUDE process contributes to the load module. All references in the included object modules must be inclusive; that is, the reentrant load module must be self-contained. Option NOAUTO should be in effect during the course of the link-edit.

6. All address constants must be absolute. The code must relocate them, if necessary.

7. Normally, only one reentrant object module would be specifically included. The linkage editor will permit more than one reentrant object module to be specifically included provided the user program directly references only one of the object modules being included. Stated another way, there must be only one object module that is at the highest level of the intercalling sequence within the group of reentrant object modules being included. The name on the LOADM control statement must be the same as the name of the highest level reentrant object module.

8. The reentrant load module must be output on the system load library (YLOD).

9. The ISD records in the object module are ignored.

Shared Records

The linkage editor creates special records in load modules that reference reentrant code and also in the reentrant load module. These records provide information on linkages to be established and the reentrant modules that are required.

The format of a nonreentrant user load module that references reentrant modules is shown in Figure 4-18. A flag is set in the phase header record indicating that shared code is referenced. For each reentrant module referenced, a resource record is produced. This record contains the name of the reentrant load module and a unique number called the resource number. For each EXTRN in the user module resolved to a definition (CSECT/ENTRY) in the reentrant module, a SEXTRN record is produced containing the CSECT/ENTRY name and number of the resource that contains the definition. If an EXTRN is resolved to an ENTRY which has the same location as its CSECT, the reference is considered being made to the CSECT. Therefore, the SEXTRN record contains the CSECT name rather than the ENTRY name. However, several SEXTRN records can be created for each resource record depending upon the number of definitions referenced in the resource (reentrant object module). If the user module references other reentrant modules, additional sets of resource and SEXTRN records will be created.

The format of a reentrant load module is shown in Figure 4-19. The phase header is marked with a flag indicating that it is a reentrant load module. Relative definitions (CSECTs, ENTRYs, and EQUs) detected in your link-edit are called SENTRY (shared ENTRY) items and produce SENTRY records. However, a SENTRY record is not produced for a COMMON, an absolute ENTRY, an absolute EQU, or a relative ENTRY at the same location as its CSECT. A SENTRY item contains the SENTRY name, its link origin, and a unique number called the SENTRY number. No ISD records are produced in a reentrant load module. See Table B-18 for the format of a SENTRY record.

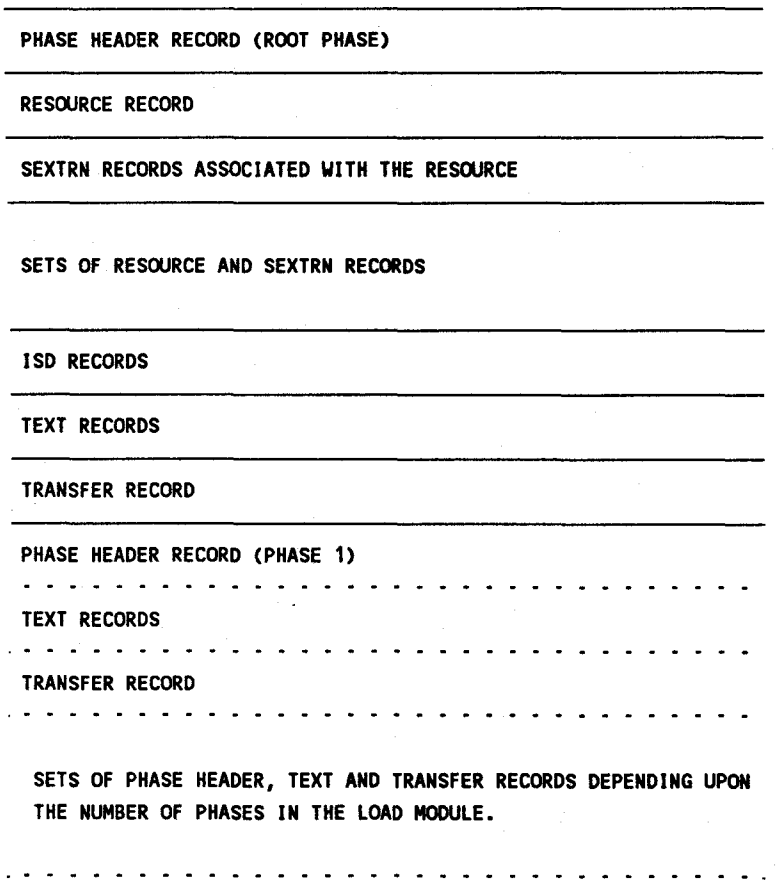


Figure 4-18. Format of a Nonreentrant Load Module that References Shared Code

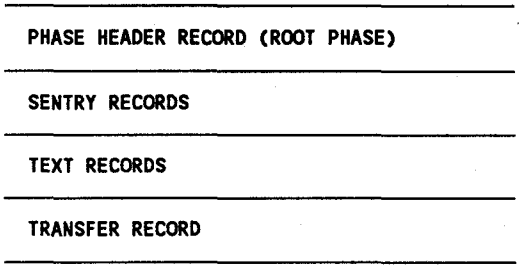


Figure 4-19. Format of a Reentrant Load Module

When the user load module is executed, its resource and SEXTRN records are examined to determine reentrant code requirements for the module. An attempt is made to resolve SEXTRNs against reentrant code already loaded. If the required reentrant code is not already loaded, YLOD is searched for the modules named in the resource records.

When a required module is found, a match is attempted between the SEXTRNs in the user module against the SENTRYs in the reentrant load modules. Linkages are then established based upon the matches, and the appropriate reentrant modules are loaded. Shared constants themselves remain unchanged. Satisfying shared resource requirements is a part of the total scheduling process and is performed before actual program execution begins.

Consider the following example. A user object module USER has external address constants R2 and T1 and a type V address constant T3. A reentrant object module R1 exists with CSECT R1 and ENTRY points R2 and R3. Another reentrant object module, T1, has CSECTs T1 and T2 and entry point T3. Figure 4-20 depicts the outputs of the link-edits of USER with the NOSHARE and SHARE options specified, and the link-edits of R1 and T1 with the RNT option specified. Note that, in b, no SEXTRN records were created for CSECTs R1 and T2 or ENTRY R3. This is because USER does not require these definitions to satisfy any of its external references.

4.7.8. Internal Symbol Dictionary (ISD) Processing

The internal symbol dictionary (ISD) records describe user program symbols. These records can appear in either object or load modules. When the ISD records appear in object modules, they are generated by certain language processors; while in load modules, the linkage editor generates them.

As just mentioned, the ISD records are used as descriptor records and, therefore, do not increase the size of the object or load module. When they appear in a load module, they are not loaded with the records at execution time, so no additional main storage is required. However, the ISD records do play an important role if your program has an abnormal termination and the JOBDUMP option was specified on the OPTION job control statement. In this case, JOBDUMP reads the ISD records in the load module and does the following:

- Prints the dump segmented by the CSECTs of the user program.
- Prints the user-defined symbols and tables. The dump produced is formatted and will include compile and link origins, source line number of the symbol, data type, and value.

You can inhibit the generation of ISD records in your load module by using the linkage editor option NOISD on the // PARAM-LINKOP control statement. If you inhibit the ISD record generation and an abnormal termination occurs in your program, then JOBDUMP will produce an unformatted dump of the load module area.

PHASE HEADER RECORD FOR USER
TEXT RECORDS (OBJECT CODE FROM USER, R1 AND T1)
TRANSFER RECORD

a. USER with NOSHARE option specified

PHASE HEADER RECORD FOR USER
RESOURCE RECORD FOR R1
SEXTRN RECORD FOR R2
RESOURCE RECORD FOR T1
SEXTRN RECORD FOR T1
SEXTRN RECORD FOR T3
TEXT RECORDS (OBJECT CODE FROM USER)
TRANSFER RECORD

b. USER with SHARE option specified

PHASE HEADER RECORD FOR R1
SENTRY RECORD FOR R1
SENTRY RECORD FOR R2
SENTRY RECORD FOR R3
TEXT RECORDS (OBJECT CODE FROM R1)
TRANSFER RECORD

c. R1 with RNT option specified

PHASE HEADER RECORD FOR T1
SENTRY RECORD FOR T1
SENTRY RECORD FOR T2
SENTRY RECORD FOR T3
TEXT RECORDS (OBJECT CODE FROM T1)
TEXT RECORDS (OBJECT CODE FROM T2)
TRANSFER RECORD

d. T1 with RNT option specified

Figure 4-20. Link-Edit Output

Object ISD Records

The object ISD records are generated in your object module by certain language processors. The linkage editor processes these records so that if an abnormal termination occurs while executing your load module, JOBDUMP can give you a formatted dump displaying user program symbols and their attributes. There are two types of object ISD records.

- Type 3 ISD record

  This record describes user-defined and compiler-generated symbols. The Type 3 record contains information such as: symbolic name, source line number of the symbol, level number, data type, and compile origin of the symbol.

- Type 4 ISD record

  This record contains English text and is used by JOBDUMP to print titles and headings in the dump.

Load ISD Records

The load ISD records are generated in your load module by the linkage editor provided the ISD option on either the // PARAM or LINKOP control statement was specified. The ISD record generation is based on the following:

- The presence of object ISD records in the object module. The linkage editor relocates their addresses and passes them into the load module.
- CSECTs/COMMONs accepted during the link-edit job.
- ISD records are not produced for the link-edit of a reentrant load module (RNT parameter).
- If you specified the SHARE parameter, the object ISD records in an included reentrant object module are ignored. Also, CSECTs and COMs in the reentrant object module do not result in the generation of either Type 1 or Type 2 ISD records.
- Deleted CSECTs or COMs do not contribute toward the generation of load ISD records.
- Object ISD records belonging to deleted CSECTs or COMs are ignored.

There are four types of load ISD records:

- Type 1 ISD record

  This record describes a CSECT. It contains the CSECT name, compile and phase relative origins, size, and phase number.

- Type 2 ISD record

  This record describes a COMMON section. It contains the COMMON name, compile and phase relative origins, size, and phase number.

- Type 3 ISD record

  This record is the relocated form of the Type 3 object ISD record.

- Type 4 ISD record

  This record is the relocated form of the Type 4 object ISD record.

These records are not loaded at execution time, and are only used if an abnormal termination occurs in your program. JOBDUMP, if specified, uses the Type 1 and Type 2 ISD records to segment the dump by CSECT and COMMON sections. JOBDUMP uses Type 3 and Type 4 ISD records to print user program symbols and values converted to their proper data types.

4.7.9. User Program Switch Indicator (UPSI) Setting

The linkage editor prints the UPSI byte settings along with the error count at the end of the link-edit map. When external references remain unresolved, the linkage editor sets the UPSI byte to X'20'. Also, when the linkage editor issues an error message, the UPSI byte is set accordingly. The linkage editor error messages are found in the *System Messages Reference Manual* (UP-8076).

Section 5
Programming Considerations

5.1. Introduction

Because the allocation of main storage is a primary concern in a multiprogramming system, the problem programmer should always consider the advantages of having the linkage editor construct this program either as a multiphase or multiregion load module. Further, he should consider this aspect of his program before he begins to code it because the use of certain object code facilities, such as common storage sections, enhance the ability of a program to be structured.

This section concerns itself with the considerations involved with structuring a load module. The most important factor in the construction of a load module is how its object code segments are coded, and whether these segments take advantage of the capabilities of the linkage editor.

5.2. Overlay Structures and Dependencies

Programs can be overlaid to minimize their main storage requirements. The possibility of constructing an overlay program depends on the relationships between the various control sections and phases to be created. These can overlay each other only if they do not need to be in storage at the same time and do not reference each other, either directly or indirectly.

The phase segments of an overlay program must be organized in an overlay tree structure. The following factors should be considered when organizing the tree structure of a load module:

- Phase and control section dependency
- Length of the multiphase program
- Frequency of usage of each control section
- Possibility of using separate overlay regions

To begin building an overlay structure, the user should first form a root phase that contains the modules that will receive control from the start of execution and those which should always remain in main storage. The rest of the structure should then be developed in accordance with the information presented in the following paragraphs.

5.2.1. Phase Dependencies

Whenever a phase is in main storage or is being loaded in main storage, all the phases in its path also should be in main storage. Phases may be loaded in any sequence whatsoever and reloaded any number of times, as required by the logic of the program. The assigned location of the phases has no bearing on the order in which the phases are executed. Any part of a phase that is modified during its execution will remain so only until the phase is overlaid.

5.2.2. Control Section Dependencies

A control section can receive control from another control section, but both sections must be in main storage before execution can continue beyond a given point in the program. The requirements of a control section for a given routine in another control section determine such dependency. Conversely, that control section is dependent upon any other control section from which it can receive control or where the other section has a need to process the former section's data.

5.2.3. Program Length

The length of a multiphase program must be considered by the user in the construction of his program. The length of the longest path is the minimum storage requirement for a multiphase program. Also, when a program is constructed with the automatic overlay mechanism, the storage requirements of the necessary control routine, entry table, phase table, and possibly region table also must be considered.

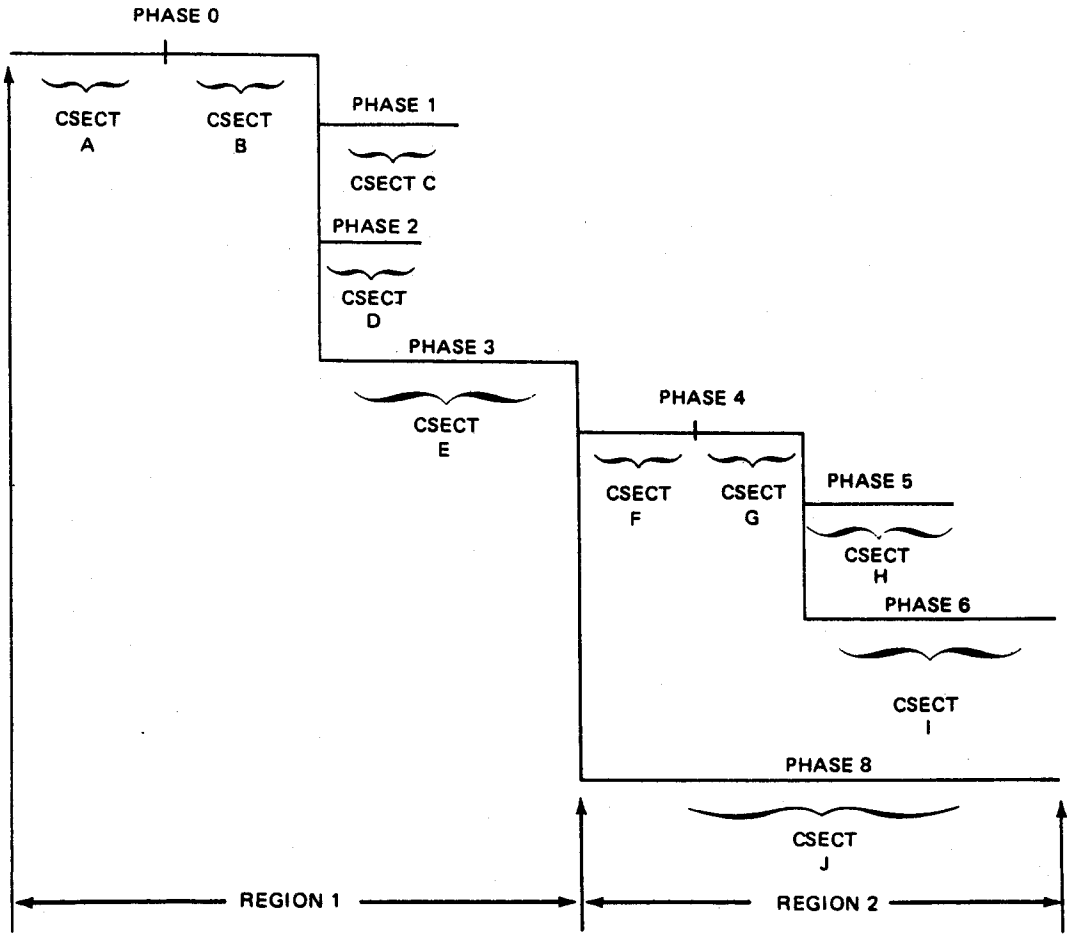
5.2.4. Phase Origins and Node Points

The origin of the initial, or root phase, segment is assigned by the linkage editor at zero. The relative node point of each phase is determined as zero plus the length of all phases in the path. The characteristics of the first symbol in the operand field of an OVERLAY statement designate the phase origin and node point.

5.2.5. Use of Multiregions

Multiregion structures can, in certain instances, capitalize on loading efficiency and realize a saving in main storage space. Phases not on common paths can access each other and, where identical copies of one or more CSECTs are required at different times by different exclusive overlays, they may reside in a separate region not affected by the path loading in the opposite region. Region structures also will decrease main storage needs when the creation of a common inclusive phase is not feasible or possible without increasing the length of the longest path of the current region. Such structures can provide a useful and convenient method of load module structuring where such concerns are paramount.

Figure 5-1 illustrates a program whose structure as a multiregion load module results in a saving of main storage space, given the phase and CSECT dependencies noted in the illustration. As shown, if CSECTs F, G, and J were root phase resident, the load module storage needs would increase because these CSECTs could no longer overlay each other.



NOTES:

Assumed program logic dictates the following:

1. Phases 1, 2, and 3 can overlay each other.
2. Phases 1 and 3 require CSECTs F, G, and J.
3. Phase 2 requires CSECT J.

Figure 5-1. Example of a Program Structured as a Multiregion Load Module

Creation of two additional phases in region 1, each containing the exclusive CSECTs, would not appear possible because a path conflict would be created. Thus, the region structure shown is more feasible.

If CSECTs F, G, H, I, and J were required by phases 1, 2, and 3, and if they were required at the same time, such that they could not be overlaid, root phase residence would be practical.

If CSECTs F, G, H, and I were needed by phases 1 and 2, and CSECT J only needed by phase 3, another common node point and phase could be constructed in region 1. In this case, a second region, as shown in Figure 5-1, would adversely affect the amount of storage required for the load module.

Section 6
Control Statements

6.1. Introduction

The linkage editor control statements described in this section direct the construction of a load module for OS/3 from specific or implied object modules and object module elements. Linkage editor control statements normally appear in an OS/3 job control stream, as illustrated in Figure 6-1. However, linkage editor control statements also may be contained in a source module or embedded (nested) in the object modules called by the linkage editor INCLUDE control statements. The rules for coding and embedding linkage editor control statements follow.

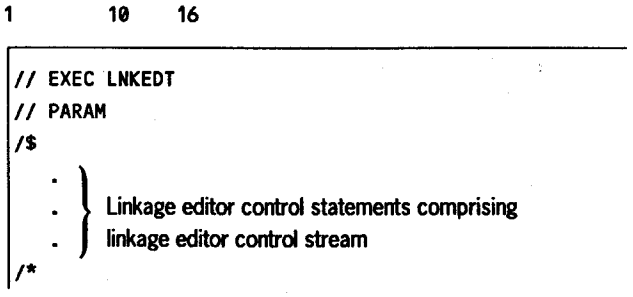


Figure 6-1. Typical Linkage Editor Control Stream

6.2. Coding Format

The general format of a linkage editor control statement is shown in Figure 6-2. The label field begins in column 1, is terminated by a blank column, and may contain up to eight alphanumeric characters. The label field is blank for all linkage editor control statements except the equate (EQU) statement. The operation field must be preceded and followed by at least one blank column, and contains the operation code of the function to be performed. If the label field is blank, the operation field may start in column 2. The operand field begins with the first nonblank character following the operation field, is terminated by a blank column, and cannot extend beyond column 71.

The operand field may contain any number of operands, depending on the function to be performed, and operands must be separated by commas. Continuation statements are not allowed.

1 LABEL	ΔOPERATIONΔ	OPERAND 71	72 80
Blank (1- to 8-character string for EQU only)	Must be delimited by blank columns.	Cannot extend beyond column 71; continuation statements are not allowed.	Not used

Figure 6-2. General Linkage Editor Control Statement Format

Comment statements, identified by an asterisk in column 1, may appear anywhere in a linkage editor control stream. Comments do not initiate any processing by the linkage editor but are printed out on the process map portion of the link-edit map.

6.3. Placement of Control Statements

In keeping with their functional uses, the following guidelines apply to the placement and sequencing of control statements:

- A LINKOP, LOADM, OVERLAY, or REGION control statement may be followed by an INCLUDE, EQU, MOD, or RES statement.
- An INCLUDE statement must be followed by at least one object module header record, and may then be followed by any number of embedded control statements, followed by a transfer record and, again, any number of embedded control statements. Any linkage editor control statement may then follow the INCLUDE statements.
- An ENTER statement may not be followed by an INCLUDE statement.
- The LOADM statement normally is the first control statement in a control stream.
- The ENTER statement normally is the last control statement in each phase of a control stream.
- Embedded control statements may be placed before or after the control sections in an object module, but may not be placed within a control section.
- Embedded control statements that affect load module structure (LINKOP, LOADM, OVERLAY, REGION, and ENTER) cannot be embedded in an automatically included object module.

6.4. Embedded Control Statements

Linkage editor control statements may be embedded in object modules either immediately after the object module header record or immediately after the object module transfer record. They may also be placed before or after the control sections in an object module, but may not be placed within a control section. Embedded control statements must be inserted in object modules prior to their being link-edited. The system librarian may be used to perform this function.

Embedded statements are processed as are other control statements, except when a statement that affects the structure of a load module (OVERLAY, REGION, LINKOP, LOADM, and ENTER) is detected in an automatically included module. Because automatically included modules always are included in the root phase of a load module, these statements are not permitted to be embedded in automatically included modules and are flagged as errors in the link-edit map. If one or more object modules contain embedded INCLUDE control statements for a module (including itself) which is already being included, then an inclusion loop may result for the link-edit in question.

All the control statements embedded in an object module are processed by the linkage editor, even if the object module is being accessed for a partial inclusion of its object code.

Whenever the control statements LINKOP and LOADM are encountered as embedded control items, they immediately signal the end of the current link-edit but are never subsequently processed. Therefore, these control items must not be embedded in object modules with the implied intention of triggering multiple link-edit operations or (via LINKOP) of supplying additional parameters to the current link-edit.

6.5. Basic Control Statement Processing

The linkage editor cycles through two control modes for each load module it generates. The exact processing done in each mode is determined by the control statements processed in each mode. For this discussion, the control statements that affect the operation of the linkage editor are divided into two groups. The first group comprises all the control statements that direct the basic operation of the linkage editor and includes:

- All job control statements directed to the linkage editor - start-of-data (/$), parameter specification (// PARAM), and end-of-data (/*)
- All LINKOP statements
- The LOADM statement

These statements are processed in the first control mode.

The second group of control statements consists of those which basically affect the structure and content of the load module, rather than the operation of the linkage editor. All remaining linkage editor control statements comprise this group, and are processed in the second control mode.

All the group 1 statements input to the linkage editor for the purpose of constructing any given load module may come from the primary control stream, plus any number of user source libraries. (See 6.6.1 for a description of the // PARAM and LINKOP CLIB keyword.) All group 2 statements, however, must come from a single source. The first group 2 statement detected initiates mode 2 processing, and mode 2 processing continues until INCLUDE processing is terminated for the load module. A given load module may thus pick up options and/or its load module name from multiple sources, but its structure must be defined in a single input source (primary control stream input or user source library).

User source libraries for linkage editor control statements are specified by the CLIB keyword of the // PARAM or LINKOP control statements. The processing performed during construction of a load module depends, in some respects, on the source that contains the CLIB specification. In most cases, the statement containing the CLIB specification is the last statement processed for the current load module from the source that contains the CLIB specification. The only exception to this is if the source specified by the CLIB specification contains only group 1 control statements. If the linkage editor control statements are being input from the primary control stream when the CLIB specification is processed, the current location in the control stream is saved and is returned to when the control statements in the specified source are exhausted. In contrast, if a CLIB specification is processed while in a source library, the source library is disconnected and can never be returned to. Thus, it can be seen that multiple CLIB specifications can be meaningfully specified only in the primary control stream because only the first CLIB specification in a source module will ever be processed.

A single link-edit job step that produces multiple load modules proceeds as follows:

1. Enter mode 1 and process group 1 control statements for the first load module to be produced.

2. Enter mode 2 and process all group 2 control statements to produce the first load module.

3. Reenter mode 1 and process group 1 control statements for the second load module.

4. Reenter mode 2 and process group 2 control statements to produce the second load module.

5. Repeat steps 3 and 4 until all load modules are produced.

6.6. Control Statement Descriptions

The following detailed descriptions of the basic linkage editor control statements are presented in their logical order of appearance in a linkage editor control stream. Each description includes the function of the control statement, illustrates its coding format, describes its parameters, and, when necessary, provides illustrated examples of how it may appear in a control stream.

6.6.1. Specify Linkage Editor Options (// PARAM or LINKOP)

Function

Specifies the linkage editor options that are to be in effect during construction of a load module. The options are declared as keywords in the operand field of either a // PARAM or LINKOP control statement. Both control statements perform the same function; however, the // PARAM statement cannot appear within a linkage editor control stream and the LINKOP statement cannot appear outside the linkage editor control stream. Thus, the initial linkage editor options may be specified either by a // PARAM or LINKOP control statement, but only the LINKOP statement can be used to change them within a job step. Each time a LINKOP control statement is processed by the linkage editor, it initiates the construction of a new load module.

All option specifications take effect as soon as they are detected, and remain in effect until changed by a succeeding LINKOP control statement or until the job is terminated. This applies to system-supplied (default) parameter specifications as well as user-supplied specifications. Once a default specification is overriden by the user, the user-supplied specification remains in effect for the remainder of the job step unless the default specification is explicitly respecified by the user. When conflicting specifications are detected, the linkage editor assumes that the last statement is correct and functions accordingly.

If no // PARAM or LINKOP control statements are present in a control stream, the linkage editor functions under the direction of the default parameter specifications, as follows:

1. Performs automatic inclusion of required object modules, as necessary, and assumes that the standard system object library file (YOBJ) is the only file that might contain the required modules.

2. Processes V-CON references, as required.

3. Stores the output load modules in the system job run library file (YRUN).

4. Uses only the control statements contained in the primary control stream to produce load modules.

5. Generates phase header records that contain blank comment fields and do not require the system loader to clear main storage before the phase is loaded.

6. Processes all the control statements contained in the control stream, even if processing errors that would render a load module useless are detected.

7. Provides for the promotion of common storage areas.

8. Generates a link-edit for each load module generated that lists all the information normally desired in the link-edit map.

9. Produces nonreentrant load modules.

10. Recognizes reentrant object modules and creates the shared records needed to link their load module counterparts with the load module being created.

11. Creates internal symbol dictionary (ISD) records in the load module. At execution time, these records enable JOBDUMP to print a formatted dump of the load module area.

Format

LABEL	ΔOPERATIONΔ	OPERAND
[//]	PARAM	[ALIB=filename]
	LINKOP	[,CLIB=modulename/filename]
		[,CMT={'character-string '|ΔΔ}]
		[,OUT={filename|(N)|YRUN}]
		[,RLIB={filename|YOBJ}]
		[,{CNL|NOCNL}] [,{ZRO|NOZRO}]
		[,{NOAUTO|AUTO}] [,{NOV|V}] [,{NOPROM|PROM}]
		[,{NOLIST|LIST}] [,{NOCNTCD|CNTCD}] [,{NOERR|ERR}]
		[,{NODICT|DICT}] [,{NODEF|DEF}] [,{NOPHS|PHS}]
		[,{DEL|NODEL}] [,{RCNTCD|NORCNTCD}] [,{REF|NOREF}]
		[,{SHARE|NOSHARE}] [,{RNT|NORNT}] [,{ISD|NOISD}]

Notes:

1. *System-supplied parameters are in effect only until changed by the user. Once changed within a job step, they must be reset to their default state by the user before the default option is effected again.*

2. *Keywords may be specified in any order, but they must be separated by commas with no spaces between the keywords unless they identify a delimited character string.*

3. *Private file names that are declared are always logical and must be accompanied by the appropriate job control DVC/LFD statements.*

Label

//
Must be specified if PARAM operation is used.

Keyword Parameter ALIB

ALIB=filename
Specifies the file to be searched during automatic inclusion processing. This keyword essentially provides the capability of naming an additional library for use during automatic inclusion processing. When specified, the ALIB library always is searched first in an attempt to locate a required object module, and, if necessary, the file identified by the RLIB keyword is searched.

If omitted, only the RLIB-specified file is searched during the automatic inclusion process.

Keyword Parameter CLIB

CLIB=modulename/filename
Specifies the name of a source module, and the file in which it resides, that contains the linkage editor control statements to be processed for this link-edit job. When this parameter is detected in the primary control stream input, the linkage editor branches to the source module identified in this parameter, processes the control statements contained in the source module, then returns to the primary control stream to complete the processing of the remaining control statements, as applicable. When this parameter is detected in a source module control stream input, the linkage editor branches to the new source module identified in this parameter, processes the control statements contained in the new source module, and returns to the primary control stream. The linkage editor never returns to a source module control stream after processing the first CLIB keyword specification it detects in that control stream source.

If omitted, the current control stream source (primary input or source module input) continues to be accessed for control statements.

Keyword Parameter CMT

`CMT='character-string'`
Specifies a character string of up to 30 characters that is to be inserted in the comment field of each phase header record produced for the generated load modules. The character string must be enclosed in apostrophes and may contain blanks, commas, and other special symbols.

`CMT='Δ'`
Specifies that the phase header comment fields are to be blank.

If omitted, the phase header comment fields are left blank unless a previous CMT keyword specification specified otherwise.

Keyword Parameter OUT

`OUT=filename`
Specifies a library filename in which the output load module is to be stored. If a load module with the same name already exists in the output file specified by this keyword, it is replaced by the new load module. This filename corresponds to the LFD name assigned to the file.

`OUT=(N)`
Indicates that the output load modules produced by the linkage editor are not to be stored in any library file; however, the link-edit map will still be produced if it is not inhibited by the specification of the NOLIST keyword parameter.

`OUT=$Y$RUN`
Specifies that the output load modules produced by the linkage editor are to be stored in the standard system job run library file (YRUN).

If omitted, the load modules produced are output to the system job run library file (YRUN) unless a previous OUT keyword specified otherwise.

Note: *The file designated to store the output of the linkage editor is logically extended to accommodate the load modules produced by the linkage editor.*

Keyword Parameter RLIB

`RLIB=filename`
This parameter is used to identify the file to be searched by the linkage editor, under the following conditions:

- During the automatic inclusion process, when no automatic INCLUDE library file (ALIB) has been specified (no previous ALIB keyword specification present) or when the specified ALIB does not contain the required module

- During the specific inclusion process, when no file name is specified on an INCLUDE statement and either of the following conditions exist:
  - The module is not found in the standard job run library file (YRUN).
  - The module is not found in the file last specified on a prior INCLUDE statement or no prior INCLUDE statements exist.

RLIB=YOBJ
: Identifies the standard system object library file (YOBJ) as the file to search under the conditions just described.

If omitted, the standard system object library file (YOBJ) is searched under the preceding conditions, unless a previous RLIB keyword specified otherwise.

Keyword Parameters CNL and NOCNL

CNL
: Specifies that the link-edit job is to be canceled at the end of the link-edit operation for the current load module if any diagnostic processing has been triggered during the generation of the load module; otherwise, processing continues, regardless of the number or type of error diagnostics triggered, until the normal end-of-job function for the link-edit job step is detected.

NOCNL
: Specifies that the link-edit job step is to be processed to completion, regardless of the number or type of diagnostic errors detected.

If omitted, NOCNL is assumed unless the CNL keyword was previously specified.

Keyword Parameters ZRO and NOZRO

ZRO
: Specifies that the job region is to be cleared to binary zeros prior to loading the root phase of the load module. This option is effective only if this load module name is specified on the // EXEC job control statement. Specification of this keyword sets a flag in the root phase header record to indicate the clearing requirement to the system loader.

NOZRO
: Specifies that main storage is not to be cleared to zeros before the root phase of the load module is loaded.

If omitted, NOZRO is assumed unless the ZRO keyword was previously specified.

Keyword Parameters AUTO and NOAUTO

AUTO
: Specifies that automatic inclusion processing is to be allowed for the load modules being constructed.

NOAUTO
: Specifies that automatic inclusion processing is not to be allowed for the load modules being constructed.

If omitted, AUTO is assumed unless the NOAUTO keyword was previously specified.

Note: The NOAUTO specification does not affect the automatic inclusion of the overlay control routine if V-CON processing is not inhibited (NOV keyword not specified) and valid V-CON references exist in the object code being included in the load modules being produced.

Keyword Parameters V and NOV

V
: Specifies that V-CON references are permitted to appear in the object module elements to be included in the load modules and, therefore, that automatic loading of required phases is to be enabled.

NOV
: Specifies that V-CON references are to be treated as A-CON references and, therefore, that automatic loading of phases is to be inhibited because it is not required. This specification is significant only when valid V-CON references are present in the object module elements being included.

If omitted, V is assumed unless the NOV keyword was previously specified.

Keyword Parameters PROM and NOPROM

PROM
: Specifies that common storage areas are to be promoted to the most reasonable phase within the load module being constructed.

NOPROM
: Specifies that all common storage areas are to be placed in the root phase of the load module being constructed.

If omitted, PROM is assumed unless the NOPROM keyword was previously specified.

Keyword Parameters LIST and NOLIST

LIST
: Specifies that all list options for the link-edit map are to be in effect for the current link-edit job.

NOLIST
: Specifies that no link-edit map is to be produced for the current link-edit job.

If omitted, the standard link-edit map will be produced unless LIST or NOLIST was previously specified.

Keyword Parameters CNTCD and NOCNTCD

CNTCD
: Specifies that the process map portion of the link-edit map is to be produced.

NOCNTCD
: Specifies that the process map is to be suppressed.

If omitted, CNTCD is assumed unless NOCNTCD was previously specified.

Keyword Parameters ERR and NOERR

ERR
: Specifies that diagnostic messages and unresolved references are to be included in the link-edit map.

NOERR
: Specifies that this information is to be suppressed.

If omitted, ERR is assumed unless NOERR was previously specified.

Keyword Parameters DICT and NODICT

DICT
: Specifies that the definitions dictionary and phase structure diagram portions of the link-edit map are to be produced.

NODICT
: Specifies that the definitions dictionary and phase structure diagram portions of the link-edit map are to be suppressed.

If omitted, DICT is assumed unless NODICT was previously specified.

Keyword Parameters DEF and NODEF

DEF
: Specifies that object module headers (including dates and times) and CSECT, COM, and ENTRY items are to be listed in the allocation map portion of the link-edit map along with their respective object module origin and ESID, length, linked base address, and high limit after linking.

NODEF
: Specifies that listing of ESDs is to be suppressed.

If omitted, DEF is assumed unless NODEF was previously specified.

Keyword Parameters PHS and NOPHS

PHS
: Specifies that phase data (phase name, alias-phasename, origin, length, transfer address, etc.) is to be listed in the allocation map portion of the link-edit map.

NOPHS
: Specifies that the phase data is to be suppressed.

If omitted, PHS is assumed unless NOPHS was previously specified.

Keyword Parameters DEL and NODEL

DEL
: Specifies that all definitions, including those automatically deleted due to redundant inclusions on identical paths or items not included because of partial INCLUDE specifications, are to be listed in the allocation map so long as definitions are being listed (NODEF is not specified.).

NODEL
: Specifies that automatically deleted or excluded definitions are not to be listed.

If omitted, NODEL is assumed unless DEL was previously specified in the control stream.

Keyword Parameters RCNTCD and NORCNTCD

RCNTCD
: Specifies that control statements are to be included in the allocation map portion of the link-edit map. This allows the user to easily locate a particular control statement and see its effect on the load module. Only action-type control statements are included in the allocation map; // PARAM and LINKOP statements are never listed here.

NORCNTCD
Specifies that control statements are not to be listed in the allocation map.

If omitted, NORCNTCD is assumed unless RCNTCD was previously specified.

Keyword Parameters REF and NOREF

REF
Specifies that the allocation map portion of the link-edit map is to list all references (EXTRNs) and transfer records processed, the object module assigned address and ESID, and the resolved value, if appropriate.

NOREF
Specifies that this information is to be suppressed.

If omitted, NOREF is assumed unless REF was previously specified.

Keyword Parameters SHARE and NOSHARE

SHARE
Specifies that object modules marked reentrant are to be recognized during the inclusion process (specific and automatic). The text from such modules is not to be included. Instead, shared records are to be created for references made to reentrant code. The load module produced is nonreentrant, but may contain references to reentrant code. This keyword is ignored if a reentrant load module is being generated (the RNT keyword is specified) or the job control GO option is in effect.

NOSHARE
Specifies that all object modules are to be treated as nonreentrant and no shared records are to be generated. A nonreentrant load module is produced that contains no references to shared code.

If omitted, SHARE is assumed unless the NOSHARE keyword was previously specified.

Keyword Parameters RNT and NORNT

RNT
Specifies that a reentrant load module is to be produced and SENTRY records are to be generated. Normally, only one object module would be included in the link-edit, in which case, the load module name must be the same as the object module name. Parameter NOAUTO should be specified when using the RNT parameter.

NORNT
Specifies that a nonreentrant load module is to be produced although references may be made to reentrant code.

If omitted, NORNT is assumed unless RNT keyword was previously specified.

Keyword Parameters ISD and NOISD

ISD

Specifies that Type 3 and Type 4 ISD records from the included object module are relocated and passed to the load module. It also specifies that Type 1 and Type 2 ISD records are generated based on the undeleted CSECTs and COMMONs detected during the link-edit. If execution of the load module terminates, these records are used by JOBDUMP (if specified) to print a formatted dump; segmenting it by CSECTs and printing user program symbols.

NOISD

Specifies that ISD record generation is to be suppressed.

If omitted, ISD is assumed unless NOISD keyword was previously specified.

6.6.2. Begin Load Module (LOADM)

Function

This control statement is used to initiate construction of a load module. Also, it specifies a program name for the load module. The LOADM control statement is normally the first control statement in a control stream, but it may follow a LINKOP or comment statement, or a complete set of previous control statements when used in a control stream that is generating more than one load module.

If the LOADM control statement is omitted from an otherwise valid linkage editor control stream and no LOADM control statement is embedded in an included object module, by default, the load module produced by the linkage editor is assigned the name LNKLOD.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	LOADM	[{name / LNKLOD}]

Positional Parameter 1

name

One to six (eight, if it is the link-edit of a reentrant module) alphanumeric characters, the first of which must be alphabetic, that specifies the name to be assigned to the load module. If the specified name is less than six (eight, if it is the link-edit of a reentrant module) characters long, it is padded on the right with EBCDIC zeros. If the specified name is more than six (eight, if it is the link-edit of a reentrant module) characters long, it is truncated to the maximum allowable limit.

If omitted, the default name LNKLOD is assigned to the load module.

6.6.3. Include Object Code (INCLUDE)

Function

Requests that a specific object module or selected control sections of a specific object module be included in the current phase of the load module being constructed. The INCLUDE statement may follow any linkage editor statement except the ENTER statement. Also, it may be embedded in an object module. Nesting of embedded INCLUDE statements may continue indefinitely. Nested INCLUDE statements are identical in format and capability to those not nested and may thus specify full or partial inclusion, as well as alternate files. The same applies during the automatic inclusion process, although the initial module located and used to satisfy a specific reference always is included in full.

If no INCLUDE statements are present in an otherwise valid linkage editor control stream, all the object modules currently residing in the system job run library (YRUN) are included in the load module phase being constructed.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	INCLUDE	[modulename][s1,...,s9)][,filename]

Positional Parameter 1

modulename
: Is a one- to eight-character alphanumeric character string that identifies the object module to be included.

If omitted, it is assumed that the INCLUDE statement is nested and that the object module being referenced immediately follows the previously referenced object module in the library being accessed. Otherwise, an error message is output on the link-edit map.

Subparameters

s1,...,s9
: Is a list of from one to nine control section labels (one to eight characters long) that identify the control sections to be included in the load module phase being constructed. The control sections referenced in this parameter must be contained in the object module referenced by the INCLUDE statement; otherwise, an error diagnostic is output on the link-edit map. The order in which the control sections appear in the object module is the order in which they will appear in the load module, regardless of the order in which they are listed in this parameter.

When a subparameter list is specified, no unnamed CSECT in the module being scanned is ever included in the load module, but all requests for common storage are honored, as are all embedded control statements.

If omitted, the entire object module is included in the load module except for those control sections that may be automatically deleted by the linkage editor.

Positional Parameter 2

`filename`
Is a one- to eight-character alphanumeric character string that identifies the symbolic name of the file in which the referenced object module is stored. This name must be specified exactly as it is specified in the job control logical file definition (LFD) that identified the file; otherwise, an error message is output on the link-edit map. When this parameter is specified, it is the only file searched for the specified object module.

If omitted, the object module is assumed to be in the system job run library (YRUN) or, if not there, in the last library specified in an INCLUDE statement or, if not there and a file name was specified in a previous RLIB keyword in that file; otherwise, if no RLIB library was specified, the object module is assumed to be in the default RLIB library, the system object library file (YOBJ).

6.6.4. Begin Overlay Phase (OVERLAY)

Function

Identifies the beginning of an overlay phase (a phase other than the initial phase) and defines the relative position of the phase within the load module structure. The object modules included thereafter constitute a single, separate phase until the next OVERLAY, REGION, LOADM, LINKOP, or end-of-data (/*) control statement is detected.

This statement must not be used in the link-edit of a reentrant module.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	OVERLAY	symbol[,alias-phasename]

Positional Parameter 1

`symbol`
Is the name of a logical or relative node point that defines the starting address of the phase. It may consist of one to eight alphanumeric characters. If the symbol is relative (a CSECT, ENTRY, or EQU name), it must be on the path of the phase. However, the relative symbol must not be a shared definition.

The symbol specified may cause the phase to begin at an origin newly established by this node name or may set that origin to an address defined by a previous OVERLAY statement, the LOADM statement, or a relative definition (provided it is on the current path).

The starting address of a phase is not necessarily the entry point to that phase; therefore, use of the symbol in the operand field of an OVERLAY statement does not automatically define the symbol as an entry point. The same symbol, however, may be used to define both the entry point to this phase and the logical (or relative) origin of the phase (if in the current path). If the symbol is on an exclusive path, a logical node is established.

Positional Parameter 2

`alias-phasename`
Is a one- to six-character user-supplied phase name that can be used in placed of the linkage-editor-supplied phase name, to address the phase being created by the OVERLAY statement. If an alias phase name longer than six characters is supplied, it is truncated to six. Alias-phasenames are always padded with two trailing blanks.

6.6.5. Begin New Region (REGION)

Function

Initiates construction of the first phase in a new region, starting at the end of the longest path currently constructed for the load module. Once a region has been started, no prior region structure may be continued. Thus, all parts of a given region should be fully specified and structured before beginning a new region. The only phase common to all regions in a load module is the root phase.

OVERLAY control statements may be interspersed among REGION control statements to structure the phases within each region. The first of any region, including the initial phase, may be overlaid by referencing the region node (or LOADM node) or a symbol with that address, using an OVERLAY control statement. Inasmuch as the REGION statement effectively replaces an OVERLAY statement, INCLUDE and other control statements used for the construction of a phase follow immediately. Up to 10 regions may be declared for a single load module.

This statement must not be used in the link-edit of a reentrant module.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	REGION	symbol[,alias-phasename]

Positional Parameters 1 and 2

Refer to parameter descriptions for the OVERLAY control statement. Note that inasmuch as the implied origin of a new region always is at the end of the longest path of the current region, a symbol in a REGION statement may specify only a logical node name and not a relative definition name as in the case in an OVERLAY statement.

6.6.6. Define Phase Execution Entrance (ENTER)

Function

Defines the program entry point of the load module phase being constructed. This is the address to which program control is optionally transferred when the phase is loaded by a supervisor FETCH macroinstruction. The ENTER statement, if present, is normally the last control statement issued for a given phase. It may, however, be followed by a MOD, EQU, RES, or comment statement in addition to an OVERLAY statement for the next phase, a REGION statement for a new region, a LOADM or LINKOP statement for a new load module, or an end-of-data (/*) statement, which terminates execution of the linkage editor.

If an INCLUDE statement immediately follows an ENTER statement, a diagnostic warning indicating the sequence error is listed on the link-edit map, but the INCLUDE statement will, nonetheless, be processed for the current phase.

If no ENTER statement is provided for a phase, the phase entry point, or transfer address as it is commonly referred to, is obtained from the first specifically included object module in the phase that has a valid transfer address. If no specifically included object module contains a valid transfer address, the entry point address used by the linkage editor is the relocated address assigned to the first CSECT specifically included in the phase. Automatically included modules are not checked for valid transfer addresses. If no CSECTs have been included in the phase (zero length phase), then the transfer address is assigned to the node point of the phase.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	ENTER	[expression]

Positional Parameter 1

expression

Specifies the transfer address for the phase. This expression, which usually represents a relative phase address, may have one of the following forms:

- A decimal number from one to eight digits long.

- A hexadecimal number from one to six digits long, in the form X'nnnnnn'.
- A previously defined symbol (the name of a control section or an entry point in an object module that was previously included or defined by a previous EQU directive). For the link-edit of a nonreentrant module, this symbol must not be a shared definition.
- A previously defined symbol plus or minus a decimal or hexadecimal number as previously described.

If omitted, the ENTER statement has no effect and the default criteria previously described apply.

6.6.7. Define Label (EQU)

Function

The EQU control statement is used to provide the linkage editor with the value of a label that might not otherwise be defined. The definition of a symbol by an EQU statement is subject to the same rules for automatic deletion as entry points.

Format

LABEL	ΔOPERATIONΔ	OPERAND
symbol	EQU	expression

Label

symbol
Is a one- to eight-character alphanumeric character string, which is the label to be defined.

Positional Parameter 1

expression
Specifies the value to be assigned to the label. The expression may have one of the following forms:

- A decimal number from one to ten digits long.
- A hexadecimal number from one to eight digits long.
- A previously defined label (the name of a control section or an entry point in an object module that was previously included or defined by a previous EQU statement).

- A previously defined label plus or minus a decimal or hexadecimal number, as previously described. For the link-edit of a nonreentrant module, this label must not be a shared definition.
- An asterisk (*) to indicate a reference to the current value of the location counter.

Examples

```
      1         10    16
1.  DDSAC     EQU   32
2.  DDECT1    EQU   X'20'
3.  DPSCHK1   EQU   DDECT1
4.  DPSCHK2   EQU   DDECT1     6X'20'
5.  DPSCHK3   EQU   *
```

1. Equates the label DDSAC to the decimal value 32.
2. Equates the label DDECT1 to the hexadecimal value 20.
3. Equates the label DPSCHK1 to the value of the label DDECT1.
4. Equates the label DPSCHK2 to the value of the label DDECT1 plus the hexadecimal value 20.
5. Equates the label DPSCHK3 to the current value of the location counter.

Notes:

1. *If the operand of the EQU statement is symbolic (a previously defined symbol is involved) and the label supplies a multiple definition, then no such earlier definitions may occur between the definition of the operand symbol and the EQU statement itself. If a symbolic operand is multiply defined, the label of the EQU statement is equated to the last such definition.*
2. *KE$ALP and KE$RES, which are the addressable entry points to the reserve storage area, cannot be used as operands of the EQU control statement.*

6.6.8. Modify Location Counter (MOD)

Function

The MOD control statement instructs the linkage editor to adjust its location counter to coincide with a specific power of 2 and a specific remainder. Initially, the location counter is set to zero by the LOADM control statement, and then incremented by the length of each control section included in a phase. Also, it is set to the value associated with a node point when an OVERLAY or REGION control statement is detected in the control stream.

Format

```
LABEL | ΔOPERATIONΔ | OPERAND
      | MOD         | power[,{remainder}]
                           [{(0)      }]
```

Positional Parameter 1

power

Is a decimal or hexadecimal (X'n') number that specifies the power of 2 relative to which the location counter is to be adjusted. The only acceptable powers of 2 that may be specified are 8, 16,32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32,768.

Positional Parameter 2

remainder

Is a decimal or hexadecimal (X'n') number that is a multiple of 4, which specifies the desired remainder of the new value of the location counter relative to the specified power of 2. If the decimal number specified for this parameter is not a multiple of 4, it is rounded to the next higher multiple of 4, then truncated to a value less than the specified power of 2.

If omitted, the remainder is assumed to be zero.

Example

```
1         10    16
|         MOD   32,8
```

This control statement instructs the linkage editor to adjust its location counter, if necessary, to a value that is 8 more than a multiple of 32.

- If the current value of the location counter is 16,384, which is an exact multiple of 32 (32 x 512 = 16,384), the location counter would be incremented by 8 to a value of 16,392.
- If the current value of the location counter were 16,392, no adjustment would be required.
- If the current value were 16,400, which is 16 greater than a multiple of 32 (32 x 512 + 16 = 16,400), the new value would be adjusted to 16,424 (32 x 513 + 8 = 16,424).

6.6.9. Reserve Storage (RES)

Function

Instructs the linkage editor to reserve additional load module storage space following the end of the longest path in the highest region of the load module. The additional storage requested by the RES statement adds to the total length of the module, and is recorded in each phase header record, but is not included in the size requirements for any particular phase. One or more of these statements, therefore, may be placed anywhere within the control stream for a load module. The sum of all RES values processed during the construction of a load module is used to extend the longest path of the load module.

The linkage editor automatically assigns two addressable entry points to the reserve storage area. These entry points may be addressed by the user to access the reserve storage area by declaring them as EXTRNS in your program. The two addressable entry points assigned are:

- KE$ALP - which is the effective address of the end of the longest path in the load module, or the starting address of the reserve storage area.
- KE$RES - which is the effective address of the end of the longest path plus the sum of all the reserve storage area size specifications (RES statements).

These two entry points cannot be used in the operand field of the EQU control statement.

Format

LABEL	ΔOPERATIONΔ	OPERAND
	RES	value

Positional Parameter 1

value

Specifies the number of bytes of storage to be reserved in one of the following forms:

- Decimal number from one to ten digits long
- Hexadecimal number one to eight digits long, in the form X'nnnnnnnn'

Section 7
Link-Edit Map

7.1. Introduction

Unless otherwise specified in a // PARAM or LINKOP control statement, the linkage editor produces a link-edit map for each link-edit job it performs. Basically, the map details the load module produced by the linkage editor, and is produced in six parts:

1. Process map
2. Unresolved EXTRN reference list
3. Definitions dictionary
4. Phase structure diagram for multiphase load modules
5. Allocation map
6. Error legend and count list

However, via // PARAM or LINKOP parameter specifications, all or part of the link-edit map may be suppressed. A detailed description of each map part follows.

7.2. Process Map

The initial part of the link-edit map is a listing of the linkage editor control stream used to produce the subject load module (see Figure 7-1). Both the control statement specifications listed by the user and those inserted by the linkage editor in the control stream from referenced source and object modules are listed. Special process map messages also may appear in the process map. These messages, which are always enclosed in asterisks, are used to explain the presence of some of the control statement data included in the process map. The special messages are listed and described in Table 7-1.

```
UNISYS SYSTEM OS/3 LINKAGE EDITOR
DATE- 88/05/02  TIME- 11.45

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB=OBJMOS
            // PARAM OUTPUT=NO
            /*
                     LOADM      CFLNKJSS
                     INCLUDE    LKSASS,OBJMOS
-----K019-INCLUDE MODULE NOT LOCATED
                     INCLUDE     LKIROOT,OBJMOS
                     OVERLAY                 A1
                     INCLUDE    AJ1@AJ101JC,AJ102JC,AJ103JC,AJ104JC@,OBJMOS
                     INCLUDE     AO2@@AO2030C@,OBJMOS
                     INCLUDE     AC3,OBJMOS
                     INCLUDE     AD1@@AC1@,OBJMOS
                     INCLUDE     AO2@@AO2090C,AO2030C,AO2020C,AO2010C,AO2@,OBJMOS
            /*
AO2060C    *AUTO-INCLUDED*
CLSOCP     *AUTO-INCLUDED*
```

Figure 7-1. Typical Link-Edit Process Map Listing

Table 7-1. Special Process Map Messages

Message	Meaning
AUTO-INCLUDED	Item was, of necessity, automatically included.
EMBEDDED	Control statement, as processed, exists within an included object module.
GENERATED	A necessary control statement was generated by the linkage editor.
SORS FILE	Control statement processed resides within a source module of linkage editor directives.
RUN LIBE MODULE	Module was included by default from the job run library (/* item is not printed on process map).

Diagnostic warnings also may be interspersed among in-error control statements and the processing triggered by their presence. These messages always are delineated by five leading dashes (- - - - -) and an error code. They are listed and described in the *System Messages Reference Manual* (UP-8076).

The process map may be suppressed by the declaration of the NOCNTCD keyword specification in the // PARAM or LINKOP control statement.

7.3. Unresolved EXTRN Reference List

If, after all INCLUDE processing terminates, references are made to one or more symbols for which no corresponding definitions exist, a list of the undefined symbols is output on the link-edit map and the UPSI byte is set to X'20'. The symbols are not sorted into any sequence and are listed only for the convenience of the user (see Figure 7-2). After this list is generated and all CSECT, relocation, and phase sizes, and address assignments are computed, diagnostic messages may be interspersed among the list of undefined EXTRN references, as applicable.

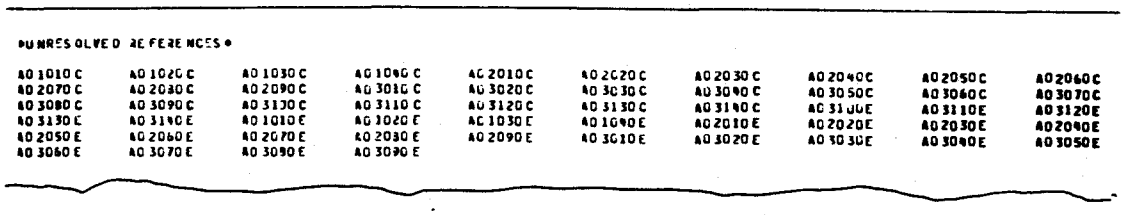


Figure 7-2. Typical Unresolved EXTRN Reference List

7.4. Definitions Dictionary

The definitions dictionary part of the link-edit map lists and describes all the symbols referenced in the link-edit job. These symbols are listed alphabetically, in three horizontal columns (see Figure 7-3). Each symbol definition includes

- Type identification
- Phase assignment identification for nonshared items
- Linkage-editor-assigned address when applicable
- Optional information character

The type identifications that may be specified for a symbol and their meanings are listed in Table 7-2.

The phase assignment identification identifies the load module in which the symbol appears. The phase identifications that may appear in this field are listed and described in Table 7-3.

DEFINITIONS DICTIONARY

SYMBOL.	TYPE.	PHASE.	ADDRESS.	SYMBOL.	TYPE.	PHASE.	ADDRESS.	SYMBOL.	TYPE.	PHASE.	ADDRESS.
A01010C	EXTRN	--*V*	--------	A01010E	EXTRN	--	--------	A01C20C	EXTRN	--*V*	--------
A01020E	EXTRN	--	--------	A01030C	EXTRN	--*V*	--------	A01030C	EXTRN	--	--------
A01040C	EXTRN	--*V*	--------	A01040E	EXTRN	--	--------	A02010C	EXTRN	--*V*	--------
A02010E	EXTRN	--	--------	A02020C	EXTRN	--*V*	--------	A02G20Z	EXTRN	--	--------
A02030C	EXTRN	--*V*	--------	A02030E	EXTRN	--	--------	A02040C	EXTRN	--*V*	--------
A02040E	EXTRN	--	--------	A02050C	EXTRN	--*V*	--------	A02050E	EXTRN	--	--------
A02060C	EXTRN	--*V*	--------	A02C60E	EXTRN	--	--------	A02G70C	EXTRN	--*V*	--------
A02070E	EXTRN	--	--------	A02080C	EXTRN	--*V*	--------	A02G80E	EXTRN	--	--------
A02090C	EXTRN	--*V*	--------	A02C90E	EXTRN	--	--------	A03010C	EXTRN	--*V*	--------
A03010E	EXTRN	--	--------	A03G20C	EXTRN	--*V*	--------	A03G20E	EXTRN	--	--------
A03030C	EXTRN	--*V*	--------	A03G30E	EXTRN	--	--------	A03040C	EXTRN	--*V*	--------
A03040E	EXTRN	--	--------	A03C50C	EXTRN	--*V*	--------	A03050E	EXTRN	--	--------
A03060C	EXTRN	--*V*	--------	A03060E	EXTRN	--	--------	A03070C	EXTRN	--*V*	--------
A03070E	EXTRN	--	--------	A03080C	EXTRN	--*V*	--------	A03G80E	EXTRN	--	--------
A03090C	EXTRN	--*V*	--------	A03C90E	EXTRN	--	--------	A03110C	EXTRN	--*V*	--------
A03120E	EXTRN	--	--------	A03110C	EXTRN	--*V*	--------	A03110E	EXTRN	--	--------
A03120C	EXTRN	--*V*	--------	A03120E	EXTRN	--	--------	A03130C	EXTRN	--*V*	--------
A03130E	EXTRN	--	--------	A03140C	EXTRN	--*V*	--------	A03140E	EXTRN	--	--------
A0501C	CSECT	01	00003358	A0501E	ENTRY	01	0000335E	A0502C	CSECT	02	00000368
A0502E	ENTRY	02	00003372	A0503C	CSECT	03	00003378	A0503E	ENTRY	03	00000386
A0504C	CSECT	04	00003390	A0504E	ENTRY	04	000033A2	A0505C	CSECT	05	000003A8
A0505E	ENTRY	05	000033BE	A0506C	CSECT	06	000033C8	A0506E	ENTRY	06	000003E2
A0507C	CSECT	07	000033E8	A0507E	ENTRY	07	00003406	A0508C	CSECT	08	00000410
A0508E	ENTRY	08	00003432	A0509C	CSECT	09	00003438	A0509E	ENTRY	09	0000345E
A0510C	CSECT	10	00003468	A0510E	ENTRY	10	00003492	A0511C	CSECT	11	00000498
A0511E	ENTRY	11	000034C6	A0512C	CSECT	12	000034D0	A0512E	ENTRY	12	00000502
A0513C	CSECT	13	00003508	A0513E	ENTRY	13	0000353E	A0514C	CSECT	14	00000358
A0514E	ENTRY	14	00003392	*E$ALP	ENTRY	ABS	00003548	*ESRES	ENTRY	ABS	00000548
LK1R00T	CSECT	R00F	00003000								

Figure 7-3. Typical Link-Edit Definitions Dictionary List

Table 7-2. Definitions Dictionary Type Identifications

Type	Meaning
COM *	Identifies a common section name.
ENTRY	Identifies an absolute or relative entry point definition.
CSECT *	Identifies a control section name.
EQU	Identifies a link-time-defined symbol for an entry point.
EQU*	Identifies a link-time reference to the location counter for an entry point.
EXTRN	Identifies a symbol referenced but never defined (also listed in the unresolved EXTRN references list).

* If a COM or CSECT symbol name is blank, the item listed represents a blank common or unnamed control section, respectively.

Table 7-3. Definitions Dictionary Phase Field

Phase	Meaning
1-99	Load module phase number
ROOT	Root phase
ABS	Absolute ENTRY symbol (no phase assignment applicable)
--	EXTRN symbol (no phase assignment applicable)
SHR	Shared definition

The address field is expressed in hexadecimal and may be absolute or relocatable, depending on the definition type. Dashes appear in the address field of EXTRN symbols. Blanks appear in the address field of shared definitions since they do not have an address.

The information characters that also may appear in a symbol definition are listed and described in Table 7-4. When included, these characters are listed between the PHASE and ADDRESS fields of a symbol definition.

Table 7-4. Definitions Dictionary Information Characters

Character	Meaning
G	Used to identify a dictionary item referenced by text and included in the load module during a partial INCLUDE sequence, and defined in another CSECT of the same object module that was not to be included in the load module. Whenever such a symbol is detected, the linkage editor generates an EXTRN label for the symbol in hopes of satisfying the reference. No automatic INCLUDE processing ever is triggered for such references and, therefore, the symbol remains undefined unless specifically obtained by the user through an INCLUDE or EQU statement.
M	Used to identify a symbol that is multiply defined and, therefore, listed at least twice.
V	Used to indicate that a symbol is a valid type V definition and is a candidate for residence in the entry point table (KL$NTB) for V references.

Printing of the definitions dictionary can be suppressed by a // PARAM or LINKOP control statement that specifies the keyword NODICT; however, suppression of the definitions dictionary also causes suppression of the phase structure map.

7.5. Phase Structure Diagram

A scaled pictorial drawing for multiphase load modules is provided next in the link-edit map (see Figure 7-4). This figure depicts the phase and overlay structure of the load module being generated. Each phase is shown with its linked load origin in association with other phases. Each phase is identified by a decimal phase number and its size is listed in hexadecimal. Vertical bars (I) represent established node origins; horizontal bars (dashes) represent the phase lengths. Each printer bar is intended to depict a specific number of bytes with regard to phase storage needs and is scaled to a predetermined factor that is specified in the heading of the listing. In Figure 7-4, each horizontal bar represents 0D hexadecimal (13 decimal) bytes.

The phase structure diagram can be suppressed by a // PARAM or LINKOP control statement that specifies the keyword NODICT; however, suppression of the phase structure diagram also causes suppression of the definitions dictionary. In either event, no phase structure diagram is produced for single-phase load modules.

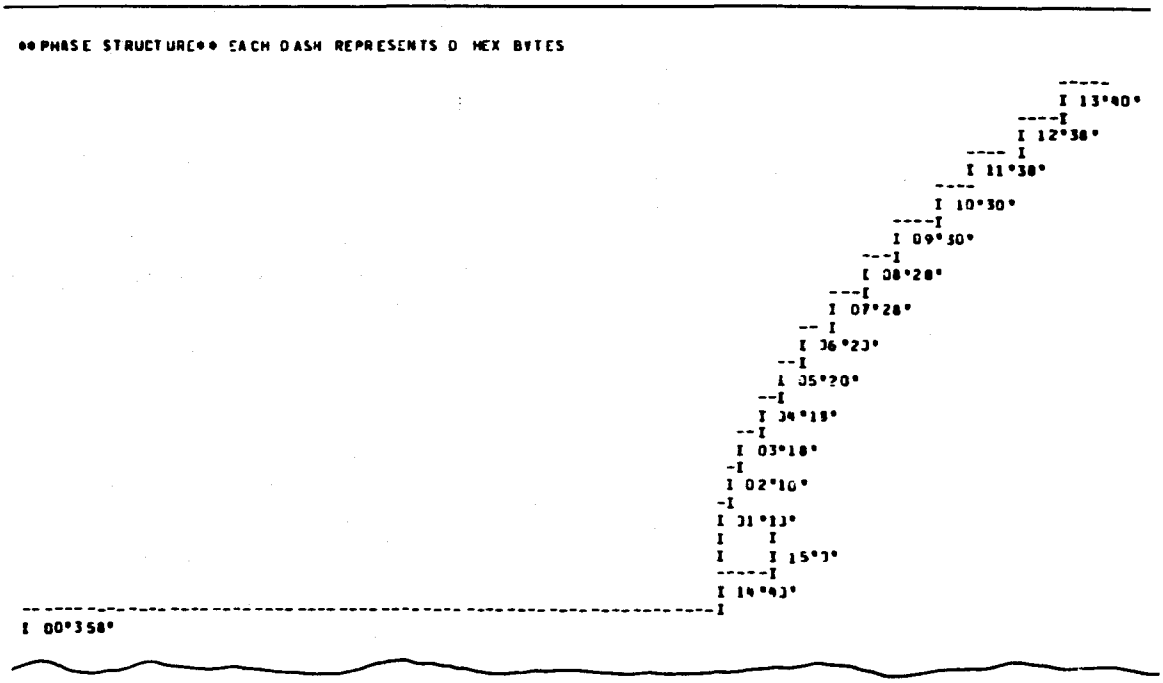


Figure 7-4. Typical Phase Structure Diagram

7.6. Allocation Map

The allocation map provides a detailed description of the items processed during the link-edit operation. The following operation, depending on the // PARAM or LINKOP specified parameters, may include:

- Name and size of the load module.
- Linkage editor assigned name of each phase (and possible alias-phasename), and an indication of any common storage areas assigned to the phase (COMMON indicates residence in current phase).
- Origin, length, and high address of each phase.
- Transfer address assigned to each phase.
- Name, type, and ESID of each definition or reference processed.
- Linked origin, high address, length, and object module origin (if applicable) of each definition or reference processed; certain language processors omit the length field in the CSECT record but indicate it in the transfer record. When the linkage editor lists such a CSECT, the high address is omitted, the word DEFERRED appears under the LENGTH heading, and the code L appears under the FLAG reading. The actual CSECT length is printed when the transfer record is listed. Blanks appear in the link origin high address and length field for shared definitions.
- Set of flag codes for items representing special conditions.
- Relist of interspersed control statements that triggered the results shown.
- Indication of the accessed object modules and whether they were automatically included or not.
- Object module transfer records processed. If the deferred length is present, it is printed under LENGTH and the code L appears under FLAG.

A typical allocation map is illustrated in Figure 7-5.

```
                                        ** ALLOCATION MAP **

                    LOAD MODULE -  SK13000      SIZE -  00000548

PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE      ESID     LNK ORG    HIADDR     LENGTH     OBJ ORG
SK130003        NODE - 9001                                      00003000   00003358   00000358
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS ***
        - 00/01/88 20.44  -       LK1R300 T  OBJ
                                  LK1R300 T  CSECT     02        00003000   00003358   00000358   00000000
              00003000
SK130001        NODE -   NODE31                                  00003358   00003368   00000010
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS01C     CSECT     03        00003358   00003368   00000010   00000008
                                  AOS01E     ENTRY     03        0000335E                         0000000E
              00003358
SK130002        NODE -   NODE32                                  00003368   00003378   00000010
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS02C     CSECT     04        00003368   00003378   00000010   00000018
                                  AOS02E     ENTRY     04        00003372                         00000022
              00003368
SK130003        NODE -   NODE33                                  00003378   00003390   00000018
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS03C     CSECT     05        00003378   00003390   00000018   00000028
                                  AOS03E     ENTRY     05        00003386                         00000036
              00003378
SK130004        NODE -   NODE34                                  00003390   000033A8   00000018
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS04C     CSECT     06        00003390   000033A8   00000018   00000040
                                  AOS04E     ENTRY     06        000033A2                         00000052
              00003390
SK130005        NODE -   NODE35                                  000033A8   000033C8   00000020
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS05C     CSECT     07        000033A8   000033C8   00000020   00000058
                                  AOS05E     ENTRY     07        000033BE                         0000006E
              000033A8
SK130006        NODE -   NODE36                                  000033C8   000033E8   00000020
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS06C     CSECT     08        000033C8   000033E8   00000020   00000078
                                  AOS06E     ENTRY     08        000033E2                         00000092
              000033C8
SK130007        NODE -   NODE37                                  000033E8   00003410   00000028
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS07C     CSECT     09        000033E8   00003410   00000028   00000098
                                  AOS07E     ENTRY     09        00003406                         00000084
              000033E8
SK130008        NODE -   NODE38                                  00003410   00003438   00000028
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS08C     CSECT     0A        00003410   00003438   00000028   000000C0
                                  AOS08E     ENTRY     0A        00003432                         000000E2
              00003410
SK130009        NODE -   NODE39                                  00003438   00003468   00000030
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS09C     CSECT     0B        00003438   00003468   00000030   000000E8
                                  AOS09E     ENTRY     0B        0000345E                         0000010E
              00003438
SK130010        NODE -   NODE10                                  00003468   00003498   00000030
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS10C     CSECT     0C        00003468   00003498   00000030   00000118
                                  AOS10E     ENTRY     0C        00003492                         00000142
              00003468

PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE      ESID     LNK ORG    HIADDR     LENGTH     OBJ ORG
SK130011        NODE -   NODE11                                  00003498   000034D0   00000038
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS11C     CSECT     0D        00003498   000034D0   00000038   00000148
                                  AOS11E     ENTRY     0D        000034C6                         00000176
              00003498
SK130012        NODE -   NODE12                                  000034D0   00003508   00000038
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS12C     CSECT     0E        000034D0   00003508   00000038   00000180
                                  AOS12E     ENTRY     0E        000035D2                         00000182
              000034D0
SK130013        NODE -   NODE13                                  00003508   00003548   00000040
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS13C     CSECT     0F        00003508   00003548   00000040   00000188
                                  AOS13E     ENTRY     0F        0000353E                         000001EE
              00003508
SK130014        NODE -   NODE14                                  00003548   00003598   00000040
        - 00/01/88 06.56  -       AOS        OBJ
                                  AOS14C     CSECT     10        00003548   00003598   00000040   000001F8
                                  AOS14E     ENTRY     10        00003592                         00000232
              00003548
SK130015        NODE -   NODE15                                  00003598   00003598   00000000
-----ROOT- SK10J015  ZERO  LENGTH  PHASE
              00003598
```

Figure 7-5. Typical Allocation Map

7.7. Error Legend, Error Count List, and UPSI Setting

The final part of the link-edit map concerns the termination of the link-edit job itself. An error code legend is printed, indicating the meanings of flags that may have been detected during the link-edit job. The error legend is followed by a printout showing a count of the number of errors appearing in the link-edit map and an UPSI byte setting at the end of the map. The flag code legend is summarized in Table 7-5.

If the link-edit job has been successful, a completion message and the date/time will so indicate. A typical error legend and count list is illustrated in Figure 7-6.

Table 7-5. Error Legend and Count List Flag Code Descriptions

Flag Code	Description
B	The control section is considered a block data CSECT and is to be used to load a common storage area.
D	The item has been automatically deleted because another relative definition for the same symbol is in the same path or one absolute definition already exists. If a transfer record is involved, the item has not been accepted for processing because no valid address is in the record.
E	A direct A-type reference has been made, which is exclusive and whose definition may not be resident with the reference.
G	An EXTRN has been generated because of the partial inclusion of a control section whose text references a control section that is not included.
I	A V-type reference has been made to a definition that is inclusive and, therefore, has been converted to a direct type A reference.
L	A deferred CSECT or COMMON length is in effect, and the actual length is listed with the object module transfer record.
M	The item is validly multiply defined in this link-edit.
N	The item has not been included because of a partial INCLUDE, which purposely omitted it.
P	The item is a COM storage area and has been promoted to another phase based on the phases in which it was declared, the references which point to it, and the block data CSECTS which load it.
R	A shared record has been produced for this item.

continued

Table 7-5. Error Legend and Count List Flag Code Descriptions (cont.)

Flag Code	Description
S	This is a shared item.
U	The item was unresolved and remains undefined in the load module.
V	The item is a V-CON item and will be loaded automatically when referenced by a type V constant.

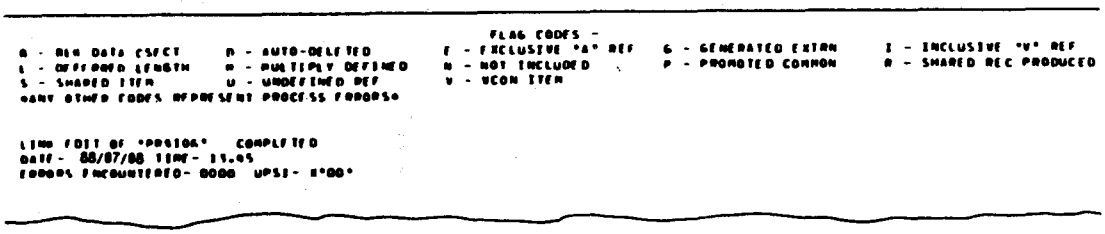


Figure 7-6. Typical Error Legend and Count List

Section 8
Program Examples

A representative set of link-edit jobs is illustrated in this section. Figure 8-1 illustrates a typical job control stream that could be used to execute the linkage editor, while Figures 8-2 through 8-6 illustrate the link-edit maps produced for each job. Figure 8-2 illustrates a rather simple single-phase load module. Figures 8-3 through 8-5 illustrate some typical multiphase load modules, and Figure 8-6 illustrates a rather sophisticated multiphase load module.

```
1         10        16
// JOB LINKSTA
// DVC 20 // LFD PRNTR
// WORK1
// EXEC LNKEDT
/$
** LINKAGE EDITOR CONTROL STATEMENTS GO HERE **
/*
/&
// FIN
```

Figure 8-1. Typical Linkage Editor Job Control Stream

Notes:

1. *When storage is available, the job card should specify a maximum storage size of X '8000' for optimum performance. If you omit the maximum main storage parameter, job control allocates the minimum and maximum main storage requirements to execute the linkage editor. The speed of the linkage editor is directly related to the amount of main storage allocated to the job.*

2. *The printer must always be allocated regardless of the fact that the list options selected may cause only minimal printing.*

3. *If the keyword parameter EXTSP is coded on WORK1 jproc, a minimum value of 5 must be specified.*

4. *In addition to using the / / EXEC LNKEDT statement to call the linkage editor, a job procedure call statement (jproc call) is also available. The jproc call is an easier method for executing the linkage editor. For example, if all the defaults associated with either the / / PARAM or LINKOP statement were utilized, and the only other linkage editor control statement required was the LOADM statement, the following jproc call can be used:*

   / / LINK

   By using this one statement, the following is automatically generated:

   - *Printer assignment*
   - *Work file*
   - *EXEC LNKEDT statement*
   - */ / LOADM statement*
   - *Data delimiter (/$ and /*) job control statements*

   You can alter the default conditions by using the optional parameters. By choosing the optional parameters, you can use a specific printer, a certain scratch file, a specific file containing the load module, etc.

   For more information about the jproc call, refer to the Job Control Language Programming Guide *(UP-9986).*

```
UNISYS SYSTEM OS/3  LINKAGE EDITOR                                              VER770503
DATE-  88/06/09  TIME- 07.33

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

          // PARAM RLIB#OBJMOS
          // PARAM OUT#%N@
          /%
                    LINKOP NOA T                                                JCL00130
                    LOADM      SK1                                              JCL00140
                    ENTER LIB3MOD9                                              JCL00150
 -----K031-CONTROL CARD PLACEMENT ERROR
 -----K005-ENTER OPERAND UNKNOWN
                    INCLUDE    LK1ROOT,OBJMOS                                   JCL00160
 -----K031-CONTROL CARD PLACEMENT ERROR
                    INCLUDE    .01%A01010C,A01020C,A01030C,A01040C@,OBJMOS      JCL00170
                    INCLUDE    A02%A02050C@,OBJMOS                              JCL00180
                    INCLUDE    A03,OBJMOS                                       JCL00190
                    INCLUDE    A01%A01@,OBJMOS                                  JCL00200
                    INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS  JCL00210
                    INCLUDE    A02%A02090C,A02080C,A02070C,A02060C@,OBJMOS      JCL00220
          /*


                                        *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.  PHASE.  ADDRESS.     SYMBOL.   TYPE.  PHASE.  ADDRESS.     SYMBOL.   TYPE.  PHASE.  ADDRESS.

A01       CSECT  ROOT    00000117     A01010C   CSECT  ROOT    00000358     A01010E   ENTRY  ROOT    000003DC
A01020C   CSECT  ROOT    000003E0     A01020E   ENTRY  ROOT    00000468     A01030C   CSECT  ROOT    00000470
A01030E   ENTRY  ROOT    000004F4     A01040C   CSECT  ROOT    00000500     A01040E   ENTRY  ROOT    00000590
A02       CSECT  ROOT    00001180     A02010C   CSECT  ROOT    00001188     A02010E   ENTRY  ROOT    00001220
A02020C   CSECT  ROOT    00000122     A02020E   ENTRY  ROOT    000012C4     A02030C   CSECT  ROOT    000012C8
A02030E   ENTRY  ROOT    00001360     A02040C   CSECT  ROOT    00001370     A02040E   ENTRY  ROOT    00001414
A02050C   CSECT  ROOT    00000590     A02050E   ENTRY  ROOT    00000640     A02060C   CSECT  ROOT    00001418
A02060E   ENTRY  ROOT    000014C4     A02070C   CSECT  ROOT    000014C8     A02070E   ENTRY  ROOT    00001578
A02080C   CSECT  ROOT    00001580     A02080E   ENTRY  ROOT    00001634     A02090C   CSECT  ROOT    00001638
A02090E   ENTRY  ROOT    000016F4     A03       CSECT  ROOT    00000648     A03010C   CSECT  ROOT    00000650
A03010E   ENTRY  ROOT    000006F0     A03020C   CSECT  ROOT    00000700     A03020E   ENTRY  ROOT    00000780
A03030C   CSECT  ROOT    000007B8     A03030E   ENTRY  ROOT    0000086C     A03040C   CSECT  ROOT    00000870
A03040E   ENTRY  ROOT    00000928     A03050C   CSECT  ROOT    00000930     A03050E   ENTRY  ROOT    000009EC
A03060C   CSECT  ROOT    000009F0     A03060E   ENTRY  ROOT    00000AB0     A03070C   CSECT  ROOT    00000AB8
A03070E   ENTRY  ROOT    00000B70     A03080C   CSECT  ROOT    00000B80     A03080E   ENTRY  ROOT    00000C48
A03090C   CSECT  ROOT    00000C50     A03090E   ENTRY  ROOT    00000D1C     A03100C   CSECT  ROOT    00000D20
A03100E   ENTRY  ROOT    00000DF0     A03110C   CSECT  ROOT    00000DF8     A03110E   ENTRY  ROOT    00000FCC
A03120C   CSECT  ROOT    00000ED0     A03120E   ENTRY  ROOT    00000FA8     A03130C   CSECT  ROOT    00000FB0
A03130E   ENTRY  ROOT    00001080     A03140C   CSECT  ROOT    00001090     A03140E   ENTRY  ROOT    00001170
KE$ALP    ENTRY  ABS     000016F4     KE$RES    ENTRY  ABS     000016F4     LK1ROOT   CSECT  ROOT    00000000


                                        ** ALLOCATION MAP **

                          LOAD MODULE -   SK1000          SIZE -   000016F4
```

Figure 8-2. Link-Edit Example 1 (Part 1 of 3)

```
PHASE NAME   TRANS ADDR    FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
SK100000       NODE - ROOT                                        00000000    000016F3    000016F4
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
        -  88/07/02  11.49  -      LK1ROOT     OBJ
                               LK1ROOT     CSECT     01        00000000    00000353    00000354    00000000
        -  88/06/27  12.23  -      A01         OBJ
                               A01010C     CSECT     02        00000358    000003DF    00000088    00000708
                               A01020C     CSECT     1E        000003E0    0000046B    0000008C    00000090
                               A01030C     CSECT     1F        00000470    000004FF    00000090    00000120
                               A01040C     CSECT     20        00000500    00000593    00000094    00000180
                               A01010E       ENTRY   02        000003DC                            0000008C
                               A01020E       ENTRY   1E        00000468                            00000118
                               A01030E       ENTRY   1F        000004FC                            000001AC
                               A01040E       ENTRY   20        00000590                            00000240
        -  88/06/27  12.25  -      A02         OBJ
                               A02050C     CSECT     21        00000598    00000643    000000AC    00000298
                               A02050E       ENTRY   21        00000640                            00000340
        -  88/06/27  12.28  -      A03         OBJ
                               A03         CSECT     01        00000648    00000649    00000002    00000000
                               A03010C     CSECT     02        00000650    000006FF    000000B0    00000008
                               A03020C     CSECT     1E        00000700    00000783    000000B4    000000B8
                               A03030C     CSECT     1F        00000768    0000086F    000000B8    00000170
                               A03040C     CSECT     20        00000870    0000092B    000000BC    00000228
                               A03050C     CSECT     21        00000930    000009EF    000000C0    000002E8
                               A03060C     CSECT     22        000009F0    00000AB3    000000C4    000003A8
                               A03070C     CSECT     23        00000AB8    00000B7F    000000C8    00000470
                               A03080C     CSECT     24        00000B80    00000C4B    000000CC    00000538
                               A03090C     CSECT     25        00000C50    00000D1F    00000000    00000600
                               A03100C     CSECT     26        00000020    00000DF3    00000004    00000608
                               A03110C     CSECT     27        00000DF8    00000ECF    00000008    00000780
                               A03120C     CSECT     28        00000FD0    00000FAB    0000000C    00000888
                               A03130C     CSECT     29        00000FB0    0000108F    000000E0    00000968
                               A03140C     CSECT     2A        00001090    00001173    000000E4    00000A48
                               A03010E       ENTRY   02        000006FC                            00000084
                               A03020E       ENTRY   1E        00000780                            0000016B
                               A03030E       ENTRY   1F        0000086C                            00000224
                               A03040E       ENTRY   20        00000928                            000002E0
                               A03050E       ENTRY   21        000009EC                            000003A4
                               A03060E       ENTRY   22        00000AB0                            00000468
                               A03070E       ENTRY   23        00000B7C                            00000534
                               A03080E       ENTRY   24        00000C48                            00000600
                               A03090E       ENTRY   25        00000D1C                            000006D4
                               A03100E       ENTRY   26        00000DF0                            000007A8
                               A03110E       ENTRY   27        00000ECC                            00000884
                               A03120E       ENTRY   28        00000FA8                            00000960
                               A03130E       ENTRY   29        0000108C                            00000A44
                               A03140E       ENTRY   2A        00001170                            00000B28
        -  88/06/27  12.23  -      A01         OBJ
                               A01         CSECT     01        00001178    00001179    00000002    00000000
        -  88/06/27  12.25  -      A02         OBJ
                               A02         CSECT     01        00001180    00001181    00000002    00000000
                               A02010C     CSECT     02        00001188    00001223    0000009C    00000008
                               A02020C     CSECT     1E        00001228    000012C7    000000A0    000000A8
                               A02030C     CSECT     1F        000012C8    00001368    000000A4    00000148
                               A02040C     CSECT     20        00001370    00001417    000000A8    000001F0
```

Figure 8-2. Link-Edit Example 1 (Part 2 of 3)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
                                      A02010E     ENTRY      02      00001220                            00000DA0
                                      A02020E     ENTRY      1E      000012C4                            00000144
                                      A02030E     ENTRY      1F      00001368                            00000158
                                      A02040E     ENTRY      20      00001414                            00000294
          -  88/06/27  12.25  -       A02        OBJ
                                      A02060C    CSECT       22      00001418    000014C7    00000080    00000348
                                      A02070C    CSECT       23      000014C8    00001578    00000084    000003FA
                                      A02080C    CSECT       24      00001580    00001637    00000088    00000480
                                      A02090C    CSECT       25      00001638    000016F3    0000008C    0000056E
                                      A02060E     ENTRY      22      000014C4                            000003F4
                                      A02070E     ENTRY      23      00001578                            00000488
                                      A02080E     ENTRY      24      00001634                            00000564
                                      A02090E     ENTRY      25      000016F0                            00000620
                  00000000

                                                    FLAG CODES -
 - - RLK DATA CSECT       D - AUTO-DELETED        E - EXCLUSIVE .A. REF    G - GENERATED EXTRN      I - INCLUSIVE .V. REF
 L - DEFERRED LENGTH      M - MULTIPLY DEFINED    N - NOT INCLUDED         P - PROMOTED COMMON      R - SHARED REC PRODUCED
   - SHARED ITEM          U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*

LINK EDIT OF .SK1000.    COMPLETED
DATE- 88/06/09  TIME- 07.34
ERRORS ENCOUNTERED- 0003  UPSI- X.00.
```

Figure 8-2. Link-Edit Example 1 (Part 3 of 3)

```
UNISYS SYSTEM OS/3  LINKAGE EDITOR                                              VER770503
DATE-  88/06/09  TIME- 07.30

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

          // PARAM RLIB#OBJMO
          // PARAM OUT#%N@
          /$
                    LOADM      SK1                                                  JCL00130
                    LINKOP NOA T                                                    JCL00140
                    INCLUDE    LK1ROOT,OBJMOS                                       JCL00150
                    OVERLAY A1                                                      JCL00160
                    INCLUDE   A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS           JCL00170
                    INCLUDE    A02%A02050C@,OBJMOS                                  JCL00180
                    INCLUDE    A03,OBJMOS                                           JCL00190
                    INCLUDE    A01%A01@,OBJMOS                                      JCL00200
                    INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS      JCL00210
                    INCLUDE    A02%A02090C,A02080C,A02070C,A02060C@,OBJMOS           JCL00220
                    OVERLAY A1                                                      JCL00230
                    INCLUDE   A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS           JCL00240
                    INCLUDE    A02%A02050C@,OBJMOS                                  JCL00250
                    INCLUDE    A03,OBJMOS                                           JCL00260
                    INCLUDE    A01%A01@,OBJMOS                                      JCL00270
                    INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS      JCL00280
                    INCLUDE    A02%A02090C,A02080C,A02070C@,OBJMOS                  JCL00290
                    ENTER A020 0C                                                   JCL00300
 -----A030-ENTER OPERAND NOT IN CURRENT PATH
          /*


*UNRESOLVED REFERENCES*

KL$OCP


                                        *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.  PHASE.  ADDRESS.      SYMBOL.   TYPE.  PHASE.  ADDRESS.      SYMBOL.   TYPE.  PHASE.  ADDRESS.

A01       CSECT  01    M 00001230      A01       CSECT  02    M 00001230      A01010C   CSECT  01*V*M 00000410
A01010C   CSECT  02    M 00000410      A01010E   ENTRY  01    M 00000494      A01010E   ENTRY  02    M 00000494
A01020C   CSECT  01*V*M 00000498      A01020C   CSECT  02    M 00000498      A01020E   ENTRY  01    M 00000520
A01020E   ENTRY  02    M 00000520      A01030C   CSECT  01*V*M 00000528      A01030C   CSECT  02    M 00000528
A01030E   ENTRY  01    M 00000584      A01030E   ENTRY  02    M 00000584      A01040C   CSECT  01*V*M 00000588
A01040C   CSECT  02    M 00000588      A01040E   ENTRY  01    M 00000648      A01040E   ENTRY  02    M 00000648
A02       CSECT  01    M 00001230      A02       CSECT  02    M 00001420      A02010C   CSECT  01*V*M 00001240
A02010C   CSECT  02    M 00001230      A02010E   ENTRY  01    M 000012D8      A02010E   ENTRY  02    M 00001200
A02020C   CSECT  01*V*M 000012E0      A02020C   CSECT  02    M 000012D8      A02020E   ENTRY  01    M 0000137C
A02020E   ENTRY  02    M 00001374      A02030C   CSECT  01*V*M 00001380      A02030C   CSECT  02    M 00001378
A02030E   ENTRY  01    M 00001420      A02030E   ENTRY  02    M 00001418      A02040C   CSECT  01*V*M 00001428
A02040C   CSECT  02    M 0C001424      A02040E   ENTRY  01    M 000014CC      A02040E   ENTRY  02    M 000014CC
A02050C   CSECT  01*V*M 00000650      A02050C   CSECT  02    M 00000650      A02050E   ENTRY  01    M 000006F8
A02050E   ENTRY  02    M 000006F8      A02060C   CSECT  01*V*  00001400      A02060E   ENTRY  01      00001570
A02070C   CSECT  01*V*M 00001580      A02070C   CSECT  02    M 00001400      A02070E   ENTRY  01    M 00001630
```

Figure 8-3. Link-Edit Example 2 (Part 1 of 6)

```
A02070E  ENTRY  02    M  00001580      A02080C  CSECT  01*V*M  00001638      A02080C  CSECT  02    M  00001588
A02080E  ENTRY  01    M  00001EEC      A02080E  ENTRY  02    M  0000163C      A02090C  CSECT  01*V*M  000016FD
A02090C  CSECT  02    M  0000164C      A02090E  ENTRY  01    M  000017A8      A02090E  ENTRY  02    M  000016F8
A03      CSECT  01    M  00000700      A03      CSECT  02    M  00000700      A03010C  CSECT  01*V*M  00000708
A03010C  CSECT  02    M  00000706      A03010E  ENTRY  01    M  00000784      A03010E  ENTRY  02    M  00000784
A03020C  CSECT  01*V*M  00000786      A03020C  CSECT  02    M  00000788      A03020E  ENTRY  01    M  00000868
A03020E  ENTRY  02    M  00000868      A03030C  CSECT  01*V*M  00000870      A03030C  CSECT  02    M  00000870
A03030E  ENTRY  01    M  00000924      A03030E  ENTRY  02    M  00000924      A03040C  CSECT  01*V*M  00000928
A03040C  CSECT  02    M  00000926      A03040E  ENTRY  01    M  000009E0      A03040E  ENTRY  02    M  000009E0
A03050C  CSECT  01*V*M  000009E       A03050C  CSECT  02    M  000009E8      A03050E  ENTRY  01    M  00000AA4
A03050E  ENTRY  02    M  00000AA4      A03060C  CSECT  01*V*M  00000AA8      A03060C  CSECT  02    M  00000AA8
A03060E  ENTRY  01    M  00000B6E      A03060E  ENTRY  02    M  00000968      A03070C  CSECT  01*V*M  00000970
A03070C  CSECT  02    M  00000B7C      A03070E  ENTRY  01    M  00000C34      A03070E  ENTRY  02    M  00000C34
A03080C  CSECT  01*V*M  00000C3E      A03080C  CSECT  02    M  00000C33      A03080E  ENTRY  01    M  00000D00
A03080E  ENTRY  02    M  00000D0C      A03090C  CSECT  01*V*M  00000D08      A03090C  CSECT  02    M  00000D08
A03090E  ENTRY  01    M  00000DD4      A03090E  ENTRY  02    M  00000DD4      A03100C  CSECT  01*V*M  00000D08
A03100C  CSECT  02    M  00000DD       A03100E  ENTRY  01    M  00000EA8      A03100E  ENTRY  02    M  00000EA8
A03110C  CSECT  01*V*M  00000EBC      A03110C  CSECT  02    M  00000E80      A03110E  ENTRY  01    M  00000F84
A03110E  ENTRY  02    M  00000F84      A03120C  CSECT  01*V*M  00000F88      A03120C  CSECT  02    M  00000F88
A03120E  ENTRY  01    M  00001064      A03120E  ENTRY  02    M  00001060      A03130C  CSECT  01*V*M  00001068
A03130C  CSECT  02    M  00001064      A03130E  ENTRY  01    M  00001144      A03130E  ENTRY  02    M  00001144
A03140C  CSECT  01*V*M  00001148      A03140C  CSECT  02    M  00001148      A03140E  ENTRY  01    M  00001228
A03140E  ENTRY  02    M  00001228      KE$ALP   ENTRY  ABS     000017AC      KE$RES   ENTRY  ABS     000017AC
KL$NTB   ENTRY  ROOT    00000000      KL$OCP   EXTRN  --      --------      KL$PTB   ENTRY  ROOT    00000084
LK1ROOT  CSECT  ROOT    00000084
```

Figure 8-3. Link-Edit Example 2 (Part 2 of 6)

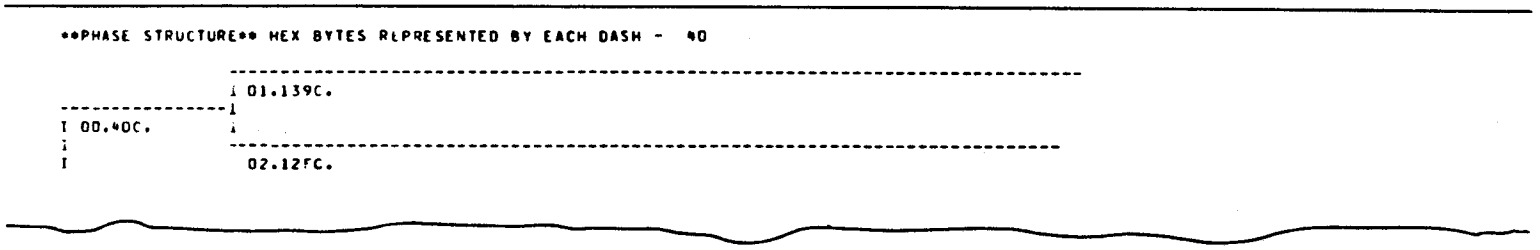


Figure 8-3. Link-Edit Example 2 (Part 3 of 6)

```
                                    ** ALLOCATION MAP **

                    LOAD MODULE -   SK1000          SIZE -   000017AC

PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
*K100000        NODE - ROOT                                          00000000    0000040R    0000040C
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
         -  88/07/02  11.49  -          LK1P00T    OBJ
                                       LK1R00T    CSECT     01        00000088    0000040R    00000354    00000000

            00000088
*K100001        NODE -   A1                                          00000410    000017AR    0000139C
         -  88/06/27  12.23  -          A01        OBJ
                                       A01010C    CSECT     02        00000410    00000497    000000R8    0000000R
                                       A01020C    CSECT     1E        00000498    00000523    0000008C    00000090
                                       A01030C    CSECT     1F        00000528    00000587    00000090    00000120
                                       A01040C    CSECT     20        00000588    0000064R    00000094    00000180
                                       A01010E      ENTRY   02        00000494                            0000008C
                                       A01020E      ENTRY   1E        00000520                            0000011R
                                       A01030E      ENTRY   1F        00000584                            000001AC
                                       A01040E      ENTRY   20        00000648                            00000240
         -  88/06/27  12.25  -          A02        OBJ
                                       A02050C    CSECT     21        00000650    000006FR    000000AC    00000298
                                       A02050E      ENTRY   21        000006F8                            00000340
         -  88/06/27  12.28  -          A03        OBJ
                                       A03        CSECT     01        00000700    00000701    00000002    00000000
                                       A03010C    CSECT     02        00000708    00000787    000000R0    00000008
                                       A03020C    CSECT     1E        00000788    00000868    000000R4    00000088
                                       A03030C    CSECT     1F        00000870    00000927    000000BR    00000170
                                       A03040C    CSECT     20        00000928    000009E3    000000BC    00000228
                                       A03050C    CSECT     21        000009E8    00000AA7    000000C0    000002E8
                                       A03060C    CSECT     22        00000AA8    0000086R    000000C4    000003AR
                                       A03070C    CSECT     23        00000R70    00000C37    000000C8    00000470
                                       A03080C    CSECT     24        00000C38    00000D03    000000CC    00000538
                                       A03090C    CSECT     25        00000D08    00000DD7    000000D0    0000060R
                                       A03100C    CSECT     26        00000DD8    00000EAR    00000D34    000006D8
                                       A03110C    CSECT     27        00000EB0    00000F87    000000DR    00000780
                                       A03120C    CSECT     28        00000F88    00001063    000000EC    00000R88
                                       A03130C    CSECT     29        00001068    00001147    000000E0    00000968
                                       A03140C    CSECT     2A        00001148    0000122R    000000E4    00000A4R
                                       A03010E      ENTRY   02        00000784                            000000R4
                                       A03020E      ENTRY   1E        00000868                            0000016R
                                       A03030E      ENTRY   1F        00000924                            00000224
                                       A03040E      ENTRY   20        00000SE0                            000002E0
                                       A03050E      ENTRY   21        00000AA4                            000003A4
                                       A03060E      ENTRY   22        00000868                            0000046R
                                       A03070E      ENTRY   23        00000C34                            00000534
                                       A03080F      ENTRY   24        00000000                            00000600
                                       A03090E      ENTRY   25        00000DD4                            0000060R
                                       A03100E      ENTRY   26        00000FA8                            000007AR
                                       A03110E      ENTRY   27        00000F84                            00000R84
                                       A03120E      ENTRY   28        00001060                            00000960
                                       A03130E      ENTRY   29        00001144                            00000A44
                                       A03140E      ENTRY   2A        00001228                            00000928
         -  88/06/27  12.23  -          A01        OBJ
                                       A01        CSECT     01        00001230    00001231    00000002    00000000
```

Figure 8-3. Link-Edit Example 2 (Part 4 of 6)

PHASE NAME	TRANS ADDR	FLAG	LABEL	TYPE	ESID	LNK ORG	HIADDR	LENGTH	OBJ ORG
-	88/06/27 12.25	-	A02	OBJ					
			A02	CSECT	01	00001238	00001239	00000002	00000000
			A02010C	CSECT	02	00001240	000012DB	0000009C	00000008
			A02020C	CSECT	1E	000012E0	0000137F	000000A0	000000A8
			A02030C	CSECT	1F	00001380	00001423	000000A4	00000148
			A02040C	CSECT	20	00001428	000014CF	000000A8	000001F0
			A02010E	ENTRY	02	000012D8			000000A0
			A02020E	ENTRY	1E	0000137C			00000144
			A02030E	ENTRY	1F	00001420			000001E8
			A02040E	ENTRY	20	000014CC			00000294
-	88/06/27 12.25	-	A02	OBJ					
			A02060C	CSECT	22	00001480	0000157F	00000080	00000348
			A02070C	CSECT	23	00001580	00001633	000000B4	000003F8
			A02080C	CSECT	24	00001638	000016EF	000000B8	000004B0
			A02090C	CSECT	25	000016F0	000017AF	000000BC	00000568
			A02060E	ENTRY	22	0000157C			000003F4
			A02070E	ENTRY	23	00001630			000004A8
			A02080E	ENTRY	24	000016EC			00000564
			A02090E	ENTRY	25	000017A8			00000620
	00000410								
*100002	NODE -	#1				00000410	000016FB	000012[illegible]	
-	88/06/27 12.23	-	A01	OBJ					
			A01010C	CSECT	02	00000410	00000497	00000088	00000008
			A01020C	CSECT	1E	00000498	00000523	0000008C	00000090
			A01030C	CSECT	1F	00000528	00000587	00000090	00000120
			A01040C	CSECT	20	00000588	00000649	00000094	000001B0
			A01010E	ENTRY	02	00000494			0000008C
			A01020E	ENTRY	1E	00000520			00000118
			A01030E	ENTRY	1F	00000564			000001AC
			A01040E	ENTRY	20	00000648			00000240
-	88/06/27 12.25	-	A02	OBJ					
			A02050C	CSECT	21	000006E0	000C06FB	000000AC	00000298
			A02050E	ENTRY	21	000006F8			00000340
-	88/06/27 12.28	-	A03	OBJ					
			A03	CSECT	01	00000700	00000701	00000C02	00000000
			A03010C	CSECT	02	00000708	000007B7	00000080	00000008
			A03020C	CSECT	1E	000007B8	00000868	00000084	00000088
			A03030C	CSECT	1F	00000870	00000927	00000088	00000170
			A03040C	CSECT	20	00000928	000009E3	0000008C	00000228
			A03050C	CSECT	21	000009E8	00000AA7	000000C0	000002E8
			A03060C	CSECT	22	00000AA8	00000B68	000000C4	000003A8
			A03070C	CSECT	23	00000B70	00000C37	000000C8	00000470
			A03080C	CSECT	24	00000C38	00000D03	000000CC	00000538
			A03090C	CSECT	25	00000D08	00000DD7	00000000	00000608
			A03100C	CSECT	26	00000DD8	000C0EAB	00000004	000006D8
			A03110C	CSECT	27	00000EB0	00000F87	00000008	00000780
			A03120C	CSECT	28	00000F88	00001063	0000000C	00000888
			A03130C	CSECT	[illegible]	00001069	00001[illegible]47	000000C0	00000968
			A03140C	CSECT	2A	00001148	00001228	000000E4	00000A48
			A03010E	ENTRY	02	000007B4			000000B4
			A03020E	ENTRY	1E	00000868			00000168
			A03030E	ENTRY	1F	00000924			00000224
			A03040E	ENTRY	20	000009E0			000002E0
			A03050E	ENTRY	21	00000AA4			000003A4
			A03060E	ENTRY	22	00000B68			00000468

Figure 8-3. Link-Edit Example 2 (Part 5 of 6)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG      HIADDR      LENGTH      OBJ ORG
                                      A03070E     ENTRY      23      00000C34                             00000534
                                      A03080E     ENTRY      24      00000000                             00000600
                                      A03090E     ENTRY      25      00000004                             00000604
                                      A03100E     ENTRY      26      00000FA8                             000007A8
                                      A03110E     ENTRY      27      00000F84                             00000984
                                      A03120E     ENTRY      28      00001060                             00000960
                                      A03130E     ENTRY      29      00001144                             00000A44
                                      A03140E     ENTRY      2A      00001228                             00000B28
          -  88/06/27  12.23  -       A01          OBJ
                                      A01         CSECT       01      00001230     00001231    00000002    00000000
          -  88/06/27  12.25  -       A02          OBJ
                                      A02         CSECT       01      00001420     00001421    00000002    00000000
                                      A02010C     CSECT       02      00001238     00001203    0000009C    00000008
                                      A02020C     CSECT       1E      00001208     00001377    000000A0    000000A8
                                      A02030C     CSECT       1F      00001378     0000141B    000000A4    00000148
                                      A02040C     CSECT       20      00001428     000014CF    000000A8    000001F0
                                      A02010E       ENTRY     02      00001200                             000000A0
                                      A02020E       ENTRY     1E      00001374                             00000144
                                      A02030E       ENTRY     1F      00001418                             000001E8
                                      A02040E       ENTRY     20      000014CC                             00000294
          -  88/06/27  12.25  -       A02          OBJ
                                      A02070C     CSECT       23      000014D0     00001583    00000084    000003F8
                                      A02080C     CSECT       24      00001588     0000163F    000000B8    00000480
                                      A02090C     CSECT       25      00001640     000016F8    0000008C    0000056C
                                      A02070E       ENTRY     23      00001580                             000004A8
                                      A02080E       ENTRY     24      0000163C                             00000564
                                      A02090E       ENTRY     25      000016F8                             00000620
                00001400

                                                    FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED      E - EXCLUSIVE .A. REF    G - GENERATED EXTRN      I - INCLUSIVE .V. REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED  N - NOT INCLUDED         P - PROMOTED COMMON      R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF     V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*

LINK EDIT OF .SK1000.   COMPLETED
DATE- 88/06/09  TIME- 07.32
ERRORS ENCOUNTERED- 0002  UPSI- X.00.
```

Figure 8-3. Link-Edit Example 2 (Part 6 of 6)

```
UNISYS SYSTEM OS/3  LINKAGE EDITOR                                                    VER770503
DATE-  88/06/09  TIME- 07.28

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

           // PARAM RLIB#OBJMOS
           // PARAM OUT#%N0
           /$
                     LOADM     CTLNK000                                          JCL00140
 -----K016-LOADM NAME INVALID
                     INCLUDE    LK1ROOT,OBJMOS                                   JCL00150
                     OVERLAY   CTLNK0                                            JCL00160
 -----K000-ROOT PHASE OVERLAID
                     INCLUDE   A01%A01010C,A01020C,A01030C,A01040C0,OBJMOS       JCL00170
                     INCLUDE    A02%A02050C0,OBJMOS                              JCL00180
                     INCLUDE    A03,OBJMOS                                       JCL00190
                     INCLUDE    A01%A010,OBJMOS                                  JCL00200
                     INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A020,OBJMOS  JCL00210
           /*
A02060C *AUTO-INCLUDED*


*UNRESOLVED REFERENCES*

KL$OCP


                                        *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.   PHASE.  ADDRESS.      SYMBOL.   TYPE.   PHASE.  ADDRESS.      SYMBOL.   TYPE.   PHASE.  ADDRESS.

A01       CSECT   01      00000020      A01010C   CSECT   01*V*   00000000      A01010E   ENTRY   01      00000084
A01020C   CSECT   01*V*   00000086      A01020E   ENTRY   01      00000110      A01030C   CSECT   01*V*   00000118
A01030E   ENTRY   01      000001A4      A01040C   CSECT   01*V*   000001A8      A01040E   ENTRY   01      00000238
A02       CSECT   01      00000E20      A02010C   CSECT   01*V*   00000E30      A02010E   ENTRY   01      00000FC8
A02020C   CSECT   01*V*   00000ED0      A02020E   ENTRY   01      00000F6C      A02030C   CSECT   01*V*   00000F70
A02030E   ENTRY   01      00001010      A02040C   CSECT   01*V*   00001018      A02040E   ENTRY   01      000010BC
A02050C   CSECT   01*V*   00000240      A02050E   ENTRY   01      000002E8      A02060C   CSECT   ROOT    000000A8
A02060E   ENTRY   ROOT    00000154      A02070C   CSECT   ROOT    00000158      A02070E   ENTRY   ROOT    00000208
A02080C   CSECT   ROOT    00000210      A02080E   ENTRY   ROOT    000002C4      A02090C   CSECT   ROOT    000002C8
A02090E   ENTRY   ROOT    00000380      A03       CSECT   01      000002F0      A03010C   CSECT   01*V*   000002F8
A03010E   ENTRY   01      000003A4      A03020C   CSECT   01*V*   000003A8      A03020E   ENTRY   01      00000458
A03030C   CSECT   01*V*   00000460      A03030E   ENTRY   01      00000514      A03040C   CSECT   01*V*   00000518
A03040E   ENTRY   01      000005D0      A03050C   CSECT   01*V*   000005D8      A03050E   ENTRY   01      00000694
A03060C   CSECT   01*V*   00000698      A03060E   ENTRY   01      00000758      A03070C   CSECT   01*V*   00000760
A03070E   ENTRY   01      00000824      A03080C   CSECT   01*V*   00000828      A03080E   ENTRY   01      00000FF0
A03090C   CSECT   01*V*   000008F8      A03090E   ENTRY   01      000009C4      A03100C   CSECT   01*V*   000009C8
A03100E   ENTRY   01      00000A98      A03110C   CSECT   01*V*   00000AA0      A03110E   ENTRY   01      00000F74
A03120C   CSECT   01*V*   00000B7C      A03120E   ENTRY   01      00000C50      A03130C   CSECT   01*V*   00000C58
A03130E   ENTRY   01      00000D34      A0314OC   CSECT   01*V*   00000D38      A0314OE   ENTRY   01      00000F18
KE$ALP    ENTRY   ABS     000010CC      KE$RES    ENTRY   ABS     000010C0      KL$NTB    ENTRY   ROOT    00000000
KL$OCP    EXTRN   --      --------      KL$PTB    ENTRY   ROOT    000000A4      LK1ROOT   CSECT   ROOT    00000388
```

Figure 8-4. Link-Edit Example 3 (Part 1 of 4)

```
**PHASE STRUCTURE** HEX BYTES REPRESENTED BY EACH DASH -  3D

-----------------------------------
1 00.C.C.
-----------------------------------------------------------------------------
: 01.10C0.
```

Figure 8-4. Link-Edit Example 3 (Part 2 of 4)

```
                                        ** ALLOCATION MAP **

                    LOAD MODULE -   CTLNK0          SIZE -   000010C0

PHASE NAME   TRANS ADDR    FLAG      LABEL     TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
CTLNK000       NODE - ROOT                                        00000000    000006D9    000006DC
*** START OF AUTO-INCLUDED ELEMENTS -
          -  88/06/27  12.25  -      A02       OBJ
                                    A02060C   CSECT     22        000000A8    00000157    000000B0    00000348
                                    A02070C   CSECT     23        00000158    000002D8    000000B4    000003F8
                                    A02080C   CSECT     24        00000210    000002C7    000000B8    00000480
                                    A02090C   CSECT     25        000002C8    00000383    000000BC    00000568
                                    A02060E     ENTRY   22        00000154                            000003F4
                                    A02070E     ENTRY   23        00000208                            000004A8
                                    A0208CE     ENTRY   24        000002C4                            00000564
                                    A02090E     ENTRY   25        00000360                            00000620
*** END OF AUTO-INCLUDED ELEMENTS -
          -  88/07/02  11.49  -      LK1ROOT   OBJ
                                    LK1ROOT   CSECT     01        00000388    000006D8    00000354    00000000

                 000003F8
CTLNK001       NODE -   CTLNK0                                     00000000    000010BF    000010C0
          -  88/06/27  12.23  -      A01       OBJ
                                    A01010C   CSECT     02        00000000    00000087    00000088    00000000
                                    A01020C   CSECT     1E        00000088    00000113    0000008C    00000090
                                    A01030C   CSECT     1F        00000118    000001A7    00000090    00000120
                                    A01040C   CSECT     20        000001A8    0000023B    00000094    00000190
                                    A01010E     ENTRY   02        00000084                            0000008C
                                    A0102CE     ENTRY   1E        00000110                            00000118
                                    A01030E     ENTRY   1F        000001A4                            000001AC
                                    A01040E     ENTRY   20        00000238                            00000240
          -  88/06/27  12.25  -      A02       OBJ
                                    A02050C   CSECT     21        00000240    000002E8    000000A0    00000298
                                    A02050E     ENTRY   21        000002E8                            00000340
          -  88/06/27  12.28  -      A03       OBJ
                                    A03       CSECT     01        000002F0    000002F1    00000002    00000000
                                    A03010C   CSECT     02        000002F8    000003A7    00000080    00000008
                                    A03020C   CSECT     1E        000003A8    00000458    00000084    00000088
                                    A03030C   CSECT     1F        00000460    00000517    00000088    00000170
                                    A03040C   CSECT     20        00000518    000005D3    0000008C    000002A8
                                    A03050C   CSECT     21        000005D8    00000697    00000000    000002E8
                                    A03060C   CSECT     22        00000698    0000075B    000000C4    000003A8
                                    A03070C   CSECT     23        00000760    00000927    000000C8    00000470
                                    A03080C   CSECT     24        00000828    000008F3    000000CC    00000538
                                    A03090C   CSECT     25        00000BF8    000009C7    00000000    000006D8
                                    A0310CC   CSECT     26        000009C8    00000A9B    00000004    00000608
                                    A0311CC   CSECT     27        00000AA0    00000B77    00000008    00000780
                                    A0312CC   CSECT     28        00000B78    00000C53    0000000C    00000888
                                    A0313CC   CSECT     29        00000C58    00000D37    00000010    00000968
                                    A0314CC   CSECT     2A        00000D38    00000E1B    00000014    00000A48
                                    A03010E     ENTRY   02        000003A4                            00000084
                                    A03020E     ENTRY   1E        00000458                            00000168
                                    A03030E     ENTRY   1F        00000514                            00000224
                                    A03040E     ENTRY   20        000005D0                            000002E0
                                    A03050E     ENTRY   21        00000694                            000003A4
                                    A03060E     ENTRY   22        00000758                            0000046B
                                    A03070E     ENTRY   23        00000824                            00000534
```

Figure 8-4. Link-Edit Example 3 (Part 3 of 4)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HI ADDR     LENGTH     OBJ ORG
                                      A03080E     ENTRY     24       000008F0                          00000600
                                      A03090E     ENTRY     25       000009C4                          000006D4
                                      A03100E     ENTRY     26       00000A98                          000007A8
                                      A03110E     ENTRY     27       00000B74                          00000884
                                      A03120E     ENTRY     28       00000C50                          00000960
                                      A03130E     ENTRY     29       00000D34                          00000A44
                                      A03140E     ENTRY     2A       00000E18                          00000B28
          - 88/06/27  12.23  -        A01         OBJ
                                      A01         CSECT     01       00000F20     00000E21     00000002     00000000
          - 88/06/27  12.25  -        A02         OBJ
                                      A02         CSECT     01       00000F28     00000E29     00000002     00000700
                                      A02010C     CSECT     02       00000E30     00000ECB     0000009C     00000008
                                      A02020C     CSECT     1E       00000F00     00000F6F     000000A0     000000A8
                                      A02030C     CSECT     1F       00000F70     00001013     000000A4     00000148
                                      A02040C     CSECT     20       00001018     0000108F     000000AB     000001F0
                                      A02010E     ENTRY     02       00000FC8                          000000A0
                                      A02020E     ENTRY     1E       00000F6C                          00000144
                                      A02030E     ENTRY     1F       00001010                          000001E8
                                      A02040E     ENTRY     20       0000108C                          00000294
                 CC000000

                                                    FLAG CODES -
E - BLK DATA CSECT      D - AUTO-DELETED      F - EXCLUSIVE .A. REF    G - GENERATED EXTRN      I - INCLUSIVE .V. REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED  N - NOT INCLUDED         P - PROMOTED COMMON      R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF     V - VCON ITEM
ANY OTHER CODES REPRESENT PROCESS ERRORS.

LINK EDIT OF .CTLNK0.   COMPLETED
DATE- 88/06/09  TIME- 07.29
ERRORS ENCOUNTERED- 0003  UPSI- X.00.
```

Figure 8-4. Link-Edit Example 3 (Part 4 of 4)

```
UNISYS SYSTEM OS/3  LINKAGE EDITOR                                              VER770624
DATE-  88/07/08  TIME- 07.45

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB=OBJMOS
            // PARAM OUT=%NB
            /%
                    LOADM       SK1
                    LINKOP NOMAP
                    INCLUDE     LK1ROOT,OBJMOS
                    OVERLAY     NODE01
                    INCLUDE     AOS%A0501CB,OBJMOS
                    OVERLAY     NODE02
                    INCLUDE     AOS%A0502CB,OBJMOS
                    OVERLAY     NODE03
                    INCLUDE     AOS%A0503CB,OBJMOS
                    OVERLAY     NODE04
                    INCLUDE     AOS%A0504CB,OBJMOS
                    OVERLAY     NODE05
                    INCLUDE     AOS%A0505CB,OBJMOS
                    OVERLAY     NODE06
                    INCLUDE     AOS%A0506CB,OBJMOS
                    OVERLAY     NODE07
                    INCLUDE     AOS%A0507CB,OBJMOS
                    OVERLAY     NODE08
                    INCLUDE     AOS%A0508CB,OBJMOS
                    OVERLAY     NODE09
                    INCLUDE     AOS%A0509CB,OBJMOS
                    OVERLAY     NODE10
                    INCLUDE     AOS%A0510CB,OBJMOS
                    OVERLAY     NODE11
                    INCLUDE     AOS%A0511CB,OBJMOS
                    OVERLAY     NODE12
                    INCLUDE     AOS%A0512CB,OBJMOS
                    OVERLAY     NODE13
                    INCLUDE     AOS%A0513CB,OBJMOS
                    OVERLAY     NODE14
 -----K002-NODE POINT LIMIT - 14 PER PATH
                    INCLUDE     AOS%A0514CB,OBJMOS
                    OVERLAY     NODE15
            /*


*UNRESOLVED REFERENCES*

A01010E     A01020E     A01030E     A01040E     A02010E     A02020E     A02030E     A02040E     A02050E     A02060E
A02070E     A02080E     A02090E     A03010E     A03020E     A03030E     A03040E     A03050E     A03060E     A03070E
A03080E     A03090E     A03100E     A03110E     A03120E     A03130E     A03140E     A01010C     A01020C     A01030C
A01040C     A02010C     A02020C     A02030C     A02040C     A02050C     A02060C     A02070C     A02080C     A02090C
A03010C     A03020C     A03030C     A03040C     A03050C     A03060C     A03070C     A03080C     A03090C     A03100C
A03110C     A03120C     A03130C     A03140C

                                *DEFINITIONS DICTIONARY*
```

Figure 8-5. Link-Edit Example 4 (Part 1 of 5)

SYMBOL.	TYPE.	PHASE.	ADDRESS.	SYMBOL.	TYPE.	PHASE.	ADDRESS.	SYMBOL.	TYPE.	PHASE.	ADDRESS.
A01010C	EXTRN	--*V*	--------	A01010E	EXTRN	--	--------	A01020C	EXTRN	--*V*	--------
A01020E	EXTRN	--	--------	A01030C	EXTRN	--*V*	--------	A01030E	EXTRN	--	--------
A0104CC	EXTRN	--*V*	--------	A0104CE	EXTRN	--	--------	A02010C	EXTRN	--*V*	--------
A02010E	EXTRN	--	--------	A02020C	EXTRN	--*V*	--------	A02020E	EXTRN	--	--------
A0203CC	EXTRN	--*V*	--------	A02030E	EXTRN	--	--------	A02040C	EXTRN	--*V*	--------
A02040E	EXTRN	--	--------	A02050C	EXTRN	--*V*	--------	A02050E	EXTRN	--	--------
A0206CC	EXTRN	--*V*	--------	AC2060E	EXTRN	--	--------	A02070C	EXTRN	--*V*	--------
A02070E	EXTRN	--	--------	A02080C	EXTRN	--*V*	--------	A02080E	EXTRN	--	--------
A0209CC	EXTRN	--*V*	--------	A02090E	EXTRN	--	--------	A03010C	EXTRN	--*V*	--------
AC3010E	EXTRN	--	--------	A03C20C	EXTRN	--*V*	--------	A03020E	EXTRN	--	--------
AC3030C	EXTRN	--*V*	--------	A03030E	EXTRN	--	--------	A03040C	EXTRN	--*V*	--------
A03040E	EXTRN	--	--------	A03050C	EXTRN	--*V*	--------	A03050E	EXTRN	--	--------
A03060C	EXTRN	--*V*	--------	A03060E	EXTRN	--	--------	A03070C	EXTRN	--*V*	--------
A03070E	EXTRN	--	--------	A03080C	EXTRN	--*V*	--------	A03080E	EXTRN	--	--------
A0309CC	EXTRN	--*V*	--------	A03090E	EXTRN	--	--------	A03100C	EXTRN	--*V*	--------
A03100E	EXTRN	--	--------	A03110C	EXTRN	--*V*	--------	A03110E	EXTRN	--	--------
A0312CC	EXTRN	--*V*	--------	A03120E	EXTRN	--	--------	A03130C	EXTRN	--*V*	--------
A03130E	EXTRN	--	--------	A0314CC	EXTRN	--*V*	--------	A03140E	EXTRN	--	--------
A0501C	CSECT	01	00000348	A0501E	ENTRY	01	0000035E	A0502C	CSECT	02	00000368
A0502E	ENTRY	02	00000372	A0503C	CSECT	03	00000378	A0503E	ENTRY	03	00000386
A0504C	CSECT	04	000003*0	A0504E	ENTRY	04	000003A2	A0505C	CSECT	05	000003A8
A0505E	ENTRY	05	000003CE	A0506C	CSECT	06	000003C8	A0506E	ENTRY	06	000003E2
A0507C	CSECT	07	000003F8	A0507E	ENTRY	07	00000406	A0508C	CSECT	08	00000410
A0508E	ENTRY	08	00000422	A0509C	CSECT	09	00000438	A0509E	ENTRY	09	0000045E
A0510C	CSECT	10	00000468	A0510E	ENTRY	10	00000492	A0511C	CSECT	11	00000498
A0511E	ENTRY	11	000004*6	A0512C	CSECT	12	000004D0	A0512E	ENTRY	12	00000502
A0513C	CSECT	13	000005C8	A0513E	ENTRY	13	0000053E	A0514C	CSECT	14	00000358
A0514E	ENTRY	14	000003*2	KESALP	ENTRY	ABS	00000542	KESPES	ENTRY	ABS	00000542
LK1ROOT	CSECT	ROOT	00000000								

Figure 8-5. Link-Edit Example 4 (Part 2 of 5)

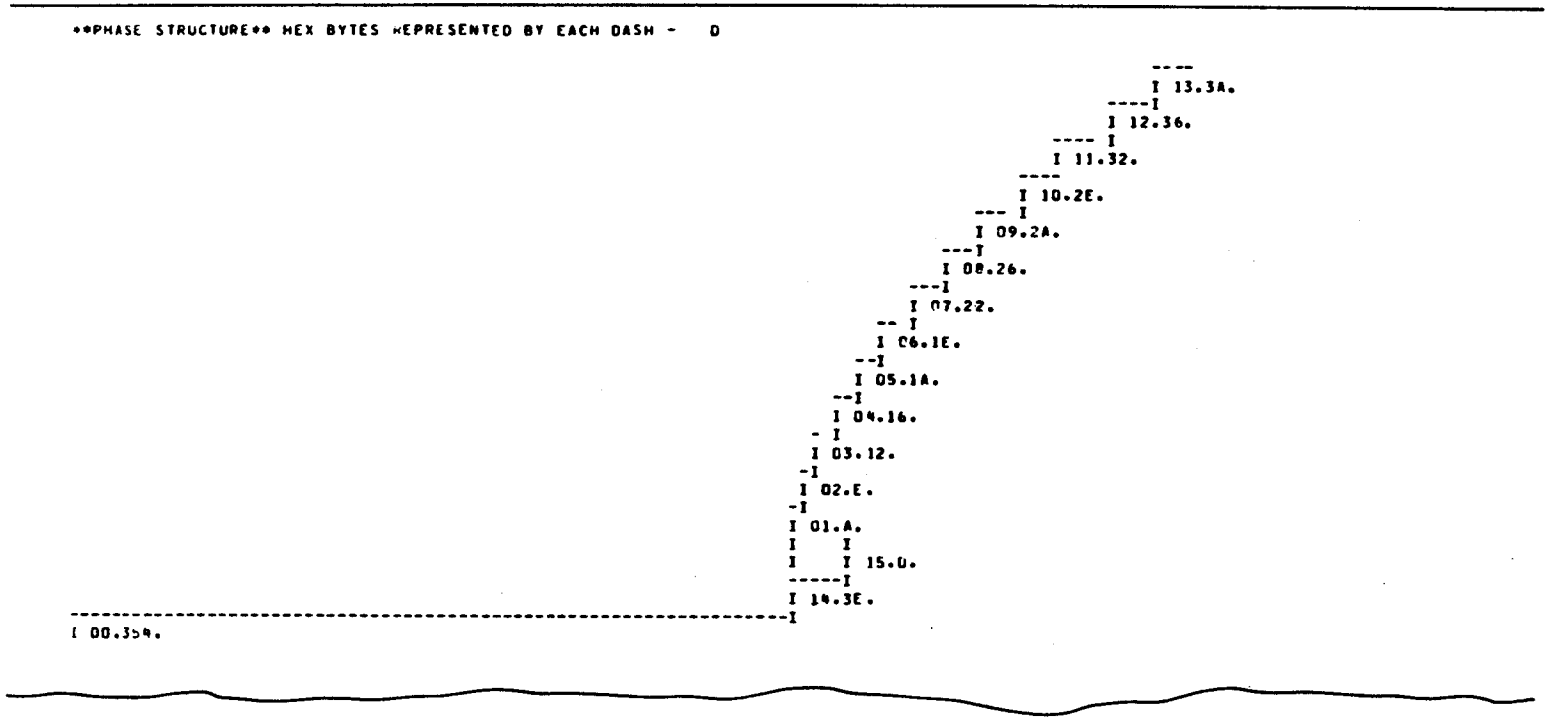


Figure 8-5. Link-Edit Example 4 (Part 3 of 5)

```
                                    ** ALLOCATION MAP **

                    LOAD MODULE -   SK1000          SIZE -   00000542

PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR     LENGTH     OBJ ORG
SK100000        NODE - ROOT                                          00000000   00000353   00000354
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
         -  88/07/02  11.49  -        LK1ROOT    OBJ
                                      LK1ROOT    CSECT     01        00000000   00000353   00000354   00000000

                00000000
SK100001        NODE -   NODE01                                      00000358   00000361   0000000A
         -  88/06/27  12.42  -        A05        OBJ
                                      A0501C     CSECT     02        00000358   00000361   0000000A   00000008
                                      A0501E     ENTRY     02        0000035E                         0000000E

                00000358
SK100002        NODE -   NODE02                                      00000368   00000375   0000000E
         -  88/06/27  12.42  -        A05        OBJ
                                      A0502C     CSECT     03        00000368   00000375   0000000E   00000018
                                      A0502E     ENTRY     03        00000372                         00000022

                00000368
SK100003        NODE -   NODE03                                      00000378   00000389   00000012
         -  88/06/27  12.42  -        A05        OBJ
                                      A0503C     CSECT     04        00000378   00000389   00000012   00000028
                                      A0503E     ENTRY     04        00000386                         00000036

                00000378
SK100004        NODE -   NODE04                                      00000390   000003A5   00000016
         -  88/06/27  12.42  -        A05        OBJ
                                      A0504C     CSECT     05        00000390   000003A5   00000016   00000040
                                      A0504E     ENTRY     05        000003A2                         00000052

                00000390
SK100005        NODE -   NODE05                                      000003A8   000003C1   0000001A
         -  88/06/27  12.42  -        A05        OBJ
                                      A0505C     CSECT     06        000003A8   000003C1   0000001A   00000058
                                      A0505E     ENTRY     06        000003BE                         0000006E

                000003A8
SK100006        NODE -   NODE06                                      000003C8   000003E5   0000001E
         -  88/06/27  12.42  -        A05        OBJ
                                      A0506C     CSECT     07        000003C8   000003E5   0000001E   00000078
                                      A0506E     ENTRY     07        000003E2                         00000092

                000003C8
SK100007        NODE -   NODE07                                      000003E8   00000409   00000022
         -  88/06/27  12.42  -        A05        OBJ
                                      A0507C     CSECT     08        000003E8   00000409   00000022   00000098
                                      A0507E     ENTRY     08        00000406                         000000B6

                000003E8
SK100008        NODE -   NODE08                                      00000410   00000435   00000026
         -  88/06/27  12.42  -        A05        OBJ
                                      A0508C     CSECT     09        00000410   00000435   00000026   000000C0
                                      A0508E     ENTRY     09        00000432                         000000E2

                00000410
SK100009        NODE -   NODE09                                      00000438   00000461   0000002A
         -  88/06/27  12.42  -        A05        OBJ
                                      A0509C     CSECT     0A        00000438   00000461   0000002A   000000E8
                                      A0509E     ENTRY     0A        0000045E                         0000010E

                00000438
```

Figure 8-5. Link-Edit Example 4 (Part 4 of 5)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
SK100010       NODE -    NODE10                                    00000468    00000495    0000002E
          -  88/06/27  12.42  -       A05          ORJ
                                      A0510C     CSECT      0B      00000468    00000495    0000002E    00000118
                                      A0510E       ENTRY    0B      00000492                            00000142
               00000468
SK100011       NODE -    NODE11                                    00000498    000004C9    00000032
          -  88/06/27  12.42  -       A05          OBJ
                                      A0511C     CSECT      0C      00000498    000004C9    00000032    00000148
                                      A0511E       ENTRY    0C      000004C6                            00000176
               00000498
SK100012       NODE -    NODE12                                    000004D0    00000505    00000036
          -  88/06/27  12.42  -       A05          ORJ
                                      A0512C     CSECT      0D      000004D0    00000505    00000036    00000180
                                      A0512E       ENTRY    0D      00000502                            00000182
               000004D0
SK100013       NODE -    NODE13                                    00000508    00000541    0000003A
          -  88/06/27  12.42  -       A05          OBJ
                                      A0513C     CSECT      0E      00000508    00000541    0000003A    00000188
                                      A0513E       ENTRY    0E      0000053E                            000001EE
               00000508
SK100014       NODE -    NODE14                                    00000358    00000395    0000003E
          -  88/06/27  12.42  -       A05          OBJ
                                      A0514C     CSECT      0F      00000358    00000395    0000003E    000001F8
                                      A0514E       ENTRY    0F      00000392                            00000232
               00000358
SK100015       NODE -    NODE15                                    00000398                00000000
-----K072-SK100015 ZERO LENGTH PHASE
               00000398


                                                    FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE .A. REF   G - GENERATED EXTRN     I - INCLUSIVE .V. REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*

LINK EDIT OF .SK1000.   COMPLETED
DATE- 88/07/08  TIME- 07.49
ERRORS ENCOUNTERED- 0056  UPSI- X.00.
```

Figure 8-5. Link-Edit Example 4 (Part 5 of 5)

```
UNISYS SYSTEM OS/3  LINKAGE EDITOR
DATE-  88/07/06  TIME-  13.06

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB$MDMOBJ
            /$
                    LOADM LNKEDT
                    LINKOP NOAUTO
            *       90/30 LINKER TEST EDIT $LINK OF LINKER$
                     INCLUDE PRSYSA,MDMOBJ
                     INCLUDE LNKEDTCA$LNKEDT1A,MDMOBJ
                     INCLUDE LNKEDTCA$LNKEDT$,MDMOBJ
                     ENTER LNKEDT
                     OVERLAY OVER1
                     INCLUDE LNKEDTCA$LKSP$,MDMOBJ
                     INCLUDE LNKEDTCA$LNKEDTCA,LNKEDTSA,LNKEDTPT,LNKEDT2$,MDMOBJ
                     INCLUDE LNKEDTIN$LNKEDTIN$,MDMOBJ
                     ENTER LNKEDTIN
                     OVERLAY OVER1
                     INCLUDE LNKEDTCA$LKSP$,MDMOBJ
                     OVERLAY OVER2
                     INCLUDE LNKEDTCA$LNKEDTCA$,MDMOBJ
                     OVERLAY OVER3
                     INCLUDE LNKEDTCA$LNKEDTSA$,MDMOBJ
                     OVERLAY OVER4
                     INCLUDE LNKEDTCA$LNKEDTPT,LNKEDT2$,MDMOBJ
                     OVERLAY    OVER5
                     INCLUDE LNKEDTIN$LNKEDTI1$,MDMOBJ
                     OVERLAY OVER5
                     INCLUDE LNKEDTIC$LNKEDTIC$,MDMOBJ
                     OVERLAY OVER6
                     INCLUDE LNKEDTIC$LKSCPH10$,MDMOBJ
                     OVERLAY OVER7
                     INCLUDE LNKEDTIC$LKSCPH04$,MDMOBJ
                     OVERLAY OVER8
                     INCLUDE LNKEDTIC$LKSCPH12$,MDMOBJ
                     OVERLAY OVER7
                     INCLUDE LNKEDTIC$LKSCPH30$,MDMOBJ
                     INCLUDE LNKEDTIC$LKSCPH09$,MDMOBJ
                     OVERLAY                    OVER7
                     INCLUDE LNKEDTIC$LKSCPH14$,MDMOBJ
                     OVERLAY OVER8
                     INCLUDE LNKEDTIC$LKSCPH05$,MDMOBJ
                     OVERLAY                    OVER9
                     INCLUDE LNKEDTIC$LKSCPH07$,MDMOBJ
                     OVERLAY                    OVER10
                     INCLUDE LNKEDTIC$LKSCPH11$,MDMOBJ
                     OVERLAY                    OVER10
                     INCLUDE LNKEDTIC$LKSCPH06$,MDMOBJ
                     OVERLAY                    OVER10
                     INCLUDE LNKEDTIC$LKSCPH51$,MDMOBJ
                     OVERLAY                    OVER9
                     INCLUDE LNKEDTIC$LKSCPH08$,MDMOBJ
                     OVERLAY                    OVER9
                     INCLUDE LNKEDTIC$LKSCPH48$,MDMOBJ
                     OVERLAY                    OVER7
                     INCLUDE LNKEDTIC$LKSCPH70$,MDMOBJ
                     INCLUDE LNKEDTIC$LKSCPH09$,MDMOBJ
                     OVERLAY                    OVER5
                     INCLUDE LNKEDTIC$LKSCPH02$,MDMOBJ
```

Figure 8-6. Link-Edit Example 5 (Part 1 of 14)

```
OVERLAY                              OVER5
INCLUDE  LNKEDTIC@LK SCPH03a,MDH0BJ
OVERLAY  OVER6
INCLUDE  LNKEDTIC@LK SCPH15a,MDH0BJ
OVERLAY  OVER6
INCLUDE  LNKEDTIC@LK SCPH13a,MDH0BJ
OVERLAY  OVER5
INCLUDE  LNKEDTIC@LK SCPH01a,MDH0BJ
OVERLAY  OVER5
INCLUDE  LNKEDTIC@LK SCPH02a,MDH0BJ
ENTER LKS9AUSP
OVERLAY                              OVER5
INCLUDE      LNKEDTIC@LKSCPH030,MDH0BJ
OVERLAY      OVER9
INCLUDE  LNKEDTIC@LK SCPH16a,MDH0BJ
ENTER LKSCPH16
OVERLAY  OVER9
INCLUDE  LNKEDTIC@LK SCPH17a,MDH0BJ
ENTER LKSCPH17
OVERLAY  OVER9
INCLUDE      LNKEDTIC@LKSCPH180,MDH0BJ
ENTER LKSCPH18
OVERLAY  OVER9
INCLUDE  LNKEDTIC@LK SCPH56a,MDH0BJ
ENTER LKSCPH56
OVERLAY  OVER9
INCLUDE  LNKEDTIC@LK SCPH57a,MDH0BJ
ENTER LKSCPH57
OVERLAY                              OVER9
INCLUDE      LNKEDTIC@LKSCPH580,MDH0BJ
ENTER LKSCPH58
OVERLAY  OVER3
INCLUDE  LKSW18LKSW1a,MDH0BJ
OVERLAY  OVER3
INCLUDE  LKSW0LK SWa,MDH0BJ
OVERLAY  OVER3
INCLUDE  LKSW19LKSW1a,MDH0BJ
OVERLAY                              OVER2
INCLUDE  LKSN0LK SNa,MDH0BJ
ENTER LKSN
OVERLAY  OVER2
INCLUDE  LKSLLAST0DATALN6a,MDH0BJ
INCLUDE  LKSLLAST0LK SLLASTa,MDH0BJ
ENTER LKSLLAST
OVERLAY  OVER2
INCLUDE  LKSLLAST0DATASHTa,MDH0BJ
INCLUDE  LKSLLAST0LK SLLASTa,MDH0BJ
ENTER LKSLLAST
OVERLAY  OVER3
INCLUDE  LKSLLAST0LASTTXTa,MDH0BJ
OVERLAY  OVER3
INCLUDE  LKSLLAST0LASTOTHa,MDH0BJ
OVERLAY  OVER3
INCLUDE  LKSLLAST0LASTPHSa,MDH0BJ
OVERLAY  OVER2
INCLUDE  LKST0LK STa,MDH0BJ
OVERLAY  OVER1
INCLUDE  LNKEDTC A0LK SPa,MDH0BJ
INCLUDE  LNKEDTC A0LNKEDTCAa,MDH0BJ
INCLUDE  LKSM0LK SM,LKS92a,MDH0BJ
ENTER LKSM
OVERLAY  OVER1
INCLUDE  LKSM0LK SM2a,MDH0BJ
RES    X'800'                        MIN TBL SPACE
```

Figure 8-6. Link-Edit Example 5 (Part 2 of 14)

DEFINITIONS DICTIONARY

SYMBOL.	TYPE.	PHASE.		ADDRESS.	SYMBOL.	TYPE.	PHASE.		ADDRESS.	SYMBOL.	TYPE.	PHASE.		ADDRESS.
DATALNG	CSECT	38		00000F58	DATASHT	CSECT	39		00000F58	DPSCON0	ENTRY	ROOT		00300000
DPSCON1	ENTRY	ROOT		00000030	DPSCON2	ENTRY	ROOT		00000000	DPSCON3	ENTRY	ROOT		00300000
DPSCON4	ENTRY	ROOT		00000000	DPSCON5	ENTRY	ROOT		00000000	DPSCON6	ENTRY	ROOT		00300000
DPSCON7	ENTRY	ROOT		00000030	KESALP	ENTRY	ABS		00002590	KESRES	ENTRY	ABS		00302090
LASTOTH	CSECT	41		00001620	LASTPHS	CSECT	42		00001620	LASTTXT	CSECT	43		00301620
LB93USR	ENTRY	01	M	00001568	LB93USR	ENTRY	05	M	00001568	LB93USS	ENTRY	01	M	00301868
LB93USS	ENTRY	05	M	00001868	LKSCDIF1	ENTRY	ABS		00000998	LKSCDIF2	ENTRY	ABS		003007C8
LKSCOBPC	ENTRY	01	M	00001818	LKSCOBPC	ENTRY	05	M	00001818	LKSCOFPC	ENTRY	01	M	00301890
LKSCOFPC	ENTRY	05	M	00001890	LKSCPH01	ENTRY	07		000019E8	LKSCPH02	CSECT	21		000019E8
LKSCPH03	CSECT	22		000019E8	LKSCPH04	CSECT	09		00001D10	LKSCPH05	CSECT	13		00301F28
LKSCPH06	CSECT	16		00002250	LKSCPH07	CSECT	14		00002168	LKSCPH08	CSECT	18		00302168
LKSCPH09	CSECT	11	M	00001070	LKSCPH09	CSECT	20	M	00001D40	LKSCPH10	CSECT	08		00301C50
LKSCPH11	CSECT	15		00002250	LKSCPH12	CSECT	10		00002008	LKSCPH13	CSECT	24		00301B90
LKSCPH14	CSECT	12		00001010	LKSCPH15	CSECT	23		00001B40	LKSCPH16	CSECT	28		00301 3E0
LKSCPH17	CSECT	29		00001 3E0	LKSCPH18	CSECT	30		00001 3E0	LKSCPH30	CSECT	11		00001D10
LKSCPH01	CSECT	25		00301 9E8	LKSCPH02	CSECT	26		000019E8	LKSCPH03	CSECT	27		00301 9E8
LKSCPH04	CSECT	19		00002168	LKSCPH51	CSECT	17		00002250	LKSCPH56	CSECT	31		00301 3E0
LKSCPH57	CSECT	32		00001 3E0	LKSCPH58	CSECT	33		00001 3E0	LKSCPH70	CSECT	20		00301010
LKSCPOR6	ENTRY	ABS		00000C50	LKSCPPCA	ENTRY	ROOT		00000 7E8	LKSCREPC	ENTRY	ROOT		003007C0
LKSCRFPC	ENTRY	ROOT		00000640	LKSCRPCA	ENTRY	ROOT		00000618	LKSCRTPC	ENTRY	ROOT		00300798
LKSCUMFP	ENTRY	01	M	00001 9C0	LKSCUMFP	ENTRY	05	M	00001 9C0	LKSCUMPC	ENTRY	01	M	00001998
LKSCUMPC	ENTRY	05	M	00001998	LKSCUPCA	ENTRY	01	M	00001698	LKSCUPCA	ENTRY	05	M	00001698
LKSCUPCF	ENTRY	01	M	00001 6C0	LKSCUPCF	ENTRY	05	M	00001 6C0	LKSDBEXT	ENTRY	ABS		00300006
LKSDBINC	ENTRY	ABS		00000030	LKSDBTRN	ENTRY	ABS		00000000	LKSDCALP	ENTRY	ABS		00303030
LKSDCHAL	ENTRY	ABS		00000002	LKSDCMOF	ENTRY	ABS		00000000	LKSDCORP	ENTRY	ABS		00300030
LKSDDLN	ENTRY	ABS		00000002	LKSDESDA	ENTRY	ABS		00000006	LKSDGTCT	ENTRY	ABS		00300030
LKSDIBCD	ENTRY	ABS		00000000	LKSDLIBS	ENTRY	ABS		00000000	LKSDMHDR	ENTRY	ABS		00300004
LKSDMODP	ENTRY	ABS		00000030	LKSDNESP	ENTRY	ABS		00000002	LKSDNSTR	ENTRY	ABS		00300030
LKSDOPSC	ENTRY	ABS		00000006	LKSDRDB1	ENTRY	ABS		00000008	LKSDSARC	ENTRY	ABS		00300000
LKSDSCNA	ENTRY	ABS		00000034	LKSDSCNC	ENTRY	ABS		00000002	LKSDSCNM	ENTRY	ABS		00300032
LKSDSCN1	ENTRY	ABS		00000002	LKSDSTSC	ENTRY	ABS		00000004	LKSDWRRC	ENTRY	ABS		00300004
LKSEALDF	ENTRY	ABS		00000686	LKSEBCAT	ENTRY	ABS		0000065C	LKSEBDCI	ENTRY	ABS		00300570
LKSEBDLB	ENTRY	ABS		000003F7	LKSEBDLO	ENTRY	ABS		00000631	LKSEBDLI	ENTRY	ABS		00300420
LKSEBDRF	ENTRY	ABS		00000038	LKSEBDSN	ENTRY	ABS		000003CA	LKSEBPAR	ENTRY	ABS		00300 4A7
LKSECSNM	ENTRY	ABS		000006F7	LKSECSNO	ENTRY	ABS		00000130	LKSECSNP	ENTRY	ABS		00300584
LKSEDNOP	ENTRY	ABS		0000003C	LKSEENCM	ENTRY	ABS		000002E8	LKSEENEX	ENTRY	ABS		00300724
LKSEENLA	ENTRY	ABS		000001CE	LKSEENLM	ENTRY	ABS		00000083	LKSEENPC	ENTRY	ABS		003003A8
LKSEEOPE	ENTRY	ABS		000001DE	LKSEEQPH	ENTRY	ABS		000001F0	LKSEILNF	ENTRY	ABS		003005DF
LKSEINVL	ENTRY	ABS		000005F4	LKSEINVO	ENTRY	ABS		0000027C	LKSEIVDE	ENTRY	ABS		00300112
LKSEIVJC	ENTRY	ABS		000006E1	LKSEIVPH	ENTRY	ABS		000004C4	LKSELABN	ENTRY	ABS		003000 9C
LKSELBDE	ENTRY	ABS		00000158	LKSELMLB	ENTRY	ABS		00000188	LKSELNNS	ENTRY	ABS		00300460
LKSELNNF	ENTRY	ABS		00000443	LKSEMIVO	ENTRY	ABS		0000023E	LKSENNDE	ENTRY	ABS		00300255
LKSENNDF	ENTRY	ABS		0000021E	LKSEMOPN	ENTRY	ABS		000002C7	LKSEMPDL	ENTRY	ABS		0030048C
LKSEMPSL	ENTRY	ABS		000006FF	LKSEMTDL	ENTRY	ABS		000003DF	LKSEMTMA	ENTRY	ABS		00300291
LKSENCUP	ENTRY	ABS		00000 34C	LKSENDN	ENTRY	ABS		0000002F	LKSENDS2	ENTRY	ABS		00300391
LKSENODL	ENTRY	ABS		0000019D	LKSENODP	ENTRY	ABS		00000051	LKSENOPH	ENTRY	ABS		00300014
LKSENORN	ENTRY	ABS		00000552	LKSENORS	ENTRY	ABS		000005 2F	LKSENOSC	ENTRY	ABS		00300010
LKSENOSR	ENTRY	ABS		00000 4E3	LKSENOTE	ENTRY	ABS		00000 32C	LKSENOTF	ENTRY	ABS		00300 17F
LKSENOUS	ENTRY	ABS		00000512	LKSEOPAB	ENTRY	ABS		000000AE	LKSEOPRN	ENTRY	ABS		00300 0C4
LKSEORNF	ENTRY	ABS		00000237	LKSEOVAB	ENTRY	ABS		00000740	LKSEOVEX	ENTRY	ABS		00300760
LKSEOVOP	ENTRY	ABS		00000783	LKSEOVRG	ENTRY	ABS		00000780	LKSEOVRT	ENTRY	ABS		00300003
LKSEPNO	ENTRY	ABS		0000001A	LKSEREGN	ENTRY	ABS		0000006E	LKSERNAL	ENTRY	ABS		00300598
LKSERTIO	ENTRY	ABS		00300010	LKSESLIB	ENTRY	ABS		00000704	LKSESOBJ	ENTRY	ABS		00300702
LKSESTBD	ENTRY	ABS		00300004	LKSETBDF	ENTRY	ABS		0000068C	LKSEUOBJ	ENTRY	ABS		00300703
LKSEWTYP	ENTRY	ABS		00000371	LKSEXNO	ENTRY	ABS		000000E1	LKSFAIM	ENTRY	ABS		0030063C
LKSFRLN	ENTRY	ABS		0000061C	LKSILDNO	ENTRY	06		00002240	LKSIO300	ENTRY	06		00301F7A
LKSIO320	ENTRY	06		00001F82	LKSIO340	ENTRY	01		00001F66	LKSIO350	ENTRY	01		00301F72

Figure 8-6. Link-Edit Example 5 (Part 3 of 14)

```
LKSLENSZ  ENTRY  ABS      00000034    LKSLLAST  CSECT  38   M 00001190    LKSLLAST  CSECT  39   M 00301190
LKSLPSTL  ENTRY  ABS      0000 1E48   LKSLPSTS  ENTRY  ABS      00000F78    LKSLPTOV  ENTRY  38   M 00301386
LKSLPTOV  ENTRY  39   M   00001346   LKSM      CSECT  99       00001168    LKSM1     CSECT  39       00301168
LKSM2     CSECT  99   M   00001200   LKSM2     CSECT  95   M   00000E50    LKSM2LEN  ENTRY  ABS      00300960
LKSN      CSECT  37        00000F58   LKSP      CSECT  01   M   00000E50    LKSP      CSECT  02   M   00300E50
LKSP      CSECT  99   M   00000E50   LKSBCMD1  ENTRY  06       00002DAC    LKST      CSECT  93       00300F58
LKSW      CSECT  35        00001168   LKSW1     CSECT  36       00001168    LKS3P09L  ENTRY  ABS      00302028
LKS94AUSP ENTRY  26        0000 19EA  LNKEDT    CSECT  ROOT     00000B10    LNKEDTCA  CSECT  01   M   00300F58
LNKEDTCA  CSECT  03   M   00000F58   LNKEDTCA  CSECT  99   M   00000F58    LNKEDTIC  CSECT  07       00301968
LNKEDTIN  CSECT  01        0000 19E8  LNKEDTI1  CSECT  06       0000 19E8   LNKEDTPT  CSECT  01   M   00301 3E0
LNKEDTPT  CSECT  05   M   0000 13E0  LNKEDTSA  CSECT  01   M   00001168    LNKEDTSA  CSECT  06   M   00301168
LNKEDT1   CSECT  ROOT     00300968   LNKEDT2   CSECT  01   M   00001568    LNKEDT2   CSECT  05   M   00301568
PHNOATIN  ENTRY  ABS      0000F3F2   PHNOATLG  ENTRY  ABS      00 00F3F3   PHNOATSH  ENTRY  ABS      00 30F3F1
PHNOCTAT  ENTRY  ABS      0000F2F6   PHNOGTH   ENTRY  ABS      00 00F9F1   PHNOPHS   ENTRY  ABS      00 30F9F2
PHNOPH01  ENTRY  ABS      00000037   PHNOPHJ2  ENTRY  ABS      00 000021   PHNOPH03  ENTRY  ABS      00 300022
PHNOPH04  ENTRY  ABS      00000009   PHNOPH05  ENTRY  ABS      00 000013   PHNOPH36  ENTRY  ABS      00 300016
PHNOPH07  ENTRY  ABS      00000014   PHNOPH08  ENTRY  ABS      00 000018   PHNOPH09  ENTRY  ABS      00 300011
PHNOPH10  ENTRY  ABS      00000008   PHNOPH11  ENTRY  ABS      00 000015   PHNOPH12  ENTRY  ABS      00 300010
PHNOPH13  ENTRY  ABS      00000029   PHNOPH14  ENTRY  ABS      00 000012   PHNOPH15  ENTRY  ABS      00 300023
PHNOPH91  ENTRY  ABS      00000025   PHNOPH92  ENTRY  ABS      00 000026   PHNOPH93  ENTRY  ABS      00 000027
PHNOPH98  ENTRY  ABS      00000019   PHNOPH99  ENTRY  ABS      00 000020   PHNOPH51  ENTRY  ABS      00 300017
PHNOSPIN  ENTRY  ABS      0000F2F9   PHNOSPLG  ENTRY  ABS      00 00F3F0   PHNOSPSH  ENTRY  ABS      00 30F2F8
PHNOTXT   ENTRY  ABS      00300F9F0  PNOLKSIN  ENTRY  ABS      00 00F0F1   PNOLKSI2  ENTRY  ABS      00 30F0F6
PNOLKSLL  ENTRY  ABS      0000F3F8   PNOLKSLS  ENTRY  ABS      00 00F3F9   PNOLKSM   ENTRY  ABS      00 30F9F9
PNOLKSM1  ENTRY  ABS      00300F3F9  PNOLKSM2  ENTRY  ABS      00 00F9F5   PNOLKSN   ENTRY  ABS      00 30F3F7
PNOLKST   ENTRY  ABS      0000F9F3   PNOLKSW   ENTRY  ABS      00 00F3F5   PNOLKSW1  ENTRY  ABS      00 30F3F6
PRNTR     ENTRY  01   M   00000CE8   PRNTR     ENTRY  02   M   00 000EE8   PRNTR     ENTRY  99   M   00 30 0EE8
PRNTRC    ENTRY  01   M   00000F1A   PRNTRC    ENTRY  02   M   00 000F1A   PRNTRC    ENTRY  99   M   00 30 0F1A
PRSYSA    CSECT  ROOT     00000000   SSCR      ENTRY  ROOT     00000668    SYSOBJ    ENTRY  01   M   00 3016E8
SYSOBJ    ENTRY  05   M   0000 16E8   SYSRUN    ENTRY  ROOT     00000 9E0
```

Figure 8-6. Link-Edit Example 5 (Part 4 of 14)

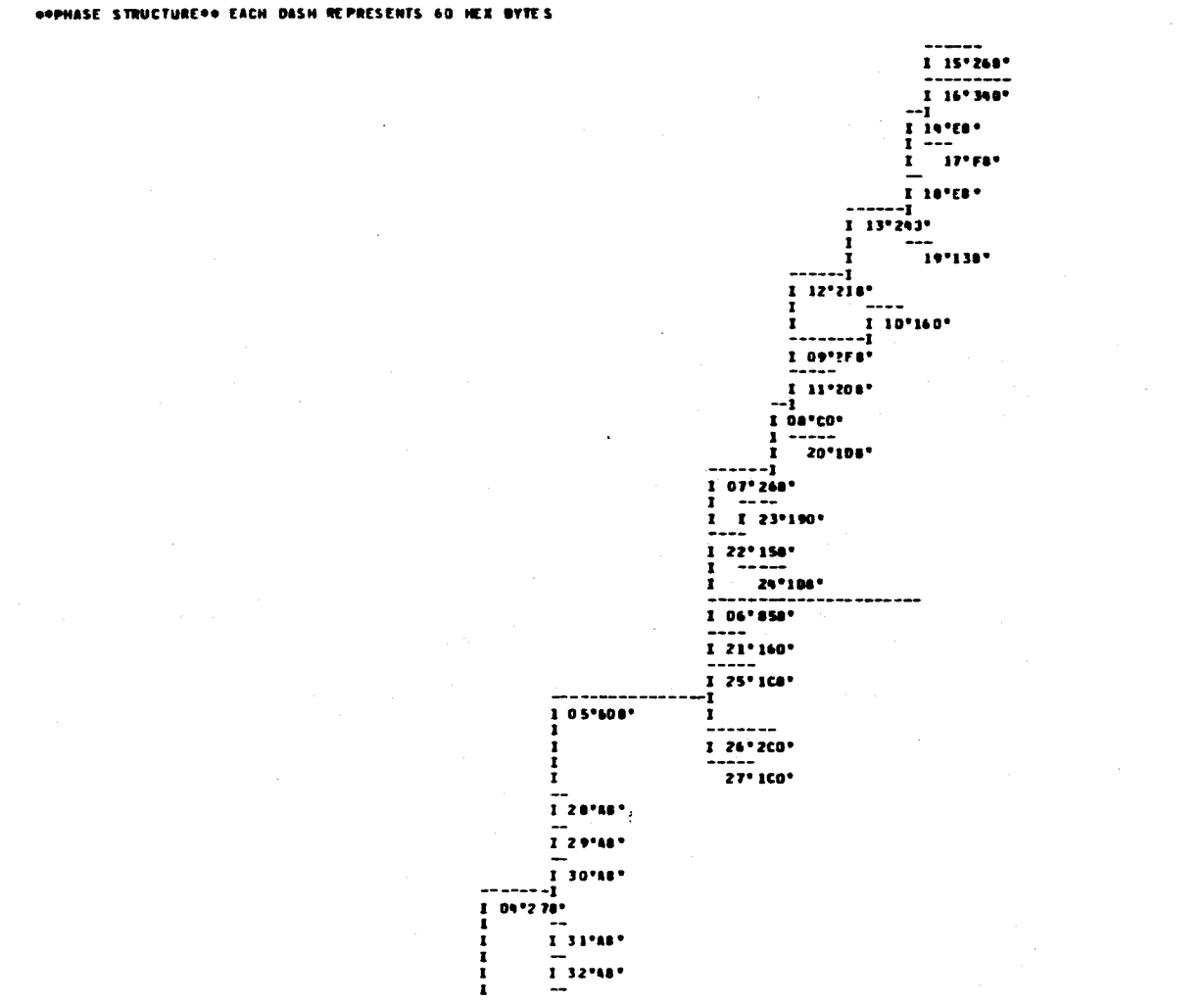


Figure 8-6. Link-Edit Example 5 (Part 5 of 14)

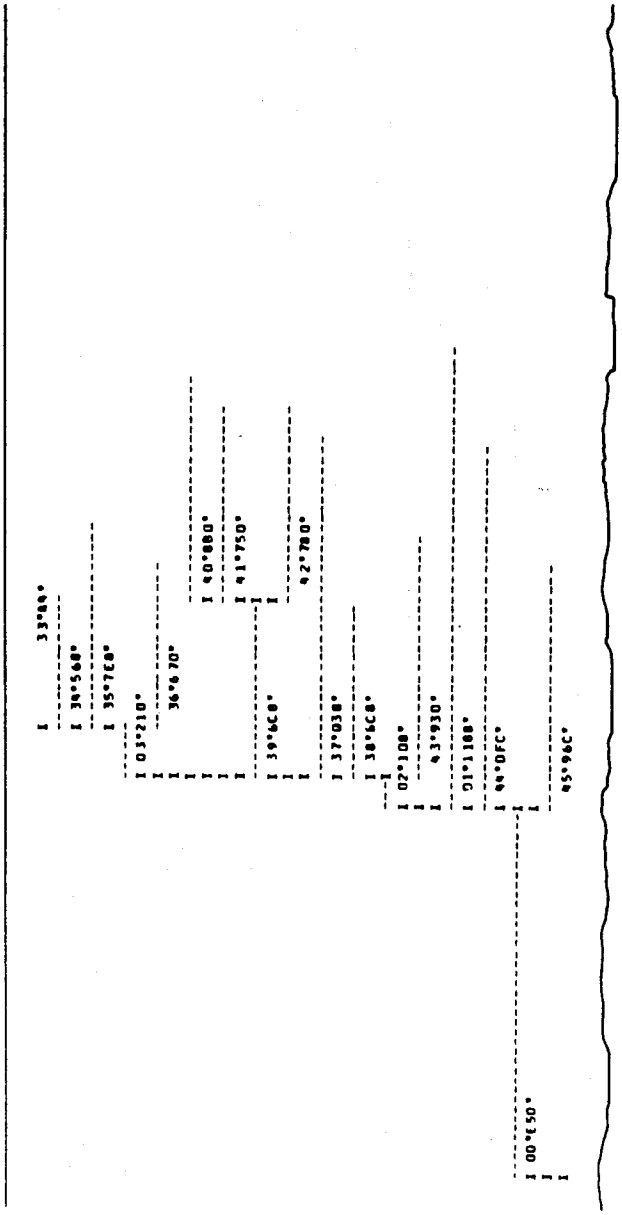


Figure 8-6. Link-Edit Example 5 (Part 6 of 14)

```
                              ** ALLOCATION MAP **

              LOAD MODULE -   LNKEDT        SIZE -   00002090

PHASE NAME  TRANS ADDR    FLAG    LABEL      TYPE      ESID       LNK ORG     HEADDR      LENGTH      OBJ ORG
LNKEDT00       NODE - ROOT                                        00000000    00000E50    00000E50
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS ***
      -  12/07/88  08.31  -        PRSYSA      OBJ
                                   PRSYSA     CSECT      02       00000000    000004E6    000004E6    00000030
                                   DPSCOM4       ENTRY   02       00000000                            00000000
                                   DPSCOM3       ENTRY   02       00000000                            00000030
                                   DPSCOM7       ENTRY   02       00000000                            00000030
                                   DPSCOM0       ENTRY   02       00000000                            00000000
                                   DPSCOM1       ENTRY   02       00000000                            00000000
                                   DPSCOM6       ENTRY   02       00000000                            00000030
                                   DPSCOM2       ENTRY   02       00000000                            00000030
                                   DPSCOM5       ENTRY   02       00000000                            00000030
      -  12/07/88  08.31  -        LNKEDTCA    OBJ
                                   LNKEDT1    CSECT      19       000004E8    00000B18    00000630    00001548
                                   SYSRUN        ENTRY   19       000004E8                            00001548
                                   LKSCRPCA      ENTRY   19       00000618                            00001678
                                   LKSCRFPC      ENTRY   19       00000640                            000016A0
                                   SSCR          ENTRY   19       00000668                            000016C8
                                   LKSCRTPC      ENTRY   19       00000798                            000017F8
                                   LKSCREPC      ENTRY   19       000007C0                            00001820
                                   LKSCPPCA      ENTRY   19       000007E8                            00001848
      -  12/07/88  08.31  -        LNKEDTCA    OBJ
                                   LNKEDT     CSECT      18       00000818    00000E50    00000338    00001210
             00000818
LNKEDT01       NODE -   OVER1                                     00000E50    00001FD8    00001188
      -  12/07/88  08.31  -        LNKEDTCA    OBJ
                                   LKSP       CSECT      1A       00000E50    00000F58    00000108    00001878
                                   PRNTR         ENTRY   1A       00000EE8                            00001C10
                                   PRNTRC        ENTRY   1A       00000F1A                            00001C42
      -  12/07/88  08.31  -        LNKEDTCA    OBJ
                                   LNKEDTCA   CSECT      02       00000F58    00001168    00000210    00001030
                                   LNKEDTSA   CSECT      1B       00001168    000013E0    00000278    00001C80
                                   LNKEDTPT   CSECT      1C       000013E0    00001568    00000188    00001EF8
                                   LNKEDT2    CSECT      1D       00001568    000019E8    00000480    00002080
                                   LKSCOFPC      ENTRY   1D       00001840                            00002358
                                   LB93USS       ENTRY   1D       00001868                            00002380
                                   LKSCUMPC      ENTRY   1D       00001998                            000024B0
                                   LKSCUMFP      ENTRY   1D       000019C0                            000024D8
                                   LB93USR       ENTRY   1D       00001568                            00002080
                                   LKSCUPCA      ENTRY   1D       00001698                            000021B0
                                   LKSCUPCF      ENTRY   1D       000016C0                            000021D8
                                   SYSOBJ        ENTRY   1D       000016E8                            00002200
                                   LKSCOBPC      ENTRY   1D       00001818                            00002330
      -  12/07/88  08.31  -        LNKEDTIN    OBJ
                                   LNKEDTIN   CSECT      02       000019E8    00001FD8    000005F0    00001030
                                   LKSI0340      ENTRY   02       00001F66                            0000157E
                                   LKSI0350      ENTRY   02       00001F72                            0000158A
             000019E8
LNKEDT02       NODE -   OVER1                                     00000E50    00000F58    00000108
      -  12/07/88  08.31  -        LNKEDTCA    OBJ
                                   LKSP       CSECT      1A       00000E50    00000F58    00000108    00001878
                                   PRNTR         ENTRY   1A       00000EE8                            00001C10
                                   PRNTRC        ENTRY   1A       00000F1A                            00001C42
             00000E50
LNKEDT03       NODE -   OVER2                                     00000F58    00001168    00000210
```

Figure 8-6. Link-Edit Example 5 (Part 7 of 14)

```
PHASE NAME   TRANS ADDR    FLAG      LABEL     TYPE     ESID     LNK ORG    HIADDR    LENGTH    OBJ ORG
         -  12/07/88 08.31  -       LNKEDTCA  OBJ
                                     LNKEDTCA  CSECT    32       00000F58   00301168  00300210  00301030

                00000F58
LNKEDT04         NODE  -     OVER3                               00001168   003013E0  00300278
         -  12/07/88 08.31  -       LNKEDTCA  OBJ
                                     LNKEDTSA  CSECT    1B       00001168   003013E0  00300278  00301C80

                00001168
LNKEDT05         NODE  -     OVER4                               000013E0   00301968  00300638
         -  12/07/88 08.31  -       LNKEDTCA  OBJ
                                     LNKEDTPT  CSECT    1C       000013E0   00301568  00300188  00301EF8
                                     LNKEDT2   CSECT    1D       00001568   00301968  00300480  00302080
                                     LKSCOFPC  ENTRY    1D       00001840                       00302350
                                     LB93USS   ENTRY    1D       00001868                       00302380
                                     LKSCUNPC  ENTRY    1D       00001998                       00302480
                                     LKSCUNFP  ENTRY    1D       000019C0                       00302408
                                     LB93USR   ENTRY    1D       00001568                       00302080
                                     LKSCUPCA  ENTRY    1D       00001698                       00302180
                                     LKSCUPCF  ENTRY    1D       000016C0                       00302108
                                     SYSOBJ    ENTRY    1D       000016E8                       00302230
                                     LKSCOBPC  ENTRY    1D       00001818                       00302330

                00003F58
LNKEDT06         NODE  -     OVER5                               000019E8   00302240  00300858
         -  12/07/88 08.31  -       LNKEDTIN  OBJ
                                     LNKEDTI1  CSECT    1C       000019E8   00302240  00300858  00302AF0
                                     LKSILDND  ENTRY    1C       00002240                       00303398
                                     LKSSCMD1  ENTRY    1C       000020AC                       00303184
                                     LKSI0300  ENTRY    1C       00001F7A                       00303002
                                     LKSI0320  ENTRY    1C       00001F82                       00303008

                000019E8
LNKEDT07         NODE  -     OVER5                               000019E8   00301C50  00300268
         -  12/07/88 08.31  -       LNKEDTIC  OBJ
                                     LNKEDTIC  CSECT    32       000019E8   00301C50  00300268  00300000
                                     LKSDCALP  ENTRY    30       00000000                       00303030
                                     LKSDBINC  ENTRY    30       00000000                       00303030
                                     LKSDCOMP  ENTRY    30       00000000                       00303030
                                     LKSDSCNM  ENTRY    30       00000002                       00300002
                                     LKSDWRRC  ENTRY    30       0000000A                       0030003A
                                     LKSDRDB1  ENTRY    30       00000008                       00303038
                                     LKSDBTRN  ENTRY    30       00000000                       00303030
                                     LKSDCMDF  ENTRY    30       00000000                       00303030
                                     LKSDSARC  ENTRY    30       00003000                       00303030
                                     LKSDSCNC  ENTRY    30       00000002                       00300002
                                     LKSDLIBS  ENTRY    30       00000000                       00303030
                                     LKSDSCNA  ENTRY    30       00000004                       00303034
                                     LKSDSTSC  ENTRY    30       00000004                       00303034
                                     LKSDESDA  ENTRY    00       00000006                       00303036
                                     LKSDSCN1  ENTRY    30       00000002                       00303032
                                     LKSDMHDR  ENTRY    30       00000004                       00303034
                                     LKSDCHAL  ENTRY    30       00000002                       00303032
                                     LKSDDLN   ENTRY    30       00000002                       00303032
                                     LKSDBEXT  ENTRY    30       00000006                       00303036
                                     LKSDGTCT  ENTRY    30       00000000                       00303030
                                     LKSCPHJ1  ENTRY    32       000019E8                       00303030
                                     LKS3P09L  ENTRY    30       00000208                       00302208
                                     LKSCPOR6  ENTRY    30       00003C50                       00303C50
                                     LKSDIBCD  ENTRY    30       00000000                       00303030
                                     LKSDNSTK  ENTRY    30       00000000                       00303000
                                     LKSDWESP  ENTRY    30       00303002                       00303002
                                     LKSDMODP  ENTRY    30       00000000                       00303030
                                     LKSDOPSC  ENTRY    30       00000006                       00303036
```

Figure 8-6. Link-Edit Example 5 (Part 8 of 14)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HEADDR      LENGTH      OBJ ORG
                                      LKSCDIF1   ENTRY     30        00000998                            00308998
                                      LKSCDIF2   ENTRY     30        000007C8                            003007C8
                 00001928
LNKEDT08         NODE -    OVER6                                    00001C50    00301D10    00300CC0
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH10   CSECT     27        00001C50    00301D10    00300CC0    00302D90
                 00001C53
LNKEDT09         NODE -    OVER7                                    00001D10    00302038    003002F8
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH04   CSECT     20        00001D10    00302038    003002F8    00302D68
                 00001D13
LNKEDT10         NODE -    OVER8                                    00002008    00302168    00900160
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH12   CSECT     2A        00002008    00302168    00300160    00303180
                 00002008
LNKEDT11         NODE -    OVER7                                    00001D10    00301F18    00300238
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH30   CSECT     2E        00001D10    00301D70    00300060    00303890
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH39   CSECT     26        00001D70    00301F18    00300188    003028E8
                 00001D13
LNKEDT12         NODE -    OVER7                                    00001D10    00301F28    00300218
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH14   CSECT     2C        00001D10    00301F28    00300218    003034E8
                 00001D13
LNKEDT13         NODE -    OVER8                                    00001F28    00302168    00300290
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH35   CSECT     21        00001F28    00302168    00300290    00302360
                 00001F28
LNKEDT14         NODE -    OVER9                                    00002168    00302250    003000E8
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH37   CSECT     23        00002168    00302250    003000E8    003028E0
                 00002168
LNKEDT15         NODE -    OVER10                                   00002250    00302488    00300268
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH11   CSECT     28        00002250    00302488    00300268    00302E50
                 00002253
LNKEDT16         NODE -    OVER10                                   00002250    00302590    00300340
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH06   CSECT     22        00002250    00302590    00300340    00302540
                 00002253
LNKEDT17         NODE -    OVER10                                   00002250    00302348    003000F8
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH51   CSECT     29        00002250    00302348    003000F8    00303088
                 00002253
LNKEDT18         NODE -    OVER9                                    00002168    00302250    003000E8
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH38   CSECT     24        00002168    00302250    003000E8    003029C8
                 00002168
LNKEDT19         NODE -    OVER9                                    00002168    00302240    00300138
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH48   CSECT     25        00002168    00302240    00300138    00302A80
                 00002168
LNKEDT20         NODE -    OVER7                                    00001D10    00301EE8    00300188
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH70   CSECT     2F        00001D10    00301D90    00300030    003038F0
        -  12/07/88 08.31  -          LNKEDTIC   OBJ
                                      LKSCPH39   CSECT     26        00001D90    00301EE8    003001A8    003028E8
                 00001D13
LNKEDT21         NODE -    OVER5                                    009019E8    00301898    00300160
```

Figure 8-6. Link-Edit Example 5 (Part 9 of 14)

PHASE NAME	TRANS ADDR	FLAG
-	12/07/88 08.31	-
	000019E8	
LNKEDT22	NODE -	OVER5
-	12/07/88 08.31	-
	000019E8	
LNKEDT23	NODE -	OVER6
-	12/07/88 08.31	-
	00001840	
LNKEDT24	NODE -	OVER6
-	12/07/88 08.31	-
	00001840	
LNKEDT25	NODE -	OVER5
-	12/07/88 08.31	-
	000019E8	
LNKEDT26	NODE -	OVER5
-	12/07/88 08.31	-
	000019EA	
LNKEDT27	NODE -	OVER5
-	12/07/88 08.31	-
	000019E8	
LNKEDT28	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT29	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT30	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT31	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT32	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT33	NODE -	OVER4
-	12/07/88 08.31	-
	000013E0	
LNKEDT34	NODE -	OVER3
-	12/07/88 08.31	-
	00001168	
LNKEDT35	NODE -	OVER3
-	12/07/88 08.31	-
	00001168	

LABEL	TYPE	ESID	LNK ORG	HEADER	LENGTH	OBJ ORG
LNKEDTIC	OBJ					
LKSCPH32	CSECT	1C	000019E8	0030 1848	0030 0160	0030 1930
			000019E8	0030 1840	0030 0158	
LNKEDTIC	OBJ					
LKSCPH33	CSECT	1E	000019E8	0030 1840	0030 0158	0030 1D50
			00001840	0030 1CD0	0030 0190	
LNKEDTIC	OBJ					
LKSCPH15	CSECT	2D	00001840	0030 1CD0	0030 0190	0030 3730
			00001840	0030 1D18	0030 0188	
LNKEDTIC	OBJ					
LKSCPH13	CSECT	2B	00001840	0030 1D18	0030 0188	0030 3310
			000019E8	0030 1880	0030 01C8	
LNKEDTIC	OBJ					
LKSCPH01	CSECT	18	000019E8	0030 1880	0030 01C8	0030 1768
			000019E8	0030 1CA8	0030 02C0	
LNKEDTIC	OBJ					
LKSCPH02	CSECT	1D	000019E8	0030 1CA8	0030 02C0	0030 1A90
LKS$AUSP	ENTRY	1D	000019EA			0030 1A92
			000019E8	0030 1848	0030 01C0	
LNKEDTIC	OBJ					
LKSCPH03	CSECT	1F	000019E8	0030 1848	0030 01C0	0030 1EA8
			000013E0	0030 1488	0030 00A8	
LNKEDTIC	OBJ					
LKSCPH16	CSECT	30	000013E0	0030 1488	0030 00A8	0030 3920
			000013E0	0030 1488	0030 00A8	
LNKEDTIC	OBJ					
LKSCPH17	CSECT	32	000013E0	0030 1488	0030 00A8	0030 3A70
			000013E0	0030 1488	0030 00A8	
LNKEDTIC	OBJ					
LKSCPH18	CSECT	34	000013E0	0030 1488	0030 00A8	0030 3BC0
			000013E0	0030 1488	0030 00A8	
LNKEDTIC	OBJ					
LKSCPH56	CSECT	31	000013E0	0030 1488	0030 00A8	0030 39C8
			000013E0	0030 1488	0030 00A8	
LNKEDTIC	OBJ					
LKSCPH57	CSECT	33	000013E0	0030 1488	0030 00A8	0030 3B18
			000013E0	0030 1484	0030 00A4	
LNKEDTIC	OBJ					
LKSCPH58	CSECT	35	000013E0	0030 1484	0030 00A4	0030 3C68
			00001168	0030 1600	0030 0568	
LKSM1	OBJ					
LKSM1	CSECT	32	00001168	0030 1600	0030 0568	0030 3030
			00001168	0030 1950	0030 07E8	
LKSW	OBJ					
LKSW	CSECT	32	00001168	0030 1950	0030 07E8	0030 3030

Figure 8-6. Link-Edit Example 5 (Part 10 of 14)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL      TYPE      ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
LNKEDT36       NODE -    OVER3                                      00001168    00301708    00300670
        -  12/07/88 08.31  -          LKSW1        OBJ
                                      LKSW1      CSECT      32      00001168    00301708    00300670    00300000

               00001168
LNKEDT37       NODE -    OVER2                                      00000F58    00301C90    00300D38
        -  12/07/88 08.31  -          LKSN         OBJ
                                      LKSN       CSECT      32      00000F58    00301C90    00300D38    00300000

               00000F58
LNKEDT38       NODE -    OVER2                                      00000F58    00301620    003006C8
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      DATALNG    CSECT      23      00000F58    00301190    00300238    00302AF0
                                      LKSLENSZ     ENTRY      30      00000004                            00300034
                                      LKSLPSTS     ENTRY      30      00000F78                            00300F78
                                      LKSLPSTL     ENTRY      30      00001E98                            00301E98
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      LKSLLAST   CSECT      32      00001190    00301620    00300490    00300030
                                      LKSLPTOV     ENTRY      32      000013A6                            00300216

               00001193
LNKEDT39       NODE -    OVER2                                      00000F58    00301620    003006C8
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      DATASHT    CSECT      22      00000F58    00301190    00300238    00302880
        - 12/07/91 08.31 -            LKSLLAST     OBJ
                                      LKSLLAST   CSECT      32      00001190    00301620    00300490    00300030
                                      LKSLPTOV     ENTRY      32      000013A6                            00300216

               00001193
LNKEDT40       NODE -    OVER3                                      00001620    00301E00    00300880
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      LASTTXT    CSECT      1F      00001620    00301E00    00300880    00301138

               00001623
LNKEDT41       NODE -    OVER3                                      00001620    00301D70    00300750
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      LASTOTH    CSECT      20      00001620    00301D70    00300750    00301968

               00001623
LNKEDT42       NODE -    OVER3                                      00001620    00301DA0    00300780
        -  12/07/88 08.31  -          LKSLLAST     OBJ
                                      LASTPHS    CSECT      21      00001620    00301DA0    00300780    00302138

               00001623
LNKEDT43       NODE -    OVER2                                      00000F58    00301888    00300930
        -  12/07/88 08.31  -          LKST         OBJ
                                      LKST       CSECT      32      00000F58    00301888    00300930    00300030
                                      LKSENOSC     ENTRY      00      00000018                            00300018
                                      PHNOPH01     ENTRY      00      00000007                            00300037
                                      PHNOPH02     ENTRY      00      00000021                            00300021
                                      PHNOPH03     ENTRY      00      00000022                            00300022
                                      PHNOPH04     ENTRY      00      00000009                            00300029
                                      PHNOPH05     ENTRY      00      00000013                            00300013
                                      PHNOPH06     ENTRY      00      00000016                            00300016
                                      PHNOPH07     ENTRY      00      00000019                            00300019
                                      PHNOPH08     ENTRY      00      00000018                            00300018
                                      PHNOPH09     ENTRY      00      00000011                            00300011
                                      PHNOPH10     ENTRY      00      00000008                            00300038
                                      PHNOPH11     ENTRY      00      00000015                            00300015
                                      PHNOPH12     ENTRY      00      00000010                            00300010
                                      PHNOPH13     ENTRY      00      00000024                            00300024
                                      PHNOPH14     ENTRY      00      00000012                            00300012
                                      PHNOPH15     ENTRY      00      00000023                            00300023
                                      PHNOPH01     ENTRY      00      00000025                            00300025
                                      PHNOPH02     ENTRY      00      00000026                            00300026
                                      PHNOPH03     ENTRY      00      00000027                            00300027
                                      PHNOPH08     ENTRY      00      00000019                            00300019
```

Figure 8-6. Link-Edit Example 5 (Part 11 of 14)

```
PHASE NAME   TRANS ADDR     FLAG      LABEL     TYPE     ESID      LNK ORG     HIADDR      LENGTH      OBJ ORG
                                      PHNOPH99  ENTRY    30      00000020                            00300020
                                      PHNOPH51  ENTRY    30      00000017                            00300017
                                      PHNOSPSH  ENTRY    30      0000F2F8                            0030F2F8
                                      PHNOSPIN  ENTRY    30      0000F2F9                            0030F2F9
                                      PHNOSPLG  ENTRY    30      0000F3F0                            0030F3F0
                                      PHNOATSH  ENTRY    30      0000F3F1                            0030F3F1
                                      PHNOATIN  ENTRY    30      0000F3F2                            0030F3F2
                                      PHNOATLG  ENTRY    30      0000F3F3                            0030F3F3
                                      PHNOCTAT  ENTRY    30      0000F2F6                            0030F2F6
                                      PNDLKSN2  ENTRY    30      0000F9F5                            0030F9F5
                                      PNDLKSI2  ENTRY    30      0000F0F6                            0030F0F6
                                      PNDLKSIN  ENTRY    30      0000F0F1                            0030F0F1
                                      PNDLKSN   ENTRY    30      0000F9F9                            0030F9F9
                                      PNDLKSN1  ENTRY    30      0000F3F4                            0030F3F4
                                      PNDLKSW   ENTRY    30      0000F3F5                            0030F3F5
                                      PNDLKSW1  ENTRY    30      0000F3F6                            0030F3F6
                                      PNDLKSN   ENTRY    30      0000F3F7                            0030F3F7
                                      PNDLKSLL  ENTRY    30      0000F3F8                            0030F3F8
                                      PNDLKSLS  ENTRY    30      0000F3F9                            0030F3F9
                                      PHNOTXT   ENTRY    30      0000F9F0                            0030F9F0
                                      PHNOOTH   ENTRY    30      0000F9F1                            0030F9F1
                                      PHNOPHS   ENTRY    30      0000F9F2                            0030F9F2
                                      PNDLKST   ENTRY    30      0000F9F3                            0030F9F3
                                      LKSESTBD  ENTRY    30      00000009                            00300009
                                      LKSEBDRF  ENTRY    30      00000008                            00300008
                                      LKSEDNOP  ENTRY    30      0000000C                            0030000C
                                      LKSERTIO  ENTRY    30      00000010                            00300010
                                      LKSENOPH  ENTRY    30      00000019                            00300019
                00000F58
LNKEDT99        NODE  -   OVER1                                   00000E50    0030 1C9C   0030 00FC
        -   12/07/88  08.31  -     LNKEDTCA  OBJ
                                      LKSP      CSECT    1A      00000E50    0030 00F8   0030 0138   0030 1B78
                                      PRNTR     ENTRY    1A      00000EE8                            0030 1C10
                                      PRNTRC    ENTRY    1A      00000F1A                            0030 1C92
        -   12/07/88  08.31  -     LNKEDTCA  OBJ
                                      LNKEDTCA  CSECT    32      00000F58    0030 1168   0030 0210   0030 1030
        -   12/07/88  08.31  -     LKSM      OBJ
                                      LKSM      CSECT    32      00001168    0030 12E0   0030 0178   0030 0030
                                      LKSM2     CSECT    18      000012E0    0030 1C9C   0030 096C   0030 00F8
                                      LKSESOBJ  ENTRY    30      000007D2                            003007D2
                                      LKSEUOBJ  ENTRY    30      000007D3                            003007D3
                                      LKSESLIB  ENTRY    30      000007D9                            003007D9
                                      LKSM2LEN  ENTRY    30      0000096C                            0030096C
                                      LKSEOVRT  ENTRY    30      00000003                            00300033
                                      LKSEPND   ENTRY    30      0000001A                            0030001A
                                      LKSENDN   ENTRY    30      0000002F                            0030002F
                                      LKSENOOP  ENTRY    30      00000051                            00300051
                                      LKSEREGN  ENTRY    00      0000006E                            0030006E
                                      LKSEENLM  ENTRY    30      00000083                            00300083
                                      LKSELABN  ENTRY    30      0000009C                            0030009C
                                      LKSEOPAB  ENTRY    30      000000AE                            003000AE
                                      LKSEOPRN  ENTRY    30      000000C9                            003000C9
                                      LKSEXND   ENTRY    00      000000E1                            003000E1
                                      LKSECSNM  ENTRY    00      000000F7                            003000F7
                                      LKSEIVDE  ENTRY    30      00000112                            00300112
                                      LKSECSNO  ENTRY    30      00000130                            00300130
                                      LKSELBDE  ENTRY    30      00000158                            00300158
                                      LKSENOTF  ENTRY    00      0000017F                            0030017F
                                      LKSENODL  ENTRY    30      00000190                            00300190
                                      LKSELNLB  ENTRY    30      00000188                            00300188
```

Figure 8-6. Link-Edit Example 5 (Part 12 of 14)

```
PHASE NAME  TRANS ADDR     FLAG     LABEL     TYPE     ESID     LNK ORG     HIADDR     LENGTH     OBJ ORG
                                     LKSEENLA  ENTRY    00       000001CE                         003001CE
                                     LKSEEOPE  ENTRY    00       000001DE                         003001DE
                                     LKSEEQPH  ENTRY    00       000001F0                         003001F0
                                     LKSEORNF  ENTRY    00       00000207                         00300207
                                     LKSEMNDF  ENTRY    00       0000021E                         0030021E
                                     LKSEMIVD  ENTRY    00       0000023E                         0030023E
                                     LKSEMNDE  ENTRY    00       00000255                         00300255
                                     LKSEINVO  ENTRY    00       0000027C                         0030027C
                                     LKSEMTMA  ENTRY    00       00000291                         00300291
                                     LKSEMOPN  ENTRY    00       000002C7                         003002C7
                                     LKSEENCM  ENTRY    00       000002E8                         003002E8
                                     LKSEMTDL  ENTRY    00       0000030F                         0030030F
                                     LKSEMOTE  ENTRY    00       0000032C                         0030032C
                                     LKSEMCUP  ENTRY    00       0000034C                         0030034C
                                     LKSEWTYP  ENTRY    00       00000371                         00300371
                                     LKSEWDSZ  ENTRY    00       00000391                         00300391
                                     LKSEENPC  ENTRY    00       000003A8                         003003A8
                                     LKSEBDSN  ENTRY    00       000003CA                         003003CA
                                     LKSEBDLB  ENTRY    00       000003F7                         003003F7
                                     LKSEBDLI  ENTRY    00       00000420                         00300420
                                     LKSELNNF  ENTRY    00       00000443                         00300443
                                     LKSELMNS  ENTRY    00       00000468                         00300468
                                     LKSEMPOL  ENTRY    00       0000048C                         0030048C
                                     LKSEBPAR  ENTRY    00       000004A7                         003004A7
                                     LKSEIVPM  ENTRY    00       000004C4                         003004C4
                                     LKSEMOSR  ENTRY    00       000004E3                         003004E3
                                     LKSEMOUS  ENTRY    00       00000512                         00300512
                                     LKSEMORS  ENTRY    00       0000052F                         0030052F
                                     LKSEMORN  ENTRY    00       00000552                         00300552
                                     LKSEBDCI  ENTRY    00       00000570                         00300570
                                     LKSERNAL  ENTRY    00       00000598                         00300598
                                     LKSECSNP  ENTRY    00       00000584                         00300584
                                     LKSEILNF  ENTRY    00       000005DF                         003005DF
                                     LKSEINVL  ENTRY    00       000005F4                         003005F4
                                     LKSFAIM   ENTRY    00       0000060C                         0030060C
                                     LKSFRLM   ENTRY    00       0000061C                         0030061C
                                     LKSEBDLD  ENTRY    00       00000631                         00300631
                                     LKSEBCNT  ENTRY    00       0000065C                         0030065C
                                     LKSETBDF  ENTRY    00       0000068C                         0030068C
                                     LKSEALDF  ENTRY    00       00000686                         00300686
                                     LKSEIVJC  ENTRY    00       000006E1                         003006E1
                                     LKSEMPSL  ENTRY    00       000006FF                         003006FF
                                     LKSEENEX  ENTRY    00       00000729                         00300729
                                     LKSEOVAB  ENTRY    00       00000748                         00300748
                                     LKSEOVEX  ENTRY    00       00000760                         00300760
                                     LKSEOVRG  ENTRY    00       0000078C                         0030078C
                                     LKSEOVOP  ENTRY    00       00000783                         00300783
                00001168
LNKEDT45        MODE -    OVER1                                   00000E50   00301780   0030096C
        - 12/07/88 08.31 -           LKSM      OBJ
                                     LKSM2     CSECT    18       00000E50   00301780   0030096C   003000F8
                00001168
```

Figure 8-6. Link-Edit Example 5 (Part 13 of 14)

```
                                        FLAG CODES -
A - ALN DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE "A" REF   G - GENERATED EXTRN     I - INCLUSIVE "V" REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*

LINK EDIT OF 'LNKEDT' COMPLETED
DATE - 88/07/06  TIME - 13.47
ERRORS ENCOUNTERED- 0000  UPSI- X'00'
```

Figure 8-6. Link-Edit Example 5 (Part 14 of 14)

Section 9
Initialize Disk Routine (DSKPRP)

9.1. Introduction

Before using any disk or diskette, you must initialize it, a process commonly called prepping. You prep a disk or diskette when you evaluate the condition of the recording surfaces of your disks/diskettes and prepare them for handling the programs and data that you will later write to them. Prepping can consist of an in-depth evaluation and preparation of your disks and diskettes, a partial analysis of only a portion of your disks and diskettes, or an operation as simple as changing a volume serial number.

Note: *You cannot execute disk prepping to do a partial prep on an active SYSRES.*

Whatever the prepping function, you do it by using the Unisys supplied prep routine called DSKPRP. The disk and diskette types currently supported by System 80 and prepped by DSKPRP are:

Disk/Diskette Type	Removable/ Nonremovable	System 80 Models
8416 disk	Removable	Models 8/10/15/20
8417 disk	Nonremovable	All models
8418 disk	Removable	Models 8/10/15/20
8419 disk	Removable	All models
8430 disk	Removable	Models 8/10/15/20
8433 disk	Removable	Models 8/10/15/20
8470 disk	Nonremovable	Models 4/6/8/10/15/20
8494 disk	Nonremovable	Models 8/10/15/20
8420 diskette	Removable	All models
8422 diskette	Removable	All models

In addition to the DSKPRP routine, there is a standalone prep program (SU@PRP) which supports 8494/8470/8417 disks on the models 8 through 20 and 8470/8417 disks on the models 3 through 6. SU@PRP is used during system installation for disk formatting with defect skipping and surface analysis. SU@PRP creates a volume label for the prepped disk. Other prep facilities are available to you in the form of canned job control streams. These control streams allow you to change volume serial numbers and to prep and allocate system-resident (SYSRES) files. The prep canned job control streams are explained in 9.10.

9.2. Prepping Your Disk

All new disks must be initially prepped before being used in your system. The only disk excluded from this requirement is the 8417 nonremovable disk. A surface analysis is performed as part of the manufacturing process for this disk and it is shipped preprepped to your installation. The only time you need perform an initial prep on an 8417 disk is when you encounter disk problems.

The prepping procedure for all disks in your system is basically the same. It comprises two functions. The first is a track analysis in which each recording surface on the disk is checked for defective tracks. The second is a label initializing function where you build the initial records that must be written on the tracks of your disk before you can write data or programs to them. These initial records are the standard volume label (VOL1) and the volume table of contents (VTOC).

The only difference in the prep procedures performed by DSKPRP on 8416, 8418, 8419, 8430, 8433, and 8494 disks and on 8470/8417 disks is the action taken when a defective area is found. When prepping the 8416, 8418, 8419, 8430, 8433, and 8494 disks, DSKPRP assigns alternate tracks/sectors for areas found to be defective. The defective tracks/sectors are then listed in a track condition table (TCT) or, for the 8494, sector condition table (SCT).

The 8494 has a physical sector size of 512 bytes. Each sector is addressable as two 256-byte logical records. There are fifty 512-byte tracks/sectors and 10 tracks/cylinders. (OS/3 imposes a logical view of the 8494 of ninety-six 256-byte tracks/records and 20 tracks/cylinders.)

Alternate sectors on the 8494 are assigned on a sector basis (as opposed to a track basis, which is done for other devices). Each cylinder has five 512-byte alternate sectors, which are arranged to minimize the overhead associated with an I/O operation that spans a cylinder boundary. The alternate sectors reside on the last physical track (head 9) of even-numbered cylinders, and on the first physical track (head 0) of odd-numbered cylinders. In addition, alternate sectors are assigned so as to minimize the overhead associated with accessing an alternate sector. (The closest available alternate sector is assigned.) If a record is defective, an odd/even record pair (sector) is assigned to an alternate sector. The disk prep output listing always reflects the odd record number for a defective sector in an alternate assignment.

When prepping the 8417 and 8470 disks, DSKPRP first attempts to reformat the sectors on a track that has a defect. That is, it positions the defective area in the gap between sectors. If, however, several defects exist on a track and the track cannot be reformatted, DSKPRP declares the entire track defective and assigns an alternate track for the defective one. A record of the surface analysis, listing the defective and alternate track assignments, is then written to a table similar to the TCT. If the disk being prepped is an 8417 disk, then a second table called the defect skip table (DST) is also prepared. It lists the locations of all defects found throughout the disk. Both the TCT and the DST are explained in 9.3.10.

The surface analysis performed on the fixed-head area of an 8417 fixed-head disk is the same analysis performed on the disk itself. If, however, a defective track in the fixed-head area cannot be reformatted and all the alternates are already assigned, the bad track is then deallocated. A track is also deallocated if the head that accesses it is bad. All deallocated tracks are listed in the track deallocation table (TDT) that is printed between the TCT and DST.

In addition to using the track analyzing and label initializing functions of the disk prep, you can perform the following operations:

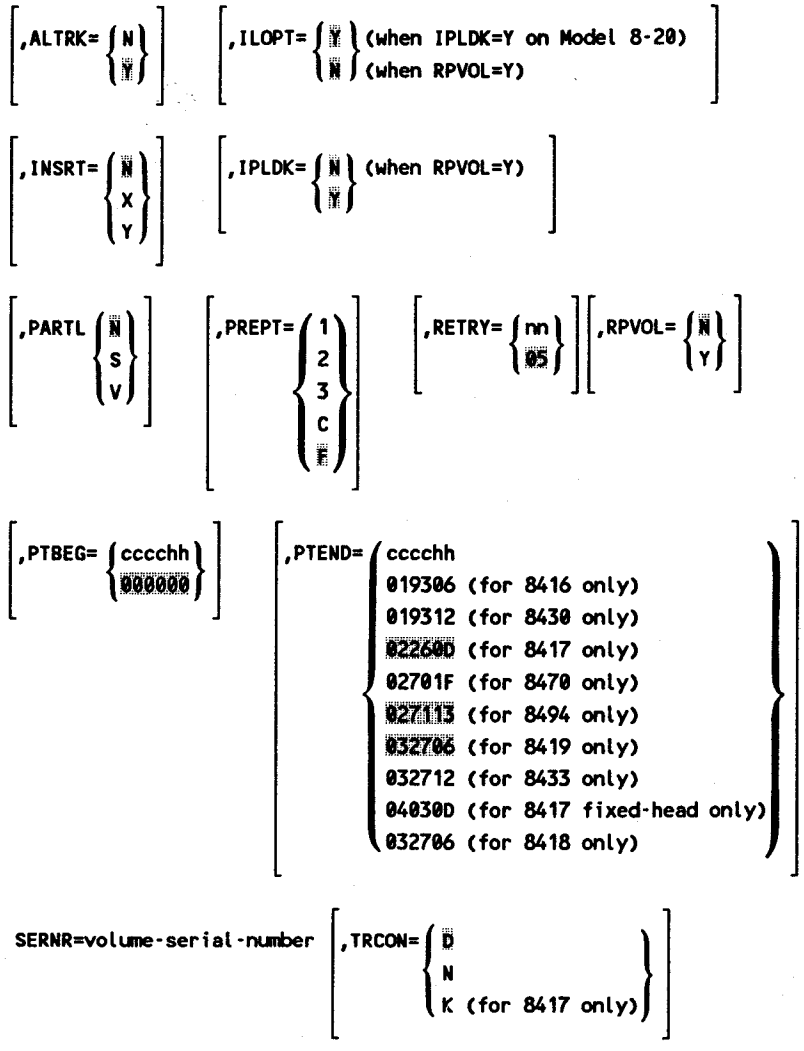
- Prepping a specific track or a group of tracks
- Prepping a specific track or a group of tracks and verifying that a file does not reside on the track(s)
- Choosing options to specify prepping only, fast surface checking (fast prep), or an extremely accurate surface analysis
- Replacing the volume serial number only
- Adding both IMPL and IPL
- Replacing IMPL only
- Replacing IPL only
- Creating a new volume serial number and a VTOC without performing a surface analysis
- Indicating that the disk is to be used as an IPL volume
- Checking the expiration date for all files

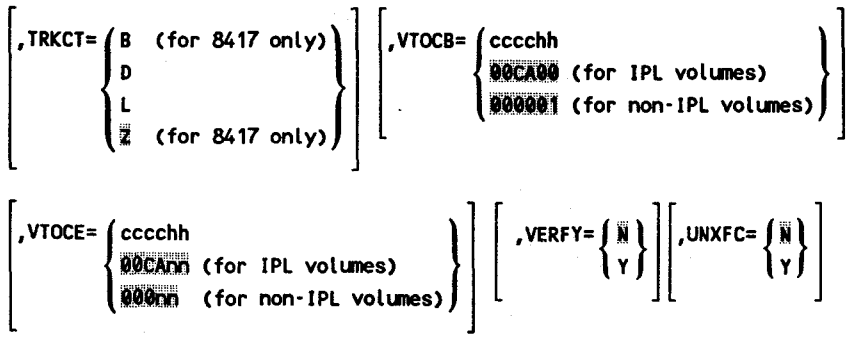
Note: *Before running a disk prep with a supervisor that is not configured to the specific system hardware, make certain that the drive for the disk being prepped is turned on. This reconfigures the information in the PUB. Otherwise, the job uses the current PUB information to determine disk type and features that may not be representative of the disk being prepped.*

9.3. Specifying Prep Options for a Disk

You choose the prep options by selecting one or more of the following keywords. The only keyword you must select is the SERNR keyword identifying the volume serial number of your disk. As many keywords as can be contained on one card are permitted (through card column 71) and as many cards as needed can be present in your control stream. Remember also, that keywords may be specified in any order.

The format of the keywords is





9.3.1. Testing Alternate Track Areas (ALTRK)

As stated earlier, one prep function is performing a surface analysis of your disk. During this process, any defective tracks are flagged and alternate tracks are assigned. When you use ALTRK=Y or omit the ALTRK keyword, the alternate track areas of the disk are tested to ensure that all tracks are usable as alternate tracks.

You can suppress the testing of the alternate tracks by using ALTRK=N.

Note: *Alternate track testing is not supported for 8494 disks. ALTRK=N is always assumed.*

9.3.2. Prepping Your Disk as an IPL Volume (ILOPT and IPLDK)

System 80 Model 8 through 20 Users

You are required to write both the initial microprogram load (IMPL) and the initial program load (IPL) to your disk when prepping it as an IPL volume. You cannot write one without the other unless you are replacing the IMPL or IPL currently existing on your disk. (See 9.3.3 for instructions on replacing an existing IMPL and IPL.)

To write the IMPL and IPL modules to your disk, simply omit both the ILOPT and IPLDK keywords from your prep job stream. The values for both of these keywords default to Y. The prep routine, when executed, preps your disks as an IPL volume and writes both the IMPL and IPL modules to your disk.

If you do not want to prep your disk as an IPL volume, specify the keywords ILOPT=N and IPLDK=N in your prep job stream.

Disk prep for 8430, 8433, and 8494 disks does not build an IMPL file because the 5039 and 5074 disk controllers are not loadable units.

System 80 Model 3 through 6 Users

You are required to write only the IPL to your disk when prepping it as an IPL volume. You are not required to write the IMPL.

To write the IPL module to your disk, simply omit the IPLDK keyword from your prep job stream. The value for this keyword automatically defaults to Y. When IPLDK=Y (specified of defaulted), the prep routine recognizes your disk as an IPL volume and writes the IPL module to it.

If you do not want to prep your disk as an IPL volume, specify IPLDK=N in your prep job stream.

You have the option to write the IMPL to your disk as part of the prep routine. If you elect to use this option, you must write both the IMPL and the IPL. You cannot write IMPL without also writing IPL to your disk. To write both the IMPL and the IPL, specify the keyword ILOPT=Y in your prep job stream and omit the keyword IPLDK (the value for IPLDK automatically defaults to Y). The prep routine, when executed, will write both the IMPL and the IPL modules to your disk.

Considerations for Users of All Models

Use the following device assignment when writing IMPL to your disk:

```
// DVC RES
// LBL $Y$SDF
// LFD $Y$SDF
```

The number of cylinders (starting from cylinder 0) reserved for IMPL and IPL on the various disks supported on System 80 are as follows:

System 80 Model	Disk Type							
	8416	8417	8418	8419	8430	8433	8470	8494
3-6	-	6	-	12	-	-	-	-
8-20	6	3	6	6	2	2	2	2

9.3.3. Replacing an Existing IMPL and/or IPL on Your Disk (ILOPT, IPLDK, and RPVOL)

You can use DSKPRP to replace either the IMPL or the IPL currently existing on your disk.

To replace an existing IMPL, specify the keywords ILOPT=Y and RPVOL=Y in your prep job stream. You must also use the following device assignment:

```
// DVC RES
// LBL $Y$SDF
// LFD $Y$SDF
```

To replace an existing IPL on your disk, specify the keywords IPLDK=Y and RPVOL=Y in your prep job stream.

In both operations, keyword RPVOL=Y allows you to make the specified replacement without changing any of the other information existing on your disk pack.

Note: The canned job stream, PRPMIC, replaces both IPL and IMPL contents (see 9.10).

9.3.4. Recording Defective Tracks (INSRT)

The INSRT keyword indicates to the prep routine whether INSERT control statements are present in your control stream. (INSERT control statements specify, hexadecimally, the addresses of defective areas known to exist on your disk pack. See 9.4 for additional information on the use of the INSERT control statement.)

Specify INSRT=X whenever you use only INSERT control statements in your control stream to identify the defective areas on your disk pack. Specify INSRT=Y whenever you use both INSERT control statements and the normal surface analysis function to flag defective areas. The Y option is automatically set whenever the prep keyword TRCON=N is included in your control stream.

To indicate to the prep routine that your control stream does not contain INSERT control statements, specify INSRT=N or simply omit it and accept the default value N for this keyword.

9.3.5. Changing the Volume Serial Number (RPVOL)

Whenever you use the RPVOL keyword, all other keywords except ILOPT, IPLDK, and SERNR are ignored. (These other keywords are syntax checked.) As with any prep operation, the VOL1 control statement must be specified.

To change the volume serial number of a disk, use RPVOL=Y and supply the new volume serial number with the SERNR keyword. If you are changing the volume serial number in a multivolume environment, then you cannot use the VCHECK parameter in the // LBL job control statement for volume checking.

The volume check function checks the sequence between the volume serial number and the file serial number. The volume serial number and file serial number are part of the VTOC. The RPVOL keyword does not change the file serial number; therefore, in multivolume files, data management does not know the sequence of volumes being processed.

ILOPT and IPLDK default to N when RPVOL=Y.

One point of caution: If you are changing the volume serial number of a SYSRES volume, SYSRUN volume, or a spooling disk, be sure you don't specify a previously assigned volume serial number. Such duplication may not result in a system duplication warning message, but it will severely impact your system. If you inadvertently assign a duplicate volume serial number, you must reinitialize (re-IPL) your entire system.

If you want to change the volume serial number without prepping your disk, use the canned job control streams described in 9.10.1.

Do not attempt to change the volume serial number and replace the IMPL or IPL at the same time. You must do these as separate operations. (See 9.3.3. for instructions on how to replace an IMPL and an IPL that currently exist on your disk.)

To replace the user address specified for a volume, use RPVOL=Y and include the new user address on the VOL1 card. If you do not wish to change the user address, you must leave that field blank on the VOL1 card.

9.3.6. Specifying a Partial Prep or Building a New VOL1/VTOC (PARTL)

When you only want to prep a portion of your disk, specify the PARTL=S keyword. Before you perform a partial prep, however, it is a good idea to make sure that the area being prepped is free of data. You do this by including the VERFY=Y keyword in your prep control stream (see 9.3.12).

When you use PARTL=S, you must use the keywords PTBEG and PTEND to indicate the area of the disk being prepped. TRCON=N must not be specified, since a partial prep does not include generating a new track/sector condition table. The testing of alternate tracks is automatically suppressed via the keyword ALTRK=N. You may specify the INSRT and PREPT keywords; however, the VTOCB, VTOCE, ILOPT, and IPLDK keywords are ignored if they are present in your control stream. Although ignored, these keywords are syntax checked. As with any prep, the keyword SERNR, which indicates the volume serial number, and the VOL1 card, which creates the standard VOL1 labels, must be specified.

When performing a partial prep, DSKPRP destroys any and all information residing in the area being prepped via the PTBEG and PTEND keywords. This can be any area on the disk including the volume serial number and the VTOC. All areas outside the prep area remain unchanged.

You can build a new VOL1 and VTOC without performing a surface analysis of your disk by using the PARTL=V keyword. You can specify this option only if your disk is already prepped and contains a valid VOL1. The new volume serial number specified by the SERNR keyword is written into the VOL1 label on the disk. In addition, all entries in the existing VTOC are removed and a new VTOC constructed. Removal of the existing VTOC entries makes your disk unusable unless you know the specific address of your files. This addressing could have been done by using the ADDR parameter in the EXT job control statement when you allocated your files. If you want to change your volume serial number and still have a usable disk, and you did not use the ADDR parameter in the // EXT job control statement, you must use the RPVOL keyword.

Note: *If PARTL=S is specified for the fixed-head area of an 8417 disk and deallocation must be performed, the prep is terminated. In this case, the entire disk must be prepped.*

9.3.7. Specifying the Extent (Depth) of Prep (PREPT and RETRY)

The PREPT keyword allows you to specify the extent to which you want DSKPRP to conduct a surface analysis. You can choose from the following options:

- PREPT=F — Fast prep - read-only (minimum analysis).
- PREPT=1 — One prep pattern is written and verified.
- PREPT=2 — Two prep patterns are written and verified.
- PREPT={3 / C} — Three prep patterns are written and verified (most in-depth analysis).

The F option is the fastest, but least accurate, of all the preps performed by DSKPRP. It instructs DSKPRP to conduct only a minimum surface analysis (read-only) and requires the least amount of time to complete. It is useful for performing subsequent preps on disk packs that are factory prepped or disks that have had in-depth preps and are not experiencing problems during processing.

The remaining options (1, 2, 3, and C) offer an in-depth surface analysis with option C or 3 being the most complete and most accurate. These options instruct DSKPRP to write and verify prep patterns to the disk. The more patterns written and verified, the more accurate the surface analysis. Of course, the more patterns you write, the more time it takes to perform the prep. However, if you are experiencing disk problems, such as unrecoverable data errors during processing, you should perform a critical surface analysis of that disk.

Regardless of the type of prep you request, DSKPRP tests a given track up to the number of times you specified on the RETRY keyword in an attempt to have it pass the surface analysis test. That is, if an area is found to be defective the first time it is tested, it is retested up to the number of times specified by the RETRY keyword before it is declared bad, and an alternate track/sector is substituted for it. Thus, lowering

the RETRY specification has the effect of making the disk prep routine perform a more critical surface analysis by giving each track/sector fewer chances to pass the surface analysis test. Conversely, increasing the RETRY specification lowers the effectiveness of the surface analysis test by giving each track/sector more chances to pass the test.

The number of retries is specified as a hexadecimal number from 00 to FF. The default number of retries is 5.

9.3.8. Specifying Where Prepping Starts and Ends (PTBEG and PTEND)

By omitting both the PTBEG and PTEND keywords, prepping starts at the primary track address of 000000 and ends at the highest primary track address of your disk.

If you want prepping to start at some place other than the beginning of the disk and end some place other than the highest primary track address, you specify these addresses (hexadecimal) in cylinder/head format *(cccchh)*.

The use of PTBEG and PTEND does not prevent the initialization of the VOL1 record, the IMPL/IPL, or the VTOC. To suppress the initialization of these records, you must specify the VERFY=Y or PARTL=S parameter.

9.3.9. Specifying Your Volume Serial Number (SERNR)

The volume serial number is six alphanumeric characters long, which make up the serial number of the disk volume being prepped, and may not contain any blank characters. Your volume serial number may already have been assigned to the disk volume through a previous prep; or it may specify a new serial number. In either case, the SERNR keyword must always be present in your disk prep control stream. If a new serial number is being specified on a previously prepped disk, you should specify either the RPVOL keyword or the NOV option on your VOL statement.

9.3.10. Specifying a Track Condition Table (TRCON, TRKCT, and FRMTG)

The track condition table (TCT) or sector condition table (SCT) tells you the general condition of your disk and provides subsequent preps with a history of the alternate track/sector assignments. Figure 9-1 is a sample of a printout generated for an 8417 disk prep. Figure 9-2 is a sample of a printout generated for an 8494 disk prep.

The track condition table in Figure 9-1 tells you that there are three defective tracks (designated by a 1 in code byte 1) on the disk. The number 5 in code byte 1 for these tracks indicates that each defective track was not found by the surface analysis, but was designated by card inserts in the prep job control stream.

```
VER= 880604          ********** OS/3 CISC INITIALIZATION **********

                     DATE  88/06/10                          TIME  19:04:33

* CONTROL STREAM PARAMETERS *

SERNR=XXXXXX
TRCON=N,INSRT=Y
VOL1


* DEFAULT PARAMETERS *

          ALTRK=Y        ILOPT=N        IPLCK=Y        PARTL=N        PREPT=F
          PTBEG=000000   PTEND=022E0D   RETRY=0A       RPVOL=N        TRKCT=7
          VERFY=N        VTOCB=000A0D   VTOCE=000A0D


* INSERT PARAMETERS *

INSERT 004302
INSERT                   004403,004506

                              OS/3  TRACK  CONDITION  TABLE

                                  SERIAL NUMBER XXXXXX

                                                                  BYTES   FIRST
                                                                  WRITTEN BAD
ALTERNATE  TRACK  CODE BYTE1 CODE BYTE2  PRIMARY   TRACK         ON BAD  BYTE    BYTE    BYTE
CYLINDER   HEAD   0123 4567  0123 4567   CYLINDER  HEAD          TRACK   FOUND   READ    EXPECTED
-------------------------------------------------------------------------------------------------
  0226      00     1 3  5                 0043      02            0000    0000    00      00
  0226      01     1 3  5                 0044      03            0000    0000    00      00
  0226      02     1 3  5                 0045      06            0000    0000    00      00
  0226      03       3   6                0000      00            0000    0000    00      00
  0226      04      2                     0000      00            0000    0000    00      00
  0226      05      2                     0000      00            0000    0000    00      00
  0226      06      2                     0000      00            0000    0000    00      00
  0226      07      2                     0000      00            0000    0000    00      00
  0226      08      2                     0000      00            0000    0000    00      00
  0226      09      2                     0000      00            0000    0000    00      00
  0226      0A      2                     0000      00            0000    0000    00      00
  0226      0B      2                     0000      00            0000    0000    00      00
  0226      0C      2                     0000      00            0000    0000    00      00
  0226      0D      2                     0000      00            0000    0000    00      00
  0227      00      2                     0000      00            0000    0000    00      00
  0227      01      2                     0000      00            0000    0000    00      00
  0227      02      2                     0000      00            0000    0000    00      00
  0227      03      2                     0000      00            0000    0000    00      00
  0227      04      2                     0000      00            0000    0000    00      00
  0227      05      2                     0000      00            0000    0000    00      00
  0227      06      2                     0000      00            0000    0000    00      00
  0227      07      2                     0000      00            0000    0000    00      00
  0227      08      2                     0000      00            0000    0000    00      00
  0227      09      2                     0000      00            0000    0000    00      00
  0227      0A      2                     0000      00            0000    0000    00      00
  0227      0B      2                     0000      00            0000    0000    00      00
  0227      0C      2                     0000      00            0000    0000    00      00
  0227      0D      2                     0000      00            0000    0000    00      00
  0228      00      2                     0000      00            0000    0000    00      00
  0228      01      2                     0000      00            0000    0000    00      00
  0228      02      2                     0000      00            0000    0000    00      00
  0228      03      2                     0000      00            0000    0000    00      00
  0228      04      2                     0000      00            0000    0000    00      00
  0228      05      2                     0000      00            0000    0000    00      00
  0228      06      2                     0000      00            0000    0000    00      00
  0228      07      2                     0000      00            0000    0000    00      00
  0228      08      2                     0000      00            0000    0000    00      00
  0228      09      2                     0000      00            0000    0000    00      00
  0228      0A      2                     0000      00            0000    0000    00      00
  0228      0B      2                     0000      00            0000    0000    00      00
  0228      0C      2                     0000      00            0000    0000    00      00
```

Figure 9-1. Typical Disk Prep Printout (Part 1 of 3)

0228	00	2	0000	00	0000	0000	00	00
0229	00	2	0000	00	0000	0000	00	00
0229	01	2	0000	00	0000	0000	00	00
0229	02	2	0000	00	0000	0000	00	00
0229	03	2	0000	00	0000	0000	00	00
0229	04	2	0000	00	0000	0000	00	00
0229	05	2	0000	00	0000	0000	00	00
0229	06	2	0000	00	0000	0000	00	00
0229	07	2	0000	00	0000	0000	00	00
0229	08	2	0000	00	0000	0000	00	00
0229	09	2	0000	00	0000	0000	00	00
0229	0A	2	0000	00	0000	0000	00	00
0229	0B	2	0000	00	0000	0000	00	00
0229	0C	2	0000	00	0000	0000	00	00
0229	0D	2	0000	00	0000	0000	00	00
022A	00	2	0000	00	0000	0000	00	00
022A	01	2	0000	00	0000	0000	00	00
022A	02	2	0000	00	0000	0000	00	00
022A	03	2	0000	00	0000	0000	00	00
022A	04	2	0000	00	0000	0000	00	00
022A	05	2	0000	00	0000	0000	00	00
022A	06	2	0000	00	0000	0000	00	00
022A	07	2	0000	00	0000	0000	00	00
022A	08	2	0000	00	0000	0000	00	00
022A	09	2	0000	00	0000	0000	00	00
022A	0A	2	0000	00	0000	0000	00	00
022A	0B	2	0000	00	0000	0000	00	00
022A	0C	2	0000	00	0000	0000	00	00
022A	0D	2	0000	00	0000	0000	00	00
022B	00	2	0000	00	0000	0000	00	00
022B	01	2	0000	00	0000	0000	00	00
022B	02	2	0000	00	0000	0000	00	00
022B	03	2	0000	00	0000	0000	00	00
022B	04	2	0000	00	0000	0000	00	00
022B	0F	2	0000	00	0000	0000	00	00
022C	06	2	0000	00	0000	0000	00	00
022C	07	2	0000	00	0000	0000	00	00
022C	0F	2	0000	00	0000	0000	00	00
022C	09	2	0000	00	0000	0000	00	00
022C	0A	2	0000	00	0000	0000	00	00
022C	0B	2	0000	00	0000	0000	00	00
022C	0C	2	0000	00	0000	0000	00	00
022C	0D	2	0000	00	0000	0000	00	00
022D	00	2	0000	00	0000	0000	00	00
022D	01	2	0000	00	0000	0000	00	00
022D	02	2	0000	00	0000	0000	00	00
022D	03	2	0000	00	0000	0000	00	00
022D	04	2	0000	00	0000	0000	00	00
022D	05	2	0000	00	0000	0000	00	00
022D	06	2	0000	00	0000	0000	00	00
022D	07	2	0000	00	0000	0000	00	00
022D	08	2	0000	00	0000	0000	00	00
022D	09	2	0000	00	0000	0000	00	00
022D	0A	2	0000	00	0000	0000	00	00
022D	0B	2	0000	00	0000	0000	00	00
022D	0C	2	0000	00	0000	0000	00	00
022D	0D	2	0000	00	0000	0000	00	00
022E	00	2	0000	00	0000	0000	00	00
022E	01	2	0000	00	0000	0000	00	00
022E	02	2	0000	00	0000	0000	00	00
022E	03	2	0000	00	0000	0000	00	00
022E	04	2	0000	00	0000	0000	00	00

Figure 9-1. Typical Disk Prep Printout (Part 2 of 3)

```
             0226      05      2                          0000     00      0000     0000    00     00
             0226      06      2                          0000     00      0000     0000    00     00
             0226      07      2                          0000     00      0000     0000    00     00
             0226      08      2                          0000     00      0000     0000    00     00
             0226      09      2                          0000     00      0000     0000    00     00
             0226      0A      2                          0000     00      0000     0000    00     00
             0226      0B      2                          0000     00      0000     0000    00     00
             0226      0C      2                          0000     00      0000     0000    00     00
             0226      0D      2                          0000     00      0000     0000    00     00

CODE BYTE1
   BIT 0= ALTERNATE TRACK LISTED IS DEFECTIVE
       1= PRIMARY TRACK LISTED IS DEFECTIVE AND AN ALTERNATE HAS BEEN ASSIGNED
       2= ALTERNATE TRACK LISTED IS GOOD, BUT NOT ASSIGNED
       3= ALTERNATE TRACK LISTED IS GOOD AND HAS BEEN ASSIGNED BY THIS RUN
       4= THIS DEFECTIVE TRACK FOUND BY SURFACE ANALYSIS
       5= THIS DEFECTIVE TRACK DESIGNATED BY CARD INSERT
       6= THIS ALT TRACK CONTAINS THE TRACK CONDTN TABLE
       7= THIS ALT TRACK WAS ASSIGNED BY A PREVIOUS RUN


                                   O S / 3   D E F E C T   T A B L E

        SERIAL NUMBER EXAMPL                                                 LAST UPDATED 80/10/21

   D E F E C T            D E F E C T            D E F E C T            D E F E C T            D E F E C T
    E N T R Y              E N T R Y              E N T R Y              E N T R Y              E N T R Y
FFCCCCHH DDDDDDLL      FFCCCCHH DDDDDDLL      FFCCCCHH DDDDDDLL      FFCCCCHH DDDDDDLL      FFCCCCHH DDDDDDLL
----------------------------------------------------------------------------------------------------------

00000307 001611AC      00000307 0016F1A0      00006800 00781810      0000310A 00003810      0000820A 0009621C
0000830A 0009481C      0000840A 0009501C      0000960A 0009681C      00010507 0109601C      00012905 0001481C
00016508 0006181C      00010500 00F70810      00010600 00F60810      00010700 00FCA010      0001E403 01708010
00020A09 0190C81C      00022809 0258F810      0001700C 017AA010      00007F0A 00095010      0001790C 0042101C
00010002 01AF001C      00022E08 0095E81C
* DEFECT TABLE END *

USAK4 IPL  HAS BEEN LOADED SUCCESSFULLY
USAR1 NORMAL EOJ - DISK IS GOOD


UNISYS SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE-  88/06/10     TIME- 17:12:04    UPSI- X'30'
VSN-   EXAMPL       TYPE- 8417
```

Figure 9-1. Typical Disk Prep Printout (Part 3 of 3)

When a track is defective, an alternate track is automatically assigned. The tracks that are reserved as alternates are listed in the first two columns of the table. A 3 in code byte 1 indicates that the corresponding alternate track (designated in the first two columns) is to replace the corresponding defective primary track. Also, the code number 6 shows that the alternate track is used to store the track condition table.

So, in this example, track 0 of cylinder 226 is assigned as an alternate to replace primary track 2 of cylinder 43. The alternate track for defective primary track 3 of cylinder 44 is track 1 of cylinder 226. And, finally, track 6 of cylinder 45 is replaced by the alternate track 2 of cylinder 226.

All other tracks are listed as good (code 2) and are unassigned.

When prepping an 8417 disk, DSKPRP prints out a defect skip table (DST) in addition to a track condition table (TCT). The DST shown in part 3 of Figure 9-1, lists the locations of track defects. It is divided into five defect entry columns. The first half of each one shows the cylinder and head address of the track containing the defect. The second half helps you to pinpoint the defect's location on the track and shows how much space was skipped to avoid the defect. The D digits indicate how far, in bits, the defect is from the track's physical index marker. The L digits indicate the bit length of the area that was skipped to avoid the defect.

It is important that you keep both a current diskette copy of this DST and the listing handy. Should serious problems occur with your disk, your Unisys representative needs this information to remedy any problems.

There are two keywords associated with the processing of the TCT/SCT and, when prepping an 8417 disk, the DST. They are TRCON and TRKCT. TRCON instructs the DSKPRP routine on how to process the table(s) and TRKCT indicates where to place the results.

The following options are available when you specify TRCON:

- You use TRCON=D to indicate the TCT/SCT is input from disk. You use this *default* option when your disk has been previously prepped (in use) and the current track/sector assignments are to be retained. You should always use this default option unless the existing TCT/SCT is unusable.

- You use TRCON=N to indicate that a new TCT/SCT be generated. You must use this option when prepping a disk for the first time or if you cannot recover the existing TCT/SCT from the disk by using the TRCON=D option. The disk is formatted only when TRCON=N is specified.

  For 8470 disks, specify FRMTG=D also to rewrite the home address if errors are occurring on the high heads of the cylinders. Both TRCON=N and FRMTG=D must be specified to rewrite home addresses on the 8470.

- You use TRCON=K when prepping an 8417 disk. It indicates that the TCT and the DST are input from diskette.

When you specify TRKCT, the following options are available:

- You use TRKCT=D to indicate that your TCT/SCT is written to an available alternate track or, for an 8494 disk, to a reserved defect cylinder.

- You use TRKCT=Z to indicate that your TCT/SCT is written to an available alternate track or, for an 8494 disk, to a reserved defect cylinder, and listed on the printer. This option is the default for TRKCT.

- You use TRKCT=L to indicate that your TCT/SCT is listed on the printer only. No other prepping functions are performed. Your disk pack must have been previously prepped to use this option. It is useful for future references about your disk pack's track condition.

- You use TRKCT=K only when prepping 8417 disk packs. It indicates that your TCT and DST are written to disk and diskette and also listed on the printer. It is recommended that you use this option for backing up your DST.

- You use TRKCT=B only for 8417 disk packs. It indicates that your TCT and DST are written only to diskette and listed on the printer. No other prepping functions are performed. This option is useful when you want to back up your DST without prepping your disk pack. You can only use this option if your disk pack has been previously prepped.

Notes:

1. *All TRKCT options that update the TCT also write the TCT to an available alternate track so subsequent preps may use previous surface analysis results and INSERT specifications.*

2. *Processing the DST on a diskette requires a device assignment for the diskette. The LFD name required in the device assignment set is / / LFD DSKET.*

The alternate assignment table that is printed for the 8494 disk is called the sector condition table (SCT) (see Figure 9-2). The SCT is maintained on a cylinder that is reserved for storing defect information. There are significant differences between the SCT format and the TCT format.

```
                        OS/3 SECTOR CONDITION TABLE

                              SERIAL NUMBER UNISYS

   PHYSICAL                                DEFECTIVE PHYSICAL      DEFECTIVE LOGICAL
ALTERNATE ADDRESS      CODE BYTE          PRIMARY ADDRESS         PRIMARY ADDRESS
cylinder head sector   0123 4567    cylinder head sector    cylinder head sector
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  005A    09    2D      123           005A    05    2B        002E    0E    17
  00DF    00    14      123           00DF    04    0F        0073    03    3F
  01F2    09    2D    01   4
  01F2    09    2E      12           01F2    05    08        0101    01    01
  0494    09    2D      123           0494    06    1C        025C    0D    09
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  CODE BYTE DEFINITION:
     BIT 0 - SECTOR IS DEFECTIVE
     BIT 1 - SECTOR IS AN ALTERNATE
     BIT 2 - ALTERNATE SECTOR IS ASSIGNED
     BIT 3 - DEFECTIVE ENTRY FOUND BY THE FACTORY
     BIT 4 - SECTOR I.D. IS DISPLACED
     BIT 5 - SECTOR I.D. IS EXTENDED DISPLACED
```

Figure 9-2. Sample Sector Condition Table

The PHYSICAL ALTERNATE ADDRESS indicates the address of the alternate sector. The CODE BYTE provides information about the alternate sector. The DEFECTIVE PHYSICAL PRIMARY ADDRESS indicates the physical address (assuming fifty 512-byte sectors per track and 10 tracks per cylinder) of the defective sector. The DEFECTIVE LOGICAL PRIMARY ADDRESS indicates the logical address (assuming ninety-six 256-byte records and 20 tracks per cylinder) of the defective sector. The record number represents an odd/even pair of 256-byte records. OS/3 uses these logical addresses to access the disk. If an alternate sector is defective, there are no defective physical or logical primary address entries.

The sample SCT shown in Figure 9-2 has five alternate sector entries. One of the alternates is defective with a displaced I.D. field (01F2/09/2D). The other four alternates are assigned to defective primary sectors.

9.3.11. Specifying the VTOC Address (VTOCB and VTOCE)

The VTOC is a directory that contains the addresses of the files contained in your volume. You can indicate where you want the VTOC to reside by specifying six hexadecimal numbers representing the starting primary track address in cylinder/head format (*ccccchh*) on the VTOCB keyword. The starting address must be less than the end VTOC address and must be at least one track in length; however, we recommend that one cylinder be allocated for the VTOC. By omitting the VTOCB keyword, DSKPRP automatically assigns the starting VTOC address for you. Table 9-1 lists the default VTOC starting addresses for both IPL and non-IPL disk volumes.

Table 9-1. Default Starting VTOC Addresses for Disk

Disk	Non-IPL Volume	IPL Volume
8416	000001	00CA00
8417	000001	00CA00
8418	000001	00CA00
8419	000001	00CA00
8430	000001	00CA00
8433	000001	00CA00
8470	000001	00CA00
8494	000001	00CA00

After you have indicated the starting address of your VTOC, the next step is specifying the ending address. You can specify the hexadecimal numbers to indicate the cylinder and head address of the ending track on the VTOCE keyword. By omitting the VTOCE keyword, DSKPRP automatically assigns the ending VTOC address for you. Table 9-2 lists the VTOC ending addresses for both IPL and non-IPL disk volumes.

Table 9-2. Default Ending VTOC Addresses for Disk

Disk	Non-IPL Volume	IPL Volume
8416	000006	00CA06
8417	00000D	00CA0D
8418	000006	00CA06
8419	000006	00CA06
8430	000012	00CA12
8433	000012	00CA12
8470	00001F	00CA1F
8494	000013	00CA13

Notes:

1. The VTOC cannot be placed in the fixed-head area of an 8417 disk.

2. The number of cylinders (starting from cylinder 0) reserved for IMPL and IPL on the various disks supported on System 80 are as follows:

System 80 Model	Disk Type							
	8416	8417	8418	8419	8430	8433	8470	8494
3-6	-	6	-	12	-	-	-	-
8-20	6	3	6	6	2	2	2	2

9.3.12. Testing an Area before Prepping (VERFY)

If you're doing a partial prep, it's a good idea to test the area to be prepped first to make certain that that area is free of data. You use the VERFY=Y keyword to do this testing. Whenever you specify VERFY=Y, the keywords PTBEG, PTEND, and SERNR must also be specified. In addition, TRCON cannot be an N. The keyword ALTRK=N is assumed while the keywords VTOCB, VTOCE, ILOPT, and IPLDK are ignored if they're present in your control stream. They are, however, syntax checked. The INSRT and PREPT keywords may be used, if needed. As with all preps, the VOL1 card must be specified.

By specifying VERFY=N or omitting the keyword, no testing is performed.

Note: *If the area defined by the PTBEG and PTEND keywords is occupied, the prep will automatically stop, and no action will take place. Also, the VOL1 label or the VTOC area is not rebuilt.*

9.3.13. Checking the File Expiration Date (UNXFC)

You use the UNXFC=Y keyword to check the expiration date for all files on your volume. When you use this keyword, you prevent any file from being scratched if the expiration date has not expired.

DSKPRP compares the system date to the expiration date for each file. Whenever the expiration date is greater than the system date, a message is displayed indicating the expiration date has not expired (up to 10 files at a time can be displayed). Associated with this message, DSKPRP gives you the option to ignore the message and continue with the prep or cancel the prep. Normally, you would cancel the prep when the expiration date has not expired.

If you omit the UNXFC keyword or specify UNXFC=N, no expiration date validation is performed.

9.4. Assigning Alternates for Suspected Defective Tracks/Sectors (INSERT)

The INSERT control statement is used to identify any known defective tracks/sectors. Any tracks/sectors that you specify on INSERT control statements are assigned alternates.

You must identify the tracks/sectors on INSERT control statements by their hexadecimal equivalents. You can either specify one track/sector on each INSERT control statement or use one INSERT control statement to specify a number of tracks/sectors, as long as the list of entries does not extend beyond column 71. Whenever INSERT control statements are present in your control stream, you must specify either keyword INSRT=X or keyword INSRT=Y.

A list of any known defective tracks is supplied by the manufacturer for your removable disk packs. This list appears on a label located on the bottom of the plastic cover for your disk pack. The label also lists your pack's volume serial number. The defective tracks are listed both decimally and hexadecimally in cylinder and head format (cccchh). Defective tracks for your 8417 nonremovable disks are found in the OS/3 track condition table in the disk prep printed for that pack. (The 8417 disk packs go through an extensive surface analysis during the manufacturing process.)

The format of the INSERT control statement is

```
1         10
-------------------------------------------------------------
INSERT    { ccccchh[,ccccchh...,ccccchh]                     }
          { ccccchhrr[,ccccchhrr...,ccccchhrr] (8494 only)   }
          { NONE                                             }
```

where:

ccchh
Is the hexadecimal address in cylinder/head format of a possible defective track to be flagged .

ccchhrr
Is the hexadecimal address in cylinder/head/record format of a possible defective record to be flagged. This format applies to 8494 disks only.

NONE
Indicates that there are no known defective tracks.

9.5. Creating Standard Volume Labels (VOL1)

The VOL1 label is the standard volume label in OS/3. It's used to identify your volume by a unique serial number and is also used to locate the address of the VTOC. You must specify a VOL1 statement every time you do a prep.

The format of the VOL1 statement is

```
1       11                              42       51
-------------------------------------------------
VOL1    [r]                             [aaaaaaaaaa]
```

When coding the VOL1 statement, VOL1 must start in column 1. Column 11 is reserved for future use to specify a security byte (*r*). Columns 42 through 51 are used for a name or address of the disk. Normally, the contents of this field is up to the user. Figure 9-3 shows the format of a VOL1 label as it appears on a disk.

The VOL1 label is identified by the word VOL1 in the label identification field, which is the first field in the label. The VOL1 label for disk is always written on cylinder 0, head 0, record 3.

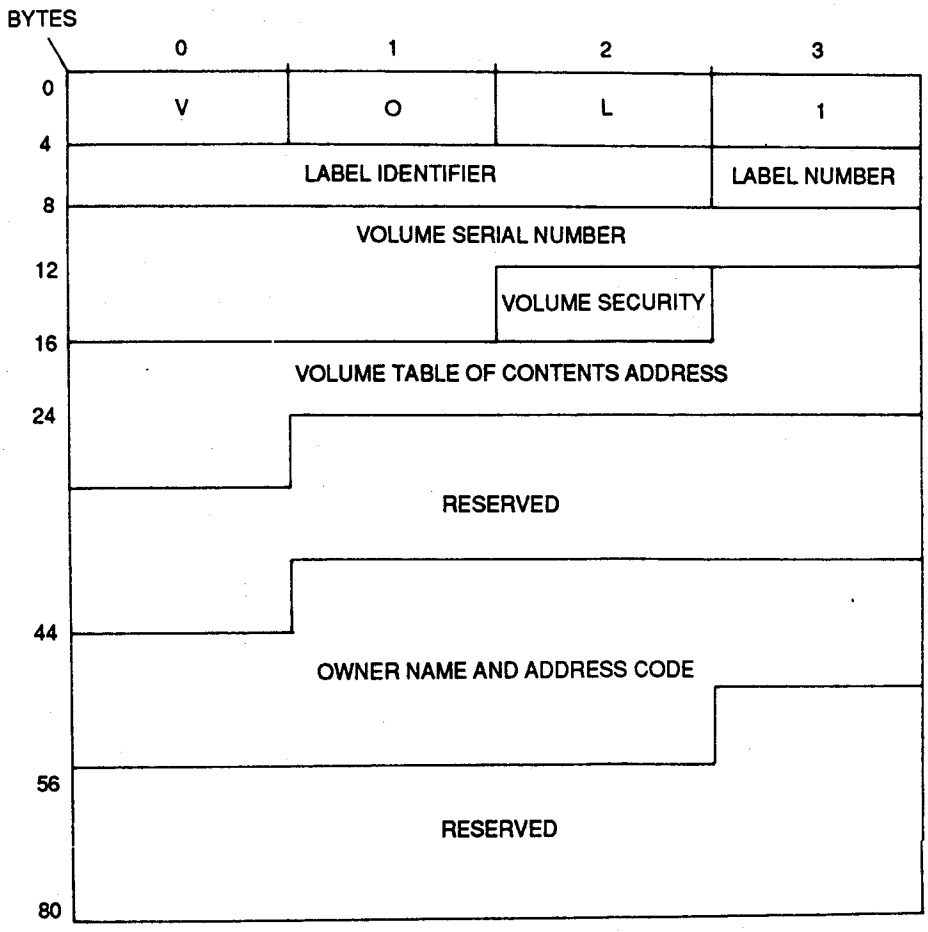


Figure 9-3. VOL1 Format

9.6. Prepping Your Diskette

The same routine for prepping disks (DSKPRP) is also used for prepping diskettes. However, because of certain hardware design differences, the prepping options for diskettes are different from that for disks. When prepping your diskette, you can

- Analyze the surface for defective tracks.
- Change your diskette volume serial number.
- Indicate whether your diskette is to be used as an IPL volume or not.
- Indicate whether your diskette is to be used as an IMPL volume or not.
- Replace either the IPL or IMPL (but not both).
- Format your diskettes in either data set label (DSL) mode or format label (FLB) mode.
- Indicate where you want the volume table of contents (VTOC) to reside on a format label diskette.
- Build a new VOL1 and VTOC or DSLs without performing a surface analysis.

For diskette preps, the DSKPRP options associated with assigning alternate tracks may not be used. Instead, each diskette already contains two tracks reserved as alternate tracks. No alternate tracks can be assigned by the user. If more than two defective tracks are found on a diskette, the diskette must be discarded.

The DSKPRP options associated with partial prepping may not be used with diskettes.

9.7. Specifying Prep Options for a Diskette

Just as when you prep a disk (see 9.3), you choose the prep options for a diskette by selecting one or more of the following keywords. The formats of the keywords are

```
[,RPVOL= {N} ], SERNR=volume-serial-number
         {Y}

[,FORMT= {DSL} ] [,FDATA= {Y} ] [,DNSTY= {1} ] [,RECSZ= {128} ]
         {FLB}           {N}             {2}           {256}
                                                       {512}

[,SPIRL= {Y} ] [,LACEG= {nn} ] ,ILOPT= {Y} [ ,IPLDK= {Y} ][,UNXFC= {Y} ]
         {N}           {01}             {N}          {N}          {N}

[,VTOCB= {cccchh                                                  } ]
         {002400 (for 8420/8422 IPL and non-IPL volumes)}

[,VTOCE= {cccchh                                                  } ]
         {002401 (for 8420/8422 IPL and non-IPL volumes)}

[,PARTL= {V} ]
         {N}

[,IMPNM= {nnnn                          } ]
         {CPU (for model 3-6 systems)   }
         {IDCU (for model 8-20 systems) }
         {IDC                           }
         {IOMP                          }
         {DBUS                          }
         {FDD0 (for model 8 systems)    }

[,CUADR=nnn]
```

Notes:

1. *The SERNR keyword must be specified every time you prep a diskette.*
2. *There is no option to specify single-sided or double-sided diskette. These are physical characteristics of a diskette.*

9.7.1. Changing the Diskette Volume Serial Number or Replacing the IMPL or IPL (RPVOL)

To change the diskette volume serial number, use RPVOL=Y and supply the new volume serial number with the SERNR keyword.

To replace the existing IMPL or IPL without destroying the existing information on your diskette, use ILOPT=Y or IPLDK=Y with RPVOL=Y. You cannot change a volume serial number and replace IMPL or IPL at the same time. Whenever you use the RPVOL keyword, all other keywords except ILOPT, IPLDK, and SERNR are ignored. However, these keywords are syntax checked. As with any prep operation, the VOL1 control statement must be specified.

To change the volume serial number without prepping the diskette, use the canned job control stream called CHGVSN (see 9.10.1).

Note: *Replacing the IMPL requires the following device assignment:*

```
// DVC RES
// LBL $Y$SDF
// LFD $Y$SDF
```

9.7.2. Specifying the Diskette Volume Serial Number (SERNR)

The volume serial number is six alphanumeric characters that make up the serial number of the diskette being prepped. It may not contain any blank characters. Your volume serial number may already have been assigned to the diskette through a previous prep or it may specify a new serial number. In either case, the SERNR keyword must always be present in your prep control stream.

If SERNR is the only keyword used, the prep will automatically perform the following:

1. Reinitialize the data set labels.
2. Change the volume serial number to the value of SERNR.
3. Write a prep pattern and analyze all data records of the diskette, ensuring their availability.

9.7.3. Indicating Diskette Format (FORMT)

A diskette can be prepped in either data set label (DSL) mode, to simulate a sequential file like a tape file, or in format label (FLB) mode, to simulate a disk file.

You specify your choice with the FORMT keyword, as follows:

- DSL indicates that the format is in data set label mode. Your diskette will contain records with either 128-, 256-, or 512-byte record sizes, written in single or double density. You cannot, however, specify a record size of 128 bytes when prepping a double-density diskette.

- FLB indicates that the format is to be in format label mode. Your diskette is limited to a 256-byte, double-density, two-sided surface. A VTOC is generated and a disk-like VOL1 label is written on sector 3 of the index track.

A VOL1 statement is always needed, no matter what format is specified.

9.7.4. Specifying File Allocation for DSL Diskettes (FDATA)

The disk prep routine automatically allocates the entire diskette (FDATA=Y) as being one file and names that file DATA. Therefore, if you are using the diskette as a work file, there is no need to allocate file space (using the // EXT statement). However, if you later decide to use the diskette in a multifile environment or use a different name, you must scratch the file before allocating a new one. You can scratch files using either the PARTL=V keyword (see 9.7.13) or via the SCR job control statement.

You can also specify the diskette to be made available for any file allocation (just as a disk) by specifying FDATA=N. (When FDATA=N, the file DATA is not built.) Once prepped, you then allocate the required file space using the // EXT job control statement with the BLK parameter or the interactive services ALLOCATE command.

Notes:

1. *You can also scratch files interactively via the interactive services ERASE command. See the current version of the* Interactive Services Operating Guide *(UP-9972) for details.*

2. *The FDATA parameter is not applicable with format label diskettes.*

9.7.5. Specifying Diskette Density (DNSTY)

When specifying the density at which the diskette is prepped, you must use the keyword DNSTY=*n*, where *n* is either 1 for single density or 2 for double density.

9.7.6. Specifying Record Size (RECSZ)

The size of the record to be created on the diskette is specified by the keyword RECSZ=*nnn*, where *nnn* is the size of the record. Possible values for this keyword are 128, 256, and 512 bytes. The default parameter is 128 bytes for the DSL mode. However, you cannot specify a record size of 512 bytes when writing IMPL or IPL to the diskette.

When prepping a double-density FLB diskette, you can only specify a record size of 256 bytes.

9.7.7. Spiraling Records to Reduce Track-to-Track Latency Time (SPIRL)

You can minimize track-to-track latency time by specifying the keyword SPIRL=Y. When this keyword is specified, a predetermined number of physical records are skipped when numbering logical records from track to track. This causes the first logical record of each track to appear to spiral in relation to its physical index marker. Therefore, when the read/write head moves from one track to another, the surface rotates the first logical record under the head without the loss of a revolution. As a result, the time needed to access records from track to track is reduced.

9.7.8. Lacing Records to Reduce Latency Time within a Track (LACEG)

You can minimize the latency time within the track itself by specifying the LACEG=*nn* keyword, where *nn* is the number of physical records on a track that are offset between logical records. You can specify a decimal number from 01 through 25, with the default being 01.

When you use the LACEG keyword, each logical record on a track is separated by the number of physical records you specify. By lacing your records properly, you can space your records on the track so that each logical record is rotated under the read/write head just as the I/O order to retrieve it is issued. As a result, you can access a number of records without the loss of a revolution.

9.7.9. Loading Initial Microprogram Load to Diskette (ILOPT)

You write initial microprogram load (IMPL) to diskette by specifying ILOPT=Y. The default for the keyword is ILOPT=N. The following device assignment set is used for your disk-resident volume when writing IMPL to diskette:

```
// DVC RES
// LBL $Y$SDF
// LFD $Y$SDF
```

You may also use the ILOPT keyword to replace IMPL on your diskette. When replacing IMPL, ILOPT=Y must be specified along with RPVOL=Y and the SERNR keyword.

Notes:

1. *Both IPL and IMPL cannot be loaded to the same diskette.*
2. *RECSZ cannot have a value of 512 bytes. RECSZ must be 128 bytes if IMPNM=FDD0. The diskette must be single sided and SERNR=S$IMPL.*

3. *CUADR must be specified on model 8 systems unless IMPNM=FDD0.*

4. *See 9.7.14 and 9.7.15 for additional information concerning IMPL loading.*

9.7.10. Indicating Your Diskette Is an IPL Volume (IPLDK)

The IPLDK keyword indicates whether the diskette being prepped is used as an IPL volume or not. If the IPLDK=Y option is specified, the diskette is indicated as an IPL volume. If the diskette is not to be used as an IPL volume, the IPLDK=N option must be specified or omitted. You replace IPL by specifying IPLDK=Y along with RPVOL=Y and the SERNR keyword. The default is IPLDK=N.

Notes:

1. *Both IPL and IMPL cannot be loaded to the same diskette.*

2. *RECSZ cannot have a value of 512 bytes.*

9.7.11. Checking the File Expiration Data (UNXFC)

You use the UNXFC keyword to check the expiration date for all files on your volume. When you use this keyword, you prevent any file from being scratched if the expiration date has not expired.

DSKPRP compares the system date to the expiration date for each file. Whenever the expiration date is greater than the system date, a message is displayed indicating the expiration date has not expired (up to 10 files at a time can be displayed). Associated with this message, DSKPRP gives you the option to ignore the message and continue with the prep or cancel the prep. Normally, you would cancel the prep when the expiration date has not expired.

If you omit UNXFC keyword, no expiration date validation is performed.

9.7.12. Specifying the VTOC Address (VTOCB and VTOCE)

The VTOC is a directory that contains the addresses of your program library files that are contained in your volume. If you specified the FORMT=FLB keyword in your job control stream, you can indicate where you want the VTOC to reside by specifying six hexadecimal numbers representing the starting primary track address in cylinder/head format (*ccchh*) on the VTOCB keyword. The starting address must be less than the end VTOC address and must be at least one track in length; however, we recommend that one cylinder be allocated for the VTOC. By omitting the VTOCB keyword, DSKPRP automatically assigns the starting VTOC address for you.

Table 9-3 shows the default VTOC starting addresses for both IPL and non-IPL diskette volumes.

Table 9-3. Default Starting VTOC Addresses for Diskette

Diskette	Non-IPL Volume	IPL Volume
8420/8422	002400	002400

After you have indicated the starting address of your VTOC, the next step is specifying the ending address. You can specify the hexadecimal numbers to indicate the cylinder and head address of the ending track on the VTOCE keyword. By omitting the VTOCE keyword, DSKPRP automatically assigns the ending VTOC address for you.

Table 9-4 shows the VTOC ending addresses for both IPL and non-IPL diskette volumes.

Table 9-4. Default Ending VTOC Addresses for Diskette

Diskette	Non-IPL Volume	IPL Volume
8420/8422	002401	002401

Note: The VTOC cannot be written on cylinder 0, head 0.

9.7.13. Building a New VOL1 and VTOC or DSLs (PARTL)

To speed up your prep (since no surface analysis is performed), you specify the PARTL=V keyword. Not only does it run faster, it is also an easy and quick method of scratching unwanted files (rather than using job control). You can only use PARTL=V on a prepped diskette.

You can also specify a new volume serial number through the PARTL=V keyword. The volume serial number indicated on the SERNR keyword is written in the VOL1 label(s). This keyword removes all the entries from the VTOC or DSLs, thus making the diskette unusable unless you know the specific address of your files. If you know the specific address, you use the ADDR parameter on the EXT job control statement during file allocation (format label only).

Whenever you use the PARTL=V keyword, the RECSZ and DNSTY keyword values must be the same as you specified for your original prep. In other words, if you specified RESCZ=256 and DNSTY=2, you cannot change the keyword values in the same control stream with the PARTL=V keyword. If you want to change the RECSZ or DNSTY values, omit the PARTL keyword or use PARTL=N.

9.7.14. Specifying the IMPL Module Type Written to Diskette (IMPNM)

There are six IMPL module types that you can write to diskette: CPU, IDC, IOMP, DBUS, IDCU, and FDD0. The first five (CPU, IDC, IOMP, DBUS and IDCU) are supported on models 3 through 6 with CPU as the default module type on these models. Module types IDCU and FDD0 are the only types supported on model 8 with IDCU as the default type. Only IDCU is supported on models 10/15/20.

When you specify IMPNM, you must also specify the parameter ILOPT=Y and the device assignment set for the YSDF file. The YSDF file must contain the name of the microcode. This name is entered via the system defintion utility (SDU) product. The actual microcode load module resides in YMIC. The prep routine searches YSDF for the microcode name associated with the module type specified by IMPNM. The microcode is then loaded from YMIC and written to the diskette. See the *Installation Guide* (UP-8839) for a description of the IMPLDSKT job stream.

If you are a model 8 only (not models 10/15/20) user, you are supplied with a diskette containing the FDD0 microcode. To back up your FDD0 diskette or to create an updated FDD0 diskette (via an SMC), you must specify the following options: IMPNM=FDD0, ILOPT=Y, and SERNR=S$IMPL. The FDD0 IMPL diskette must be single sided with a record size of 128 bytes and must not contain any defective tracks. If a bad track is found, the diskette prep is terminated with errors. Another diskette is then needed. (Attempting to specify IMPNM=FDD0 on a model 3 through 6 system results in error.) (See the *Installation Guide* (UP-8839) for a description of the FDD0DSKT jobstream that allows you to create a FDD0 IMPL diskette.) To back up your FDD0 diskette on the models 10/15/20, see the *Installation Guide* (UP-8839) for a description of the FDDCOPY jobstream.

9.7.15. Specifying the Control Unit Address for Model 8 through 20 IMPL (CUADR)

This parameter is applicable only to model 8 through 20 users. It allows you to write the IMPL (IDCU) that supports the control units onto a diskette. It is required whenever you specify the ILOPT=Y and IMPNM=IDCU parameters. The value you specify for CUADR must be the control unit address of the IDCU disk(s) and must be device number 0 on the control unit (e.g., 390, 3A0, 280, etc.). The CUADR parameter is not needed if IMPNM=FDD0. It is ignored if specified on model 3 through 6 systems except for syntax checking.

9.8. Initializing the Data Set Labels

The data set labels written on track 00 of your diskette are similar to the VTOC of a disk. The data set labels are reserved for defining the name of a set of files and the addresses associated with the maximum space the set of files can occupy. You must specify a VOL1 statement every time you do a prep.

The format of the VOL1 statement is

```
1                                       42          55
-----------------------------------------------------
VOL1                                    [aaaaaaaaaaaaaa]
```

The VOL1 statement must start in column 1. Columns 42 through 55 are used for a name or address of the diskette. The contents of this optional field is up to the user.

9.9. Executing the Disk Prep (DSKPRP)

When executing DSKPRP to prep any kind of disk or diskette, there are two file names that must always be specified on the // LFD job statements. The file name for the printer must be PRNTR, while the file name of your disk or diskette must be DISKIN. If a track condition table (TCT) and a defect skip table (DST) are wanted as input and/or output to a diskette, the file name, DSKET, must be supplied on the // LFD job statement in the diskette device assignment set. No minimum or maximum main storage sizes should be specified on the // JOB card, as the minimum amount of main storage needed by DSKPRP is automatically allocated for the job by job control, and providing any more than the minimum only wastes valuable main storage space. DSKPRP is not a program that uses main storage dynamically.

The following control streams show some typical DSKPRP examples:

Example 1: Disk 8470 Prep in which Defective Tracks Are Assigned Alternates

```
      1       10    16
      |-------------------------------------------------
1.    | // JOB PREP8470
2.    | // DVC 20 // LFD PRNTR
3.    | // DVC 50 // VOL DS8470(NOV) // LFD DISKIN
4.    | // DVC RES // LBL $Y$SDF // LFD $Y$SDF
5.    | // EXEC DSKPRP
6.    | /$
7.    |    SERNR=DS8470,TRCON=N,
8.    | VOL1
9.    | INSERT 002002,007008,011101
10.   | /*
11.   | /&
12.   | // FIN
```

1. Line 1 is the job control statement identifying your job as PREP8470.

2. Next, enter the required file name for the printer, PRNTR.

3. On line 3, you specify the device assignment set for the disk to be prepped. According to this assignment set, your disk is on device number 50, the volume serial number is DS8470, and you are using the NOV option indicating that no check is required for the volume serial number. The file

name DISKIN must be specified on the // LFD job control statement, since this is the required file name for the disk.

4. The device assignment set in line 4 is required only if your prep is being conducted on a model 8 through 20 system because the parameters used to write the IPL and IMPL to disk default to yes (Y) are IPLDK=Y and ILOPT=Y.

5. The EXEC control statement calls the disk prep routine (DSKPRP) from YLOD.

6. The /$ job control statement indicates the start of the prep data. All your prep keywords or control statements must follow this statement.

7. On line 7, you specify the volume serial number (DS8470) and indicate that a new TCT is to be generated.

8. The VOL1 statement must appear in disk prep control streams and always follows the last specified keyword. The only exception is when you are using the assign alternate track (AAT) feature of the disk prep routine.

9. The INSERT control statement flags the following addresses as defective: 002002, 007008, 011101.

10. through 12.
The /*, /&, and // FIN job control statements terminate your job.

Since no special prep is specified, it defaults to fast prep (PREPT=F). The VTOC is placed on cylinder CA and the TCT is written to an alternate track and printed.

Example 2: Disk 8417 Prep in which the TCT/DST Is Input and Output to a Data Set Label Diskette

```
      1         10    16
1.  // JOB PREP2
2.  // DVC 20 // LFD PRNTR
3.  // DVC 50,100 // VOL WORK(NOV) // LFD DISKIN
4.  // DVC 130 // VOL D08420 // LBL DATA // LFD DSKET
5.  // EXEC DSKPRP
6.  /$
7.          SERNR=WORKDC
8.          TRCON=K
9.          TRKCT=K
10.         PREPT=1
11.         IPLDK=N
12. VOL1
13. INSERT NONE
14. /*
15. /&
16. // FIN
```

1. Line 1 is the job control statement identifying your job as PREP2.

2. The device assignment set for the printer is shown in line 2.

3. You specify the device assignment set for the disk being prepped. According to this assignment, your disk is mounted on the disk drive with the hardware address 100, the VSN is WORK, and you are using the NOV option. The required file name for this assignment set is DISKIN.

4. Line 4 contains the device assignment set for the diskette that is used as the input and output for the TCT/DST. The required file name for the diskette is always DSKET.

5. The EXEC job control statement that calls the disk prep routine.

6. Enter the job control statement (/$).

7. Line 7 specifies the volume serial number as WORKDC.

8. Line 8 specifies that the input of the TCT/DST is from diskette.

9. On line 9, the entry specifies that the output of the updated tables is to diskette, disk, and the printer.

10. and 11.
    Indicate that only one prep pattern is used (PREPT=1) and that the disk is not to contain the IPL code.

12. and 13.
    Contain the VOL1 statement and INSERT NONE (0 INSERT tracks).

Example 3: Disk 8419 Prep in which IMPL/IPL Is Written to Disk

```
      1         10    16
1.  // JOB PREPIMPL
2.  // DVC 20 // LFD PRNTR
3.  // DVC RES
4.  // LBL $Y$SDF
5.  // LFD $Y$SDF
6.  // DVC 50 // VOL BACKUP // LFD DISKIN
7.  // EXEC DSKPRP
8.  /&
9.        SERNR=BACKUP
10.       ILOPT=Y
11.       IPLDK=Y
12. VOL1
13. INSERT NONE
14. /*
15. /&
16. // FIN
```

1. The job control statement identifying your job as PREPIMPL.

2. The device assignment set for the printer.

3. through 5.
   These lines specify the device assignment set for your disk-resident volume. This device assignment set must be specified whenever you are writing IMPL to disk.

6. The device assignment set for the disk being prepped.

7. Line 7 contains the EXEC job control statement that calls the disk prep routine.

8. The job control statement (/$) indicates the start of prep data.

9. Line 9 specifies the volume serial number as BACKUP for the disk being prepped.

10. The ILOPT keyword specifies that IMPL is written to disk.

11. Because IPL must also be written when writing IMPL to disk, IPLDK=Y is specified.

12. Enter the VOL1 statement on line 12.

13. Since there are no INSERT tracks, INSERT NONE is specified.

Example 4: Diskette Prep in the DSL Mode

```
      1         10    16
1.  // JOB PREP8420
2.  // DVC 20 // LFD PRNTR
3.  // DVC 130,320 // VOL D07098(NOV) // LFD DISKIN
4.  // EXEC DSKPRP
5.  /$
6.     SERNR=D07098
7.     RECSZ=256
8.     DNSTY=2,LACEG=03,FDATA=N
9.  VOL1
10. /*
11. /&
12. // FIN
```

1. The job control statement identifying your job as PREP8420.

2. The file name for the printer.

3. In line 3, you specify the device assignment set for the diskette to be prepped. According to this assignment set, your diskette is mounted on the device with the hardware address 320, your volume serial number is D07098, and you are using the NOV option indicating that no check is required for the volume serial number. The file name DISKIN must be specified on the // LFD job control statement because this is the required file name for the diskette.

4. The EXEC job control statement calls the disk prep routine (DSKPRP) from YLOD.

5. The /$ job control statement indicates the start of the prep data. All your prep keywords and control statements must immediately follow the /$ job control statement.

6. In line 6, you specify the volume serial number (D07098).

7. In line 7, you specify a record size of 256 bytes and the diskette format defaults to data set label.

8. In line 8, the density is specified as 2 and the lacing factor as 3. FDATA=N indicates that the file DATA will not be allocated.

   Figure 9-4 shows the printout generated by this control stream. (Although the printout lists a default value for the RETRY parameter, this parameter is not supported when DSKPRP is used for diskettes.)

```
VER 880317      ***************   OS/3   DISC   INITIALIZATION   ***************
                DATE  88/04/26                                      TIME  12:01:09
* CONTROL STREAM PARAMETERS *
SERNR=007098
RECSZ=256
DNSTY=2,LACEG=03,IPLDK=N,FDATA=N
VOL1
* DEFAULT PARAMETERS *
          PARTL=N     RETRY=0A     RPVOL=N     VTOCB=002400  VTOCE=002401
          SPIRL=N     UNXFC=N      FORMT=DSL   ILOPT=N
USAJU NORMAL EOJ + DISKETTE IS GOOD
UNISYS SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE- 88/04/26    TIME- 12:02:58   UPSI- X'00'
VSN-  007098      TYPE- 8420
```

Figure 9-4. Printed Ouput for DSL Diskette Prep

Example 5: Basic 8420 Diskette Prep

```
1         10    16
// JOB D13PREP
// DVC 20 // LFD PRNTR
// DVC 130 // VOL DS8420(NOV) // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=DS8420
VOL1
/*
/&
// FIN
```

When executing the basic 8420 diskette prep, the only keyword required is SERNR, indicating your volume serial number (DS8420). The diskette is prepped in data set label mode, single density, with a record size of 128 bytes.

Example 6: Changing the Volume Serial Number without Prepping

```
1         10    16
// JOB D13CHNG
// DVC 20 // LFD PRNTR
// DVC 130 // VOL DS8420(NOV) // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=DUMMY1,RPVOL=Y
VOL1
/*
/&
// FIN
```

In this example, you are changing the volume serial number from DS8420 to DUMMY1 while leaving the rest of the diskette unchanged.

Example 7: Replacing the CPU IMPL Module on an 8420 Diskette in a Model 3 through 6 Environment

```
      1        10    16
1.  // JOB DSKTIMPL
2.  // DVC // LFD PRNTR
3.  // DVC RES
4.  // LBL $Y$SDF
5.  // LFD $Y$SDF
6.  // DVC 130 // VOL IMPL01(NOV) // LFD DISKIN
7.  // EXEC DSKPRP
8.  /$
9.         SERNR=IMPL01
10.        RPVOL=Y
11.        ILOPT=Y
12. VOL1
13. /*
14. /&
15. // FIN
```

In this example, you are replacing the CPU IMPL initial microprogram load module on your diskette. The job control statements shown in lines 3, 4, and 5 compose the device assignment set for your disk-resident volume. Include them in your job control stream when writing IMPL to a diskette.

Line 6 contains the device assignment set for the diskette being prepped.

The keywords listed in lines 9 through 11 replace the IMPL module CPU on your diskette (IMPNM defaults to the value CPU on models 3 through 6). The ILOPT=Y keyword specifies that IMPL is written to diskette.

9.10. Prep Canned Job Control Streams

The following canned job control streams provide you with a more convenient method of performing certain prep functions without specifying the parameters and job control statements normally required to run them. They are

- CGV/CHGVSN - Changes the volume serial number on a prepped disk.
- SETREL - Preps and allocates RELEASE/SYSRES files.
- COPYREL - Copies selected system files.
- IMPLDSKT - Creates microcode diskette.
- FDD0DSKT - Creates system microcode diskette for model 8.

- FDDCOPY - Creates system microcode diskette for models 10/15/20.
- PRPMIC - Rewrites current IPL/IMPL module on disk.

These functions are initiated from the system console by keying in their associated job control stream names.

The CGV/CHGVSN, SETREL, and COPYREL functions are described in this section. See the *Installation Guide* (UP-8839) for descriptions of the other job control streams.

9.10.1. Change a Volume Serial Number (CHGVSN/CGV)

The CHGVSN/CGV system utility routine changes the volume serial number (VSN) on a previously prepped disk or diskette. The new VSN replaces the old one in the VOL1 record. The required parameters are supplied either on cards in the card reader or as operands with a command keyed in at the system console.

Note: Extreme care should be used if changing the VSN of a SYSRUN, a SYSRES, or a spooling disk. Specification of a duplicate VSN may not cause a duplication error message to be issued, but will severely impact the system. If a duplicate VSN is inadvertently assigned, the entire system must be reinitialized (IPL).

Card Input Keyin (CHGVSN)

When the parameters are input via cards in the card reader, the following command is keyed in at the system console:

```
RU CHGVSN
```

In addition, the following cards must be in the card reader:

```
 1        10    16
┌──────────────────────────────
|//OLDVSN JSET  'old-vsn'
|//NEWVSN JSET  'new-vsn'
|//TYPE   JSET  'disk-type'
|// FIN
```

where:

'old-vsn'
: Specifies the old volume serial number of the previously prepped disk.

'new-vsn'
: Specifies the new volume serial number to be assigned to the prepped disk.

'disk-type'
Specifies the type of disk subsystem being used. The values may be:

Value	Disk Type
16	8416
17	8417
18	8418
19	8419
20	8420
22	8422
30	8430
33	8433
70	8470
94	8494

Note: *For cardless systems, RU CHGVSN will read parameters from diskette.*

System Console Keyin (CGV)

When the parameters are entered as operands with a command keyed in at the system console, the command has the following format:

```
RV CGV,,O=old-vsn,N=new-vsn,T=disk-type
```

Keyword Parameters

O=old-vsn
Specifies the old volume serial number of the previously prepped disk.

N=new-vsn
Specifies the new volume serial number to be assigned to the prepped disk.

T=disk-type
Specifies the type of disk subsystem being used. The values may be

Value	Disk Type
16	8416
17	8417
18	8418
19	8419
30	8430
33	8433
70	8470
94	8494

After you enter the RV CGV command, the CGV job executes the DSKPRP routine to change your volume serial number. When the DSKPRP routine has completed, you receive a printed listing like the one in Figure 9-5.

```
// JOB CGV                                                                                                    L 11:32:51
// NOP ************************************************************                                           L 11:32:53
// NOP *                                                                                                      L 11:32:54
// NOP ************************************************************                                           L 11:32:54
// NOP                                                                                                        L 11:32:54
// NOP *                                                                                                      L 11:32:54
// NOP *                                                                                                      L 11:32:54
// NOP *                                                                                                      L 11:32:54
// NOP                                                                                                        L 11:32:54
// GBL O,N,T                                                                                                  L 11:32:54
// ALTJCS SG$JCS                                                                                              L 11:32:54
// OPTION SCAN,SUB                                                                                            L 11:32:55
// WRTBIG '  D00410 TO','   REL080'                                                                           L 11:32:55
// DVC 20                                                                                                     L 11:32:59
// LFD PRNTR                                                                                                  L 11:32:59
// NOP                                                                                                        L 11:32:59
//50 DSKTYP 8419                                                                                              L 11:32:59
// VOL D00410                                                                                                 L 11:33:01
// LFD DISKIN                                                                                                 L 11:33:02
// OPTION SCAN,SUB                                                                                            L 11:33:02
// EXEC DSKPRP00                                                                                              L 11:33:02
/$                                                                                                            L 11:33:03
//FINISHED NOP                                                                                                L 11:33:03
/&                                                                                                            L 11:33:03
AC01  JOB CGV         ACCT. NO.          ASSIGNED MEMORY=00048128 BYTES (PLUS 003072 BYTE PROLOGUE)   88/06/07   JOB #01  A 11:33:08
AC02  S80-3  SUP140  08.3.052                                                                                 A 11:33:08
JC01  JOB CGV      EXECUTING JOB STEP WRTBIG00 #001 11:33:09                                                  L 11:33:09
AC10  LFD - PRNTR   , FORM NAME - STAND1  , COPIES - 0001, PAGES - 00000001, STEP =001                        A 11:33:14
AC11  STEP #001  (WRTBIG00) USED 00003229 BYTES   ELAPSED WALL CLOCK TIME=00:00:05.104    TOTAL SVC CALLS=00000290    A 11:33:14
AC12      TERM CODE=000      SWITCH-PRIORITY=05   CPU TIME USED          =00:00:01.494    TRANSIENT CALLS=00000014    A 11:33:14
AC13  UPSI SETTING X'00'                                                                                      A 11:33:14
AC19              DEVICE EXCP'S     104=00000144  PRT=00000023                                               A 11:33:14
JC01  JOB CGV      EXECUTING JOB STEP DSKPRP00 #002 11:33:15                                                  L 11:33:15
USAJT  WARNING  WHEN CHANGING VSN OF SYSRES, SYSRUN, OR                                                       L 11:33:20
USAJT           SYSPOOL NO CHECK FOR DUPLICATE VSN IS MADE.                                                   L 11:33:21
AC10  LFD - PRNTR   , FORM NAME - STAND1  , COPIES - 0001, PAGES - 00000001, STEP =002                        A 11:33:23
AC11  STEP #002  (DSKPRP00) USED 00047572 BYTES   ELAPSED WALL CLOCK TIME=00:00:04.093    TOTAL SVC CALLS=00000608    A 11:33:23
AC12      TERM CODE=000      SWITCH-PRIORITY=05   CPU TIME USED          =00:00:03.181    TRANSIENT CALLS=00000037    A 11:33:23
AC13  UPSI SETTING X'00'                                                                                      A 11:33:23
AC19              DEVICE EXCP'S     104=00000329  PRT=00000025                                               A 11:33:23
AC21  JOB TOTALS    USED 00047572 BYTES     TOTAL ELAPSED WALL CLOCK TIME=00:00:17.119    TOTAL JOB SVC CALLS=00000898  A 11:33:25
AC22                                        WALL CLOCK TIME OF ALL STEPS =00:00:13.201    JOB TRANSIENT CALLS=00000051  A 11:33:25
AC23                                        TOTAL CPU TIME OF ALL STEPS  =00:00:06.010    TOTAL JOB EXCP'S   =00000521  A 11:33:25
JC02  JOB CGV      TERMINATED NORMALLY            11:33:25                                                    L 11:33:26
```

Figure 9-5. Sample Listing for CGV Job (Part 1 of 3)

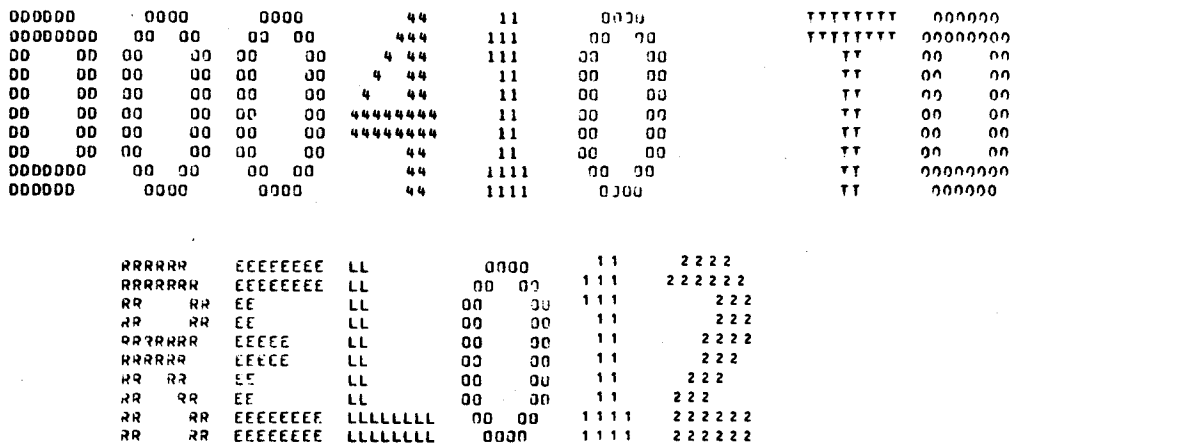


Figure 9-5. Sample Listing for CGV Job (Part 2 of 3)

```
VER 880317          ************** OS/3  DISK  INITIALIZATION  **************

                    DATE  88/05/08                                  TIME  11:33:19

* CONTROL STREAM PARAMETERS *

        RPVOL=Y,SERNR=RELEASE                                          CGV00220
VOL1                                                                   CGV00230


* DEFAULT PARAMETERS *

          ALTRK=Y       ILOPT=Y       INSRT=N       IPLOK=Y       PARTL=N
          PREPT=F       PTREG=000000  PTEND=033706  RETRY=0A       TPCON=D
          TRKCT=2       VERFY=N       VTOCP=00CA00  VTOCL=00CA05  AUTH=A
          UNXFC=N       FRMT=D

USAJT  WARNING  WHEN CHANGING VSN OF SYSRES, SYSRUN, OR
USAJT           SYSPOOL NO CHECK FOR DUPLICATE VSN IS MADE.
USAR1 NORMAL EOJ - DISK IS GOOD

UNISYS SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE- 88/05/08     TIME- 11:33:21    UPSI- X'00'
VSN-  REL060       TYPE- 8419
```

Figure 9-5. Sample Listing for CGV Job (Part 3 of 3)

This listing shows the job control for the CGV job, information about it, the old and new volume serial numbers in large print, and the parameters that CGV includes for your DSKPRP routine. You can keep this listing as a record of the way your disk is prepped. After you change a volume serial number, make sure you physically change the label on the outside of the disk so the label always shows the correct volume serial number. For this sample listing, your command would have been

```
RV CGV,,O=D00410,N=REL080,T=19
```

9.10.2. Prep and Allocate RELEASE/SYSRES Files (SETREL)

The SETREL system utility routine prepares a disk volume for use as a RELEASE or SYSRES volume. It preps the disk and allocates the standard SYSRES files. The file space is allocated according to the type of disk being used. The disk may be prepped with or without full surface analysis. The required parameters can be supplied in the SETREL command itself or on cards. After executing SETREL, you copy the RELEASE or SYSRES volume by executing COPYREL.

System Console or Workstation Keyin

SETREL can be run by keying in the following command at the system console when it is in console mode or at a workstation in system mode:

```
RV SETREL,,V=vsn,T=disk-type,P=prep-type[,CR=NO]
```

Keyword Parameters

V=vsn
: Specifies the volume serial number of the disk. For a full or partial prep, this may be a new VSN to be assigned to the disk. Otherwise, it must be the same as the current VSN. If omitted, the card reader is activated to read the parameters from cards, as though the RU SETREL command were being used. (See "Card Input," which follows in this section.)

T=disk-type
: Specifies the type of disk being used, as follows:

Disk Type	Meaning
16	8416
17	8417
18	8418
19	8419
30	8430
33	8470
70	8470
94	8494

P=prep-type

Specifies the type of prep to be performed, as follows:

Prep Type	Meaning
N	No prep (assign files only)
F	Full prep (with surface analysis)
P	Partial prep (without surface analysis)

If omitted, P is assumed. If N is used, the disk must have already been prepped.

CR=NO

When you enter RV SETREL at the system console and specify a full prep, you may enter bad track addresses on cards or interactively at the console. If you specify CR=NO, prompt messages are displayed at the console so that you can report any known bad tracks by entering their hexadecimal addresses. To enter bad track information on cards, use the RU SETREL command (see "Card Input," which follows in this section).

When you enter RV SETREL to do a full prep from a workstation, prompt messages are automatically displayed, so the CR=NO parameter can be omitted.

Omit this parameter when doing a partial prep or no prep.

Card Input

Another way to run SETREL is to key in the command at the system console but submit the required information on cards. With this method, the cards can contain the keyword parameters and the bad track addresses or just the bad track addresses. When cards are used, the RU SETREL command (rather than RV SETREL) must be used. Also, the command must be entered at the system console when it is in console mode. With this method, all files allocated will be release volume size files.

To run SETREL using cards for keyword parameters and bad track addresses, use the following command format:

```
RU SETREL
```

The cards that must be in the card reader depend on the type of prep being done and the type of disk being prepped, as follows:

- Partial prep without surface analysis

```
1         10    16
//PREP    JSET  '0'
//VSNO    JSET  'vsn'
//TYPE    JSET  'disk-type'
// FIN
```

- Full prep with surface analysis

```
1        10    16
┌──────────────────────────────
│ //PREP   JSET  '1'
│ //VSNO   JSET  'vsn'
│ //TYPE   JSET  'disk-type'
│ // FIN
│ INSERT   { NONE   }
│          { ccchh  }
│ // FIN
```

The values that may be assigned to the various parameters are as follows:

'vsn'
: Specifies the volume serial number of the output disk.

'disk-type'
: Specifies the type of disk subsystem being used. The values are

Value	Disk Type
16	8416
17	8417
18	8418
19	8419
30	8430
33	8433
70	8470

NONE
: Indicates that there are no defective cylinders and tracks on the disk. To prep a selector channel device disk with no defective tracks, omit the INSERT△△△NONE statement but still use both // FIN statements.

ccchh
: Specifies the hexadecimal address of the defective cylinder and track as listed on the disk.

To do a full prep and enter only bad track addresses on cards, include the keyword parameters in the SETREL command as follows:

```
RU SETREL,,V=vsn,T=disk-type,P=F
```

These keyword parameter values are explained earlier in this subsection. The INSERT△△△NONE or INSERT△△△ccchh cards must be present in the card reader. After these, a // FIN card is also needed to turn off the card reader.

Note: *For cardless systems, RU SETREL will read parameters from diskette.*

9.10.3. Copy System/Release File (COPYREL)

After successfully prepping and allocating your disk volume that will contain the system and release libraries using SETREL, you copy those libraries using COPYREL. You execute COPYREL by using the following console keyin:

```
RVΔCOPYREL,,V=vsn,T=disk-type,[,S=first-file][,E=last-file]
```

where:

`V=vsn`
Specifies the volume serial number of the output disk being copied.

`T=disk-type`
Specifies the type of disk subsystem being used. The values are as follows:

Value	Disk Type
16	8416
17	8417
18	8418
19	8419
30	8430
33	8433
70	8470
94	8494

`S=first-file`
Specifies the code identifying the first file to be copied. Table 9-5 shows the order in which COPYREL copies the system files and shows the codes for each system file. These are the only files that can be copied by COPYREL. If you omit the S keyword, COPYREL starts copying at YSRC.

`E=last-file`
Specifies the code for the last file to be copied (see Table 9-5). If you omit the E keyword, copying ends at YTRANA.

Note: *More detailed discussions of SETREL and COPYREL are found in the* Installation Guide *(UP-8839).*

Table 9-5. COPYREL Copy Order

Copy Order	Code	File Name
1	S	$YSSRC
2	O	$YSOBJ
3	L	$YSLOD
4	M	$YSMAC
5	J	$YSJCS
6	G	SG$JCS
7	SGMAC	SG$MAC
8	SGOBJ	SG$OBJ
9	SGLOD	SG$LOD
10	SCLOD	$YSSCLOD
11	MIC	$YSMIC
12	IVP	IVPLIB
13	SMCFILE	SMCFILE
14	SMCLOG	$YSSMCLOG
15	FMT	$YSFMT
16	SAVE	$YSSAVE
17	DIALOG	$YSDIALOG
18	SDF	$YSSDF
19	HELP	$YSHELP
20	T	$YSTRAN
21	A	$YSTRANA

9.11. Error Processing

The prep routine always terminates normally, even if errors are detected during the execution of your prep. Error messages, however, are listed on the printer. At prep completion, the hexadecimal value of the user program switch indicator (UPSI) byte is listed on the printer. The UPSI byte indicates the severity of the errors detected.

When the canned job control streams are run, if unrecoverable errors occur during the prepping of the volume, a message is displayed on the console and the job is terminated immediately. If other errors are encountered, a warning message is displayed, and the job continues processing.

Note: *The console message PREP TERMINATED WITH ERRORS is displayed if the UPSI byte is set (40_{16} or 80_{16}). However, the prep is completed even though an unrecoverable error condition occurred.*

Section 10
Assign Alternate Track (AAT)

10.1. AAT Capability

The AAT routine is used to conditionally or unconditionally assign an alternate track/sector for a suspected defective area on a disk pack. The suspected defective track/sector may be either a primary data track/sector or another alternate track/sector. Assigned conditionally means that an alternate is assigned only after a surface analysis is performed and the analysis shows defects. Assigned unconditionally means that no surface analysis is done and the track/sector is assigned regardless of its condition.

Note: *The 8494 disk only supports unconditional assignments.*

For conditional assignment, an available alternate track is assigned by AAT and the contents of the suspected defective track are copied to it, one record at a time. During the copy procedure, any errors detected are listed on the printer. These errors can be corrected in subsequent runs of AAT by using the record updating facility, ASUPD. After the records are copied to the alternate track, a surface analysis is performed on the primary track. If the track is found to be defective, it is marked as unusable and the alternate track is permanently assigned. However, if the primary track is found to be usable, then all the records on the alternate track are copied back to the primary track and the alternate track is again made available for use.

For an absolute or unconditional assignment, the process is similar. The suspected defective track/sector is read and the data is saved. Any defective records are printed. At this point, the defective track/sector is assigned and the recovered data is rewritten. No surface analysis is performed on the suspected track/sector. Remember that a suspected defective area can be an alternate as well as a primary track/sector. If it is an alternate, another alternate is reassigned as the alternate for the original primary data track/sector. The suspected alternate is then assigned to itself so that it cannot be used again.

Caution

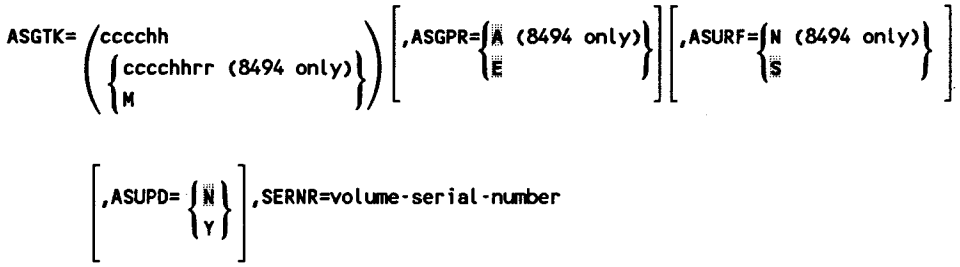
If AAT terminates abnormally, the track condition table (TCT) may be compromised and the disk files rendered inaccessible.

10.2. Interfacing with DSKPRP

The AAT function is a feature built into DSKPRP; therefore, you must execute DSKPRP (specify DSKPRP on the EXEC job control statement) to use the AAT capability. Before assigning any alternate tracks, your disk must have been previously prepped.

10.3. Specifying AAT Options

There are five keywords associated with the assigning of any alternate tracks. You must specify both the ASGTK and SERNR keywords; the remaining keywords are optional and have default values. The format of the AAT keywords is



10.3.1. Specifying Any Suspected Defective Tracks (ASGTK)

Since the AAT capability is a function of the disk prep routine, you must specify ASGTK to indicate the function to be performed. You specify the suspected defective track in hexadecimal cylinder/head format (*cccchh*); or, for 8494 disks, the defective sector in cylinder/head/record format (*cccchhrr*). For 8494 disks, you also have the option of interactively entering the addresses of the defective sectors by specifying ASGTK=M (multiple). If you specify ASGTK=M, a message is displayed requesting the disk address (*cccchhrr*) or END. This interactive method cannot be used if update records are specified (ASUPD=Y).

Remember, you cannot assign an alternate track/sector to an active SYSRES pack. If you need to assign alternate tracks/sectors to your SYSRES, another SYSRES or the current release volume must be used as the operating system. If you omit ASGTK, the disk prep routine is executed; however, the disk prep routine will be terminated because there is no VOL1 statement present in the control system. VOL1 cards cannot be used in an AAT run.

10.3.2. Printing Your Records (ASGPR)

When reading the records from the track specified by the ASGTK parameter, any records detected in error are listed on the printer by the E option automatically. However, if you need to print all the records being read, specify the A option. If you use the A option in conjunction with the ASUPD=Y keyword, no records will be printed. In other words, ASGPR=A has no effect when you specify ASUPD=Y. ASGPR=A is the default for 8494 disks; ASGPR=E is the default for all other disks.

10.3.3. Testing the Alternate Track (ASURF)

If you recall, you can assign the alternate track conditionally, that is, only after ensuring that the primary track is defective, or unconditionally, meaning no surface analysis is performed. By omitting ASURF or specifying the S option, alternate tracks are assigned conditionally. If you are certain the primary track is unusable, specify ASURF=N and no surface analysis will be performed.

***Note:** The 8494 disk only supports unconditional assignments.*

10.3.4. Patching or Modifying Existing Records (ASUPD)

If any errors are detected in the reading of your records from the primary track for writing on the alternate track, they are listed on the printer. From this listing, you key in the missing information into update records. These update records will then patch the incomplete record. However, if the missing information cannot be determined, you must recreate your file.

Whenever update records are present in your control stream, you must specify the keyword parameter ASUPD=Y. If there are no update records present, the default ASUPD=N is assumed.

At least two cards are required for each update record; one to identify the record and field to be updated and another to contain the correction data. The number of the record to be updated must always be specified as a hexadecimal number and must always begin in column 1.

The record numbers on a given track appear on the listing of errors found during the recovery of data during a previous AAT run. These records are listed in decimal and must be converted to hexadecimal for the update. The length and displacement of the field being corrected must also be identified by specifying DATA=([*d*],*L*), where *d* represents a displacement value and *L* represents a length value.

The displacement value, relative to 0, may be up to four hexadecimal characters long, or may be omitted, to indicate that the patch is to be made beginning with the first character in the identified record field. The length value must be specified and can be from one to four hexadecimal characters long.

The DATA keyword can be coded on the card containing the number of the record (rn parameter) to be patched. If coded with the rn parameter, a comma must separate the two specifications. If coded alone, the keyword must begin in column 1.

Thus, the format of an identifier update record could be illustrated as

rn *or* rn,DATA=$\left(\left[\left\{\begin{matrix}d\\0\end{matrix}\right\}\right],L\right)$ *or* DATA=$\left(\left[\left\{\begin{matrix}d\\0\end{matrix}\right\}\right],L\right)$

The actual data to be written must be submitted on a separate card apart from the update record. The actual data must be in hexadecimal format (0 through 9, A through F) and start in column 1. The length of the data must equal the length value specified in the DATA parameter. No embedded blanks are permitted and as many cards as needed may be used. The first blank character found in the data cards indicates the end of the record. The maximum amount of data per line is 80 hexadecimal digits.

It should also be noted that only one DATA keyword may appear in any one set of update records, and when ASUPD=Y is specified, the ASGPR and ASURF keywords are ignored but checked for syntax errors.

A typical example of an update control stream could look like:

```
1         10    16
.
.          } Job control statements
.
/$
  ASGTK=001A06,SERNR=DSP001
  ASUPD=Y
/*
/$
1B
DATA=(3C,A)
C1C2C3C4C5C6C7C8C9D1
/*
/&
```

In this example, record 1B is being updated. There is a displacement value of 60 bytes (3C) and a length of 10 bytes (A). The new data being written is specified on a separate card and must be in hexadecimal format. If more than one record is to be updated, an additional data set is needed for each record. You will see more typical examples of using update records in 10.4.

10.3.5. Specifying the Disk Volume Serial Number (SERNR)

The volume serial number consists of six alphanumeric characters that make up the serial number of the disk volume being used. The SERNR keyword must always be present in your AAT control stream.

10.4. Executing AAT

When executing the AAT routine, the same file names that were required in the disk prep routine must be specified here. Namely, PRNTR must be specified on the LFD job control statement for assigning the printer while DISKIN must be specified for assigning the disk. In the following control streams, you will see some typical examples of AATs.

Example 1

```
1        10    16
// JOB AATRACK
// DVC 20 // LFD PRNTR
// DVC 51 // VOL REL120  // LFD DISKIN
// EXEC DSKPRP
/$
   ASGTK=00C106,SERNR=REL080
/*
/&
// FIN
```

Here is the basic AAT control stream. The suspected defective track is located on cylinder 00C1, head 06, on disk REL080. You are omitting the ASGPR, ASURF, and ASUPD keywords, thus using the default options. Any records found to be in error are listed on the printer (ASGPR=E), a surface analysis is performed on the track (ASURF=S), and there are no update records present in your control stream.

Figure 10-1 shows the output generated from this control stream.

```
VER 880317          ********** OS/3 DISC INITIALIZATION **********
                    DATE 88/06/02                          TIME 12:56:36
* CONTROL STREAM PARAMETERS *
ASGTK=00C106,SERNR=REL080

* DEFAULT PARAMETERS *
        ALTRK=Y     ASGPR=E     ASUPD=N     ASURF=S     ILOPT=N
        INSRT=N     IPLOK=Y     PARTL=N     PREPT=F     PTSEG=00C106
        PTEND=00C106 RETRY=0A   RPVOL=N     TPCON=D     TRKCT=Z
        VERFY=N     VTOCB=00CA00 VTOCE=00CA06           UNXFC=N
```

Figure 10-1. Basic AAT (Part 1 of 2)

```
USA97 PRIMARY TRACK SPECIFIED WAS FOUND GOOD BY SURFACE ANALYSIS

                                OS/3  TRACK  CONDITION  TABLE

                                SERIAL NUMBER RELOBJ

                                                                BYTES   FIRST
                                                                WRITTEN BAD
ALTERNATE TRACK  CODE BYTE  PRIMARY   TRACK    ON BAD   BYTE    BYTE  BYTE
CYLINDER  HEAD   0123 4567  CYLINDER  HEAD     TRACK    FOUND   READ  EXPECTED
--------  ----   ---- ----  --------  ----     -------  -----   ----  --------
  0328     00     1    4  7   0033     03       0000     0000    00     00
  0328     01     1    4  7   003F     00       0000     0000    00     00
  0328     02     1    4  7   0104     03       0000     0000    00     00
  0328     03     1    4  7   00C5     03       0000     0000    00     00
  0328     04     1    4  7   0153     02       0000     0000    00     00
  0328     05       3    6   0000     00       0000     0000    00     00
  0328     06     2            0000     00       0000     0000    00     00
  0329     00     2            0000     00       0000     0000    00     00
  0329     01     2            0000     00       0000     0000    00     00
  0329     02     2            0000     00       0000     0000    00     00
  0329     03     2            0000     00       0000     0000    00     00
  0329     04     2            0000     00       0000     0000    00     00
  0329     05     2            0000     00       0000     0000    00     00
  0329     06     2            0000     00       0000     0000    00     00
  032A     00     2            0000     00       0000     0000    00     00
  032A     01     2            0000     00       0000     0000    00     00
  032A     02     2            0000     00       0000     0000    00     00
  032A     03     2            0000     00       0000     0000    00     00
  032A     04     2            0000     00       0000     0000    00     00
  032A     05     2            0000     00       0000     0000    00     00
  032A     06     2            0000     00       0000     0000    00     00
  032B     00     2            0000     00       0000     0000    00     00
  032B     01     2            0000     00       0000     0000    00     00
  032B     02     2            0000     00       0000     0000    00     00
  032B     03     2            0000     00       0000     0000    00     00
  032B     04     2            0000     00       0000     0000    00     00
  032B     05     2            0000     00       0000     0000    00     00
  032B     06     2            0000     00       0000     0000    00     00
  032C     00     2            0000     00       0000     0000    00     00
  032C     01     2            0000     00       0000     0000    00     00
  032C     02     2            0000     00       0000     0000    00     00
  032C     03     2            0000     00       0000     0000    00     00
  032C     04     2            0000     00       0000     0000    00     00
  032C     05     2            0000     00       0000     0000    00     00
  032C     06     2            0000     00       0000     0000    00     00
  032D     00     2            0000     00       0000     0000    00     00
  032D     01     2            0000     00       0000     0000    00     00
  032D     02     2            0000     00       0000     0000    00     00
  032D     03     2            0000     00       0000     0000    00     00
  032D     04     2            0000     00       0000     0000    00     00
  032D     05     2            0000     00       0000     0000    00     00
  032D     06     2            0000     00       0000     0000    00     00
  032E     00     2            0000     00       0000     0000    00     00
  032E     01     2            0000     00       0000     0000    00     00
  032E     02     2            0000     00       0000     0000    00     00
  032E     03     2            0000     00       0000     0000    00     00
  032E     04     2            0000     00       0000     0000    00     00
  032E     05     2            0000     00       0000     0000    00     00
  032E     06     2            0000     00       0000     0000    00     00

    BIT 0= ALTERNATE TRACK LISTED IS DEFECTIVE
        1= PRIMARY TRACK LISTED IS DEFECTIVE AND AN ALTERNATE HAS BEEN ASSIGNED
        2= ALTERNATE TRACK LISTED IS GOOD, BUT NOT ASSIGNED
        3= ALTERNATE TRACK LISTED IS GOOD AND HAS BEEN ASSIGNED BY THIS RUN
        4= THIS DEFECTIVE TRACK FOUND BY SURFACE ANALYSIS
        5= THIS DEFECTIVE TRACK DESIGNATED BY CARD INSERT
        6= THIS ALT TRACK CONTAINS THE TRACK CONDTN TABLE
        7= THIS ALT TRACK WAS ASSIGNED BY A PREVIOUS RUN

USAR1 NORMAL EOJ - DISK IS GOOD

UNISYS SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE=  88/06/02      TIME= 12:56:41    UPSI= X'00'
VSN=   RELOBJ        TYPE= 8419
```

Figure 10-1. Basic AAT (Part 2 of 2)

Example 2

```
1        10    16
-
// JOB UPDATE
// DVC 20 // LFD PRNTR
// DVC 51 // VOL REL120 // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=TST93J
   ASGPR=A
   ASUPD=Y
   ASGTK=00A106
/*
/$
14,DATA=(,32)
11111111111111111111111111111111111111111111111111111111111112233333333444444555
55555555667788 99C8C9
/*
/&
// FIN
```

Update records are used to patch specific key and data fields on your disk. The volume serial number of your disk is REL120, which is equated with the VOL job control statement. Since update records are present in your control stream, you specified ASUPD=Y. As you can see, all the keyword parameters are part of the first data set. The record data set contains the update records themselves. The record number in error is 14 (decimal 20). The data being corrected has a length of 32_{16} (decimal 50) bytes and is shown on the next two cards.

Note: *If more than one card of data is required, the previous card must have all 80 columns in use.*

Figure 10-2 shows the output generated from the control stream.

```
VER 880317            ***************  OS/3  DISC  INITIALIZATION  ***************

                      DATE   88/06/07                                  TIME  15:17:43

* CONTROL STREAM PARAMETERS *

SERNR=REL120
ASGPR=A
ASUPD=Y
ASGTK=00A106


* DEFAULT PARAMETERS *

          ALTRK=Y        ASURF=S        ILOPT=N        INSRT=N        IPLDK=Y
          PARTL=N        PREPT=F        PTBEG=00A106   PTEND=00A106   RETRY=0A
          RPVOL=N        TRCON=0        TRKCT=2        VERFY=N        VTOCB=00CA00
          VTOCE=00CA06                  UNXFC=N


14,DATA=(,32)
1111111111111111111111111111111111111111111111111111111111111111111111112233333333344444455
5555555566778899C8C9

USAR1 NORMAL EOJ - DISK IS GOOD


UNISYS SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE-  88/06/07     TIME- 15:17:46    UPSI- X'00'
VSN-   REL080       TYPE- 8419
```

Figure 10-2. AAT Using Update Records

Section 11
Tape Prep (TPREP)

11.1. Preparing Your Tape for Execution

Magnetic tapes are shipped blank. You must prepare (prep) these tapes before using them in OS/3. The tape prep utility (TPREP) can prep up to 36 files and up to 16 volumes for each file in one job step. It will prep them for use with or without block numbers, depending on whether your system is configured to support tape block numbering. If your system is configured to support tape block numbering, you have the option to prep tapes for use without block numbers. You cannot, however, prep a tape for use with block numbers if your system is not configured to do so.

You submit all the required information for tape prepping through job control statements. TPREP uses the prep facilities of data management to prep your tapes. The various tape record formats that are generated by the prep facility can be found in the current version of the *Consolidated Data Management Programming Guide* (UP-9978).

After your tape has been prepped, it is rewound to load point.

11.2. Tape Prep Coding Instructions

All the information required for tape prepping is submitted through job control statements. The following job control statements are used for tape prepping:

- // DVC job control statement

  You must specify a logical unit number for the tape being prepped. The range of logical unit numbers is from 90 to 127. An example of the // DVC job control statement used when prepping tapes is shown in the following example:

```
1        10    16
|// DVC 90
```

- // VOL job control statement

  You must specify a unique volume serial number for every tape being prepped. The volume serial number can be any alphanumeric character string from one to six characters long, other than the word SCRTCH. Immediately following the last character of your volume serial number, the character string (PREP) must appear. An example of the // VOL job control statement used when prepping tapes is shown in the following example:

```
1        10    16
|// VOL TAP028(PREP)
```

  Note: *The parentheses are coded as part of the parameter.*

  To prevent your tapes from being prepped for use with block numbers when your system is supporting block numbering, you must include an N parameter in your // VOL statement as follows:

```
1        10    16
| // VOL  N,DSP028(PREP)
```

- // LBL job control statement

  You specify the // LBL job control statement only when you want to assign a file identifier to a tape volume you are prepping. You can assign the same label to multiple volumes to prepare for a multivolume file. If you are familiar with the // LBL job control statement, then you would have recognized that there is a file sequence number parameter used for the numbering of files in a multifile tape volume. However, TPREP ignores the file sequence number parameter. An example of the // LBL job control statement follows:

```
1        10    16
|// LBL MASTERFILE
```

- // LFD job control statement

  You must specify a unique file name for each tape file being prepped. The // LFD file name must be in the form TAPE*xy*: under *x* is any alphanumeric character A through Z or 0 through 9; under *y* is the character A for ASCII mode or blank for EBCDIC mode. An example of the LFD job control statement used when prepping tapes in both EBCDIC and ASCII modes is shown in the following example:

```
1        10    16
|// LFD TAPE1
|// LFD TAPE1A
```

- // EXEC job control statement

  You must specify the program TPREP in your // EXEC job control statement to start the prepping of one or more tapes using TPREP. The // EXEC job control statement calls the tape prep utility from YLOD. You code the // EXEC job control statement as follows:

```
1        10    16
|// EXEC TPREP
```

The following examples show two tape preps using TPREP. Example 1 shows the job control statement for prepping six tapes using two magnetic tape drives. Example 2 shows the job control statements for prepping six tapes using six different tape drives, with the last three tapes being prepped in ASCII mode. All other tapes are prepped in EBCDIC mode.

Example 1

```
1        10    16
|// JOB TAPEPREP
|// DVC 90
|// VOL TP0001(PREP),TP0002(PREP),TP0003(PREP)
|// LFD TAPE1
|// DVC 91
|// VOL TP0004(PREP),TP0005(PREP),TP0006(PREP)
|// LFD TAPE2
|// EXEC TPREP
|/&
|// FIN
```

Example 2

```
1        10    16
|// JOB TAPEPREP
|// DVC 90 // VOL TAPE01(PREP) // LFD TAPE1
|// DVC 91 // VOL TAPE02(PREP) // LFD TAPE2
|// DVC 92 // VOL TAPE03(PREP) // LFD TAPE3
|// DVC 93 // VOL TAPE04(PREP) // LFD TAPEAA
|// DVC 94 // VOL TAPE05(PREP) // LFD TAPEBA
|// DVC 95 // VOL TAPE06(PREP) // LFD TAPECA
|// EXEC TPREP
|/&
|// FIN
```

All messages are displayed on the system console and are written to the communications output printer (COP) if it is available. A typical COP listing follows:

```
14 JC01 JOB TAPEPREP EXECUTING JOB STEP TPREP000 #001
15 TP01 TAPE SP0001 (LFDNAME=TAPE1A) PREPPED IN ASCII, NO BKNO
16 JC02 JOB TAPEPREP TERMINATED NORMALLY
```

The following message is displayed for each tape that is successfully prepped:

```
TP01 TAPE vsn (LFDNAME=filename) PREPPED IN {ASCII } , {NO  } BKNO
                                           {EBCDIC}   {WITH}
```

The absence of this message for any of the tapes specified in your job or job step indicates that the tape has not been prepped because of incorrect job control specifications.

If you are creating a file through data management, you may perform tape prepping and file creation in one job step. See the current version of the *Consolidated Data Management Programming Guide* (UP-9978).

Section 12
Disk Dump/Restore Routine (DMPRST)

12.1. DMPRST Concept

Many of the program libraries and data files stored on disk are vital to your data processing operation. Therefore, it is important to keep backup copies of your data handy in case it is lost or destroyed. The disk dump/restore routine (DMPRST) enables you to create these backups on disk, tape, diskette, or streaming tape.

You can execute DMPRST either interactively from a workstation or in a batch environment using cards. However, you'll see that the interactive method is easier to use. Both the interactive and batch methods accomplish the same result; however, there are some differences between what each method supports. These differences are listed in Table 12-1.

Table 12-1. DMPRST Differences between Interactive and Batch Methods

Item	Interactive	Batch
Modes of processing	File	Volume and File
RESTART function	Not supported	Supported
Tape-to-tape copy	Not supported	Supported (volume mode)
Diskette-to-diskette copy	Not supported	Supported
Printer assignment	Automatically assigned	User responsibility

Notes:

1. *DMPRST cannot process system files (prefixed by Y) on an active SYSRES; i.e., the SYSRES pack you are executing from. To process these files, see the current version of the* Installation Guide *(UP-8839).*

2. *In addition to DMPRST, Unisys also provides three canned job control streams (SG@DUFIL, SG@DSFIL, and SG@RUFIL) and a standalone routine (SU@RST) that you can use to dump and restore your disk files. These are most useful when you are continually making backup copies of your files. Even though they are designed as a post-SYSGEN procedure for backing up SYSRES files, you can use them for any program file. For more information, see the* Installation Guide *(UP-8839).*

3. *The standalone dump/restore program (SU@RST) is used during system installation to read diskettes or tapes (models 8 through 20 only) created in file mode by the standard dump/restore program and write tracks of data to fixed disks. The program also copies the VTOC when requested by the user. SU@RST cannot process tapes created with multiple disk volume input.*

4. *Before running DMPRST with a supervisor that is not configured to the specific system hardware, be sure to turn on the drive(s) for the input (output) disk(s) being prepped. This reconfigures the information in the PUB(s). Otherwise, the job uses the current PUB information, which may not be representative of the disk(s).*

When dumping a disk in volume mode, DMPRST must be the only job running if the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonshareable using the SET IO operator's command. Refer to the *Operations Guide* (UP-8859) for details. Making the device nonshareable ensures that an exact copy is made.

12.2. DMPRST Procedures

You can use two basic DMPRST procedures to create backups.

1. Disk copy

2. Dump/restore

Perform both procedures using the DMPRST routine. However, the devices involved and the physical creation of the backup in each procedure are different.

12.2.1. Disk Copy Procedure

The disk copy procedure involves a single DMPRST disk copy operation. Here, you are copying a disk or a portion of a disk to a disk of the same type. For example, if you are backing up an 8417 disk volume by performing a disk copy operation, your backup disk must be another 8417 disk.

Since the disk you are using to store the backup in a disk copy operation is the same type as the original, the data is copied to the backup exactly as it appears on the original. So, if your original data is lost, you can mount the backup disk and use it in place of the original.

12.2.2. Dump/Restore Procedure

You use the dump/restore procedure when the backup device is a different type than the original. In this case, you must run DMPRST twice: first to *dump* the contents of the disk or disks to a backup device and again to *restore* the contents of the backup to the original disk or disks, or to a disk or disks of the same type as the original.

The backup device created by the dump operation cannot be used directly on the system because the data is formatted differently on the backup device. Only the restored device can be used.

Normally, only one disk (or files from one disk) can be dumped/restored. However, multidisk volumes can be dumped/restored to tape in a multidisk environment (see 12.3). If you are backing up a multidisk volume file, you must dump it to tape by running DMPRST in a multidisk volume environment, or you must dump it from each disk with a separate execution of DMPRST. Output from multiple executions of DMPRST should not be written to the same tape or to the same sequential file on disk or diskette.

Dump restore supports streaming tape the same as it supports any other magnetic tape device used to create backups for your data. The streaming operation of this device when running in its default mode of 100 ips is only guaranteed when you execute the DMPRST routine in a dedicated environment. To utilize the streaming capability of this device when executing DMPRST in a nondedicated environment, operate the unit at its low speed mode of operation (25 ips).

Notes:

1. *Since alternate track processing is transparent to DMPRST, to restore a disk pack containing $IPL and $IMPL files, you may need to use the DSKPRP routine. See Section 9 for more information.*

2. *Tapes created from an 8430/33 disk on Release 8.2 and subsequent releases cannot be used to restore the disk on releases prior to Release 8.2.*

3. *Tapes created by dumping files from multidisk volumes cannot be used by DMPRST on releases prior to Release 12.0, nor can they be used by any release of SU@RST.*

12.2.3. Tape and Diskette Copy Operations

In addition to the disk copy and dump/restore operations, the DMPRST routine also performs tape-to-tape and diskette-to-diskette copy operations. The tapes and diskettes used in these operations must be created by a previous DMPRST operation. Tape-to-tape copy operations, however, can only be performed in the volume mode in a batch environment. Diskettes must be prepped as data set label diskettes having a record size of either 128 or 256 bytes.

12.3. Executing DMPRST in an Interactive Environment

Interactive processing of the DMPRST routine consists of a question and answer session (dialog). The answers you give tell the DMPRST routine what function to do, assign the necessary input and output devices, and cause DMPRST to be executed in the file mode. The DMPRST routine supplies default values whenever possible. Any errors detected are blinked on the screen with explanatory messages.

Help screens are provided for all work screens. To obtain HELP for any screen, press function key 13. To return to the work screen, press function key 14 or the XMIT key. If multiple help screens are being displayed, press the XMIT key after each help screen. When the last help screen is displayed, press the XMIT key or function key 14 and the current screen is redisplayed.

After DMPRST is executed, a summary report is automatically printed. The workstation is freed from the job executing DMPRST after the last screen is processed and is noted by displaying the following informational message screen (HU25):

```
                    HARDWARE UTILITIES        HU25

          THE INTERACTIVE INTERFACE FOR THIS PROGRAM
        HAS NOW BEEN COMPLETED. THIS WORKSTATION WILL BE
        AVAILABLE FOR USE WHILE THE PROGRAM PROCESSES AS
        YOU REQUESTED.
```

To conduct an interactive dialog with the DMPRST routine, you must:

- Log on to the system by keying in the LOGON command.
- Initiate the dialog by keying in the HU command (in system mode).
- Display the menu screen (first screen) by pressing the XMIT key.

If your system is a model 3 through 6, menu screen HU00B is displayed. If your system is a model 8 through 20, menu screen HU00C is displayed.

Hardware Utilities Menu HU00B (Models 3 through 6)

```
                    HARDWARE UTILITIES        HU00B

          1.  DUMP FILES FROM A DISK(S)
          2.  RESTORE FILES TO A DISK(S)
          3.  LIST FILES ON A BACKUP MEDIUM
          4.  COPY FILES FROM DISK TO DISK
          5.  COPY AND/OR VERIFY 8419 DISK
          6.  NONE OF THESE

                                        ENTER SELECTION _
```

Hardware Utilities MENU HU00C (Models 8 through 20)

```
                    HARDWARE UTILITIES      HU00C

          1. DUMP FILES FROM A DISK(S)
          2. RESTORE FILES TO A DISK(S)
          3. LIST FILES ON A BACKUP MEDIUM
          4. COPY FILES FROM DISK TO DISK
          5. COPY AND/OR VERIFY 8419 DISK
          6. COPY AND/OR VERIFY 8416/8418 DISK
          7. COPY AND/OR VERIFY 8430/8433 DISK
          8. NONE OF THESE

                                  ENTER SELECTION __
```

The procedure for executing DMPRST in an interactive environment is the same for all models of System 80. The HU00C menu offers two additional copy operations (8416/8418 and 8430/8433 disks) that are not applicable to models 3 through 6. Any differences in the screens displayed in a procedure are noted when applicable to a particular model.

The next set of screens displayed depends on the particular operation you select from the menu. To make a selection, key in the appropriate number for the operation you want to perform and press the XMIT key; DMPRST does the rest.

Typical examples using the copy, dump, restore, and list DMPRST functions (numbers 4, 1, 2, and 3) are described in 12.3.1 through 12.3.4. Examples of the other copy operations (numbers 5, 6, and 7) are described in Section 15. When looking at the screens, the default values are shaded and user entries are shown in REVERSE PRINT.

12.3.1. Performing a Disk Copy Operation

During a disk copy operation, you copy the contents of a disk to another disk of the same type. You perform a disk copy by keying in 4 from the selections on the hardware utilities menu screen (see 12.3). Up to four screens can be displayed. They are

- Screen 1 (HU00BI03) is an informational screen that tells you that the interactive job you selected has been initiated.
- Screen 2 (HU12) asks you to specify the input and output device information.
- Screen 3 (HU16) asks you to specify the file options that are in effect.
- Screen 4 (HU17) appears only if individual files are being copied and it asks you for the file name, relocation option, and rename specifications.

Now, let's look at a typical disk copy operation.

Screen 1 (HU00BI03): Informational Message

```
                                                        HU00BI03
A CONVERSATIONAL JOB (HU$CPY) TO COPY FILES FROM ONE DISK TO
ANOTHER WILL BE INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM
MODE FOR THE JOB TO BE SCHEDULED. IF YOU ENTERED HARDWARE
UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN SYSTEM MODE
AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU COMMAND YOU
ARE RESPONSIBLE FOR GOING INTO SYSTEM MODE.
                  *****  TRANSMIT TO CONTINUE  *****
```

This is an informational screen. Read it and press the XMIT key to continue.

Screen 2 (HU12): Input and Output Devices

```
                  COPY INPUT-OUTPUT DEVICE INFORMATION          HU12

ENTER SPECIFIC INPUT DISK DEVICE TYPE:        8419

ENTER INPUT VOLUME SERIAL NUMBER:             PUBDSK

ENTER SPECIFIC OUTPUT DISK DEVICE TYPE:       8419

ENTER OUTPUT VOLUME SERIAL NUMBER:            REL120

********** FUNCTION KEYS:  F13=HELP, F14=EXIT HELP *********
```

Looking at Screen 2, we see that both our input and output device type is 8419, our input volume serial number is PUBDSK, and our output volume serial number is REL120. Since we don't need any help, press the XMIT key.

Screen 3 (HU16): File Options

```
                                COPY                  HU16

  WAIT FOR UNLOCK                                  WAIT=NO

  COPY ALL FILES                                   ALL=NO

  UNEXPIRED FILE CHECKING                          UNXF=YES

  ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

We enter NO in Screen 3 to override the default YES for the WAIT FOR UNLOCK option. This indicates that, if the file is unavailable to be locked, skip the file and continue to the next one. Any files skipped will have their file names listed on the printer.

Since we do not want to copy all the files, we accept the default NO. This provides for file selection.

Finally, if we want to select UNEXPIRED FILE CHECKING, we press the XMIT key (assuming the default YES). This option prevents copying over any files on the output volume whose dates have not expired. (The expiration date was generated at file creation time using the // LBL job control statement.)

Screen 4 (HU17): Specifying Input File Names

```
                         COPY                    HU17

ENTER

   .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP OF FILES

=FILENAME,ALLOCATION,NEWNAME

FILE _ _ = CPYLIB
FILE _ _ = MENUFILE
FILE _ _ = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _,_ _ _,

ARE MORE FILES TO BE ENTERED?  NO

**********FUNCTION KEYS: F13=HELP, F14=EXIT HELP **********
```

Here, we specify our input file names as CPYLIB and MENUFILE. Any other files residing on our volume are not copied. You also have the option of specifying a file prefix rather than the entire file name. To indicate a prefix, enter .P immediately before the equals sign (=); enter the prefix name (up to 16 characters) immediately after the equals sign. Since we don't need to copy our file to a specific area on our output volume, we ignore the ALLOCATION and NEWNAME options and press the TAB key to the next query ARE MORE FILES TO BE ENTERED? Since we are only copying two files, we press the XMIT key. The processing of our files begins. When the processing is completed, a summary report is produced for the operation.

Notes:

1. *If you answered YES to COPY ALL FILES on Screen 3 (HU16), then Screen HU34 is Screen 4 instead of HU16. Screen HU34 requests an allocation option to be applied to all the files to be copied. All active files listed in the VTOC are copied.*

2. *The file allocation parameters are described in the following listing. For a complete description, see 12.5.3.*

   ABS *Locate a file on the same absolute extents. If a file cannot be allocated to the same extents, an error message is displayed, the file is bypassed, and processing continues.*

REL — *Requests that the file be relocated to a file the same size as the original.*

LOG — *Relocates a file and deletes all unassigned space for that file. This option deletes only space that is not assigned to a partition control appendage (PCA).*

PRE — *Relocates a file in a space that was preallocated. This space can be located anywhere on the disk.*

Omit — *Standard restore processing - DMPRST scratches the file from the output disk and makes up to three attempts to reallocate it on the disk.*

12.3.2. Performing a Dump Operation

The dump operation copies the contents of your disk volume to a MIRAM file on a diskette, another type of disk, or a tape. (If the output medium is tape, files from multidisk volumes can be dumped.) Later, a restore operation is required to copy the data back from your MIRAM file to your original disk volume in executable format. We discuss the dump operation for both single-disk and multidisk volumes here and the restore operation in 12.3.3.

You perform a dump operation by keying in 1 from the selections on the menu screen. Up to seven additional screens can be displayed. They are

- Screen 1 (HU00BI01) is an informational screen that tells you that the interactive job you selected has been initiated.
- Screen 2 (HU18) asks you for the input disk device type.
- Screen 3 (HU19) asks you for the output device type.
- Screen 4 (HU26 or HU27) asks you to specify the input volume serial number(s). Screen HU26 also requests a disk identifier for each volume to associate it with the files to be dumped from it.
- Screen 5 (HU20 or HU21) asks you to specify your output volume serial numbers and the MIRAM file name, when appropriate.
- Screen 6 (HU22 or HU28) asks you to specify the file options.
- Screen 7 (HU23 or HU29) only appears if individual files are being dumped rather than dumping all the files on the volume. It asks for the file names. Screen HU29, which is displayed for multidisk input, also requests the file identifier for the disk from which each file is to be dumped.

Performing a Single-Volume Disk Dump Operation

Screen 1 (HU00BI01): Informational Message

```
                                                          HU00BI01

     A CONVERSATIONAL JOB (HU$DMP) TO DUMP FILES FROM DISK(S) WILL BE
     INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM MODE FOR THE JOB
     TO BE SCHEDULED. IF YOU ENTERED HARDWARE UTILITIES THROUGH THE
     HU COMMAND YOU WILL BE IN SYSTEM MODE AFTER TRANSMITTING.
     IF YOU ENTERED THROUGH THE MENU COMMAND YOU ARE RESPONSIBLE FOR
     GOING INTO SYSTEM MODE.
                    *****  TRANSMIT TO CONTINUE  *****
```

This is an informational screen. Read it and press the XMIT key to continue.

Screen 2 (HU18): Defining Input Device

```
                      DUMP INPUT DEVICE INFORMATION          HU18

          ENTER INPUT DISK DEVICE TYPE: 8419

          ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On Screen 2, we indicate that our disk device type is an 8419. No help is required, so press the XMIT key.

Screen 3 (HU19): Specifying Output Device Type

```
                     DUMP OUTPUT DEVICE INFORMATION          HU19

          ENTER OUTPUT DEVICE TYPE (TAPE, DSKT OR SPECIFIC DISK TYPE): DSKT

          *********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

Here, we indicate that output is to be written to diskette. (In 12.3.3, we use these same diskettes as input for the restore operation.) Since no help is needed, press the XMIT key.

When you dump files to diskette, the diskette must be prepped as a data set label diskette with a block size of 256 or 128 bytes. You should also use the default FDATA=Y prep option to allocate the entire diskette as one file called DATA.

If you want to use a file name other than DATA, you must prep the diskette using the FDATA=N prep option. Then you must allocate this MIRAM file in blocks using the interactive services ALLOCATE command. To determine the number of blocks required for the diskette file, use the following formula:

number of cylinders being dumped from disk × number of surfaces (tracks) per disk × track capacity (bytes per track) = total number of bytes being dumped from disk

total number of bytes being dumped from disk ÷ diskette block size (bytes per block) = number of diskette blocks required

To determine the number of cylinders being dumped from disk, use an interactive services VTOC command. The number of surfaces per disk and the track capacity depend on the disk type. The diskette block size must be either 256 or 128 bytes. For these variable values, see the *Consolidated Data Management Programming Guide* (UP-9978).

Screen 4 (HU27): Specifying Input Volume Serial Number

```
                DUMP INPUT VOLUME INFORMATION          HU27

        ENTER THE INPUT VOLUME SERIAL NUMBER: REL120

        *********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we identify the volume serial number of the disk from which the files are to be dumped as REL120.

Screen 5 (HU21): Specifying Output Volume Serial Number(s)

```
                          DUMP OUTPUT VOLUME INFORMATION        HU21

      ENTER OUTPUT VOLUME SERIAL NUMBER(S) OR IDENTITY OF GROUP OF VOLUMES

            D07098          ------    ------    ------
            ------    ------    ------    ------    ------
            ------    ------    ------    ------    ------
            ------    ------    ------    ------    ------

      ENTER Y IF THIS IS THE IDENTITY OF A GROUP OF VOLUMES: N

      ENTER FILE NAME: DATA

      ********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP  ********
```

On this screen, we identify where the data is to be dumped. We can do this by listing volume serial numbers for all output volumes or by referencing them all by one group name (or identity). To specify volume serial numbers, list them in order on the lines provided. Up to 16 volume serial numbers can be listed. If you have more than 16 output volumes, you must use a group name. A group name can contain up to six characters and the first must be a letter.

In our example, we enter the volume serial number of the output diskette, D07098. Then we press the TAB key to get to the next question. We are using the volume serial number for the diskette, not a group name. Therefore, we keep the default value (N) and press the TAB key to the next field. Here, we enter the file name DATA. We do not need help, so we press the XMIT key to transmit these entries.

Note: *The output file specified for the ENTER FILE NAME entry must be allocated prior to the execution of the dump operation.*

Screen 6 (HU22): File Options

```
                                DUMP                              HU22

        WAIT FOR UNLOCK                                    WAIT=YES
        DUMP ALL FILES                                     ALL=NO

        ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we accept the default values, YES for WAIT FOR UNLOCK (waiting for our file to be available) and NO for DUMP ALL FILES. Press the XMIT key.

Screen 7 (HU23): Specifying Input File Names

```
                          DUMP                              HU23
ENTER
   .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP OF FILES

FILE _ _   = TESTIN
FILE _ _   = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
FILE _ _   = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
ARE MORE FILES TO BE ENTERED?  NO
*********FUNCTION KEYS: F13=HELP, F14=EXIT HELP *********
```

On this screen, we enter our input file name TESTIN. Since this is the only file being dumped, we accept the default NO for more files to be entered. Press the XMIT key. Processing begins. After our job is finished, a summary report is produced.

Note: *When dumping all files, only active files listed in the VTOC are dumped.*

Performing a Multidisk Volume Dump Operation

Now, let's perform a multidisk volume dump to tape operation.

Screen 1: (HU00BI01) Informational Message

```
                                                          HU00BI01

     A CONVERSATIONAL JOB (HU$DMP) TO DUMP FILES FROM DISK(S) WILL BE
     INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM MODE FOR THE JOB
     TO BE SCHEDULED. IF YOU ENTERED HARDWARE UTILITIES THROUGH THE
     HU COMMAND YOU WILL BE IN SYSTEM MODE AFTER TRANSMITTING.
     IF YOU ENTERED THROUGH THE MENU COMMAND YOU ARE RESPONSIBLE FOR
     GOING INTO SYSTEM MODE.
                    *****  TRANSMIT TO CONTINUE  *****
```

This is an informational screen. Read it and press the XMIT key to continue.

Screen 2: (HU18) Defining Input Device

```
                      DUMP INPUT DEVICE INFORMATION          HU18

     ENTER INPUT DISK DEVICE TYPE: 8419

     ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On Screen 2, we indicate that our disk device type is an 8419. No help is required, so press the XMIT key.

Screen 3 (HU19): Specifying Output Device Type

```
                     DUMP OUTPUT DEVICE INFORMATION          HU19

     ENTER OUTPUT DEVICE TYPE (TAPE, DSKT OR SPECIFIC DISK TYPE): TAPE

     *********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

Here, we indicate that output is to be written to tape. Since no help is needed, press the XMIT key.

Screen 4 (HU26): Specifying Input Volume Information

```
                DUMP INPUT VOLUME INFORMATION                HU26

ENTER DISK IDENTIFIERS (D01-D16) AND INPUT VOLUME SERIAL NUMBERS
(DISK IDENTIFIERS ARE ONLY REQUIRED FOR MULTIPLE INPUT VOLUMES):

 D01 : D07096        D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

 D02 : D07097        D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

 D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

 D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _


     ********       FUNCTION KEYS:  F13=HELP, F14=EXIT HELP       ********
```

On this screen, we specify the disk identifiers and the volume serial numbers of the disks from which the files are to be dumped.

Screen 5 (HU20): Specifying Output Volume Serial Number(s)

```
                DUMP OUTPUT VOLUME INFORMATION        HU20

          ENTER OUTPUT VOLUME SERIAL NUMBER(S):

           T07098
           _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _
           _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _
           _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _
           _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _   _ _ _ _ _ _

          ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we identify where the data is to be dumped by listing volume serial numbers in order on the lines provided. In our example, we enter the volume serial number of the output tape, T07098. We do not need help, so we press the XMIT key.

Screen 6 (HU28): File Options

```
                                DUMP                    HU28

     WAIT FOR UNLOCK                                    WAIT=YES

     ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we accept the default value YES for WAIT FOR UNLOCK (waiting for our file to be available). Press the XMIT key.

Screen 7 (HU29): Specifying Input File Names

```
                                DUMP                          HU29
ENTER

DNN WHERE NN IS THE IDENTIFIER OF THE DISK FROM WHICH THE FILES
    ARE TO BE DUMPED

          .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP OF FILES

             =FILENAME

D01   FILE_ _=FILEA _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

D02   FILE_ _=FILEB _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

D_ _  FILE_ _=_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

ARE MORE FILES TO BE ENTERED?  NO

    ********       FUNCTION KEYS:     F13=HELP, F14=EXIT HELP        *********
```

On this screen, we enter our input file names. Since these are the only files being dumped, we accept the default NO for more files to be entered. Press the XMIT key. Processing begins. After our job is finished, a summary report is produced.

12.3.3. Performing a Restore Operation

After you have successfully completed the dump operation, you must use the restore operation to use the disk on the system. You execute the restore operation by keying in 2 from the selections on the menu screen. Up to seven additional screens can be displayed. These screens are

- Screen 1 (HU00BI02) is an informational screen telling you that the interactive job you selected has been initiated.
- Screen 2 (HU02) asks you to specify the input device type.
- Screen 3 (HU03 or HU04) asks you for your input volume serial numbers and the MIRAM file name (if required).
- Screen 4 (HU05) asks you for the output device type, which must be the same device type as the original file that was dumped.

- Screen 5 (HU30 or HU31) asks you for the output volume serial number(s). Screen HU31 also requests a disk identifier for each volume to associate it with the files to be restored to it.
- Screen 6 (HU06, HU07, or HU32) asks you to specify the file options.
- Screen 7 (HU08 for single-disk input, HU33 for multidisk input, or HU34 if you are restoring all files). Screen HU08 or Screen HU33 asks you for the file names if you are restoring individual files; Screen HU33 also asks you for the identifier of the disk to which the file is to be restored. Screen HU34 requests the type of file allocation if you are restoring all files.

Performing a Single-Volume Disk Restore Operation

Now, let's do a typical single-volume restore operation. Remember, in the first dump example, we used diskettes as our output medium. These same diskettes are now used as our input medium for our restore operation.

Screen 1 (HU00BI02): Informational Message

```
                                                          HU00BI02
A CONVERSATIONAL JOB (HU$RST) TO RESTORE FILES TO DISK(S) WILL BE
INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM MODE FOR THE JOB
TO BE SCHEDULED. IF YOU ENTERED HARDWARE UTILITIES THROUGH THE
HU COMMAND YOU WILL BE IN SYSTEM MODE AFTER TRANSMITTING.
IF YOU ENTERED THROUGH THE MENU COMMAND YOU ARE RESPONSIBLE FOR
GOING INTO SYSTEM MODE.
                  *****  TRANSMIT TO CONTINUE  *****
```

This is an informational message screen. Read it and press the XMIT key to continue.

Screen 2 (HU02): Input Device Type

```
                  RESTORE INPUT DEVICE INFORMATION          HU02

ENTER INPUT DEVICE TYPE (TAPE,DSKT OR SPECIFIC DISK TYPE): DSKT


    ********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP  ********
```

The device type is diskette. Help is not required, so press the XMIT key.

Screen 3 (HU04): Input Volume Serial Number

```
                RESTORE INPUT VOLUME INFORMATION          HU04

ENTER INPUT VOLUME SERIAL NUMBER(S) OR IDENTITY OF GROUP OF VOLUMES:
     D07098        ------   ------   ------
     ------   ------   ------   ------
     ------   ------   ------   ------
     ------   ------   ------   ------
ENTER Y IF THIS IS THE IDENTITY OF A GROUP OF VOLUMES: N
ENTER FILE NAME: DATA
   ********    FUNCTION KEYS:  F13=HELP, F14=EXIT HELP    ********
```

On this screen, we identify the input device. We can do this by listing volume serial numbers for all input volumes or by referencing them all by one group name (or identity). To specify volume serial numbers, you list them in order on the lines provided. Up to 16 volume serial numbers can be listed. If you have more than 16 input volumes, you must use a group name. A group name can contain up to six characters and the first must be a letter.

In our example, we enter the volume serial number of the input diskette, D07098. Then we press the TAB key to get to the next question. We are using the volume serial number for the diskette, not a group name. Therefore, we keep the default value (N) and press the TAB key to the next field. Here, we enter the file name DATA. We do not need help, so we press the XMIT key to transmit these entries.

Screen 4 (HU05): Output Device Type

```
                RESTORE OUTPUT DEVICE INFORMATION         HU05

ENTER OUTPUT DISK DEVICE TYPE: 8419

********    FUNCTION KEYS: F13=HELP, F14=EXIT HELP    *********
```

We indicate our device type is an 8419 disk. Press the XMIT key.

Screen 5 (HU30): Output Serial Number

```
                RESTORE OUTPUT VOLUME INFORMATION         HU30

ENTER THE OUTPUT VOLUME SERIAL NUMBER: REL12Ø

 ********    FUNCTION KEYS: F13=HELP, F14=EXIT HELP   ********
```

On this screen, we specify the volume serial number of the disk to which the files are to be restored.

Screen 6 (HU06): File Options

```
                                    RESTORE          HU06

     WAIT FOR UNLOCK                                          WAIT=YES
     RESTORE ALL FILES                                        ALL=NO
     UNEXPIRED FILE CHECKING                                  UNXF=YES
     STARTING FROM VOLUME OTHER THAN ONE                      ANY=NO

     ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we take the default values for all file options and press the XMIT key. Our result is

- DMPRST will not start processing until the file becomes available.
- Only the files specified are restored.
- File expiration date is checked on the output volume.
- Restoration starts with the first volume.

Note: If you specify ALL on Screen 6 (HU06), Screen 7 is HU34, otherwise Screen 7 is HU08. Screen HU34 is illustrated following HU08.

Screen 7 (HU08): Specifying Output File Names (Selected Files)

```
                    RESTORE                HU08

ENTER
    .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP
    =FILENAME,ALLOCATION,NEWNAME

FILE_ _ = TESTIN

          - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FILE_ _ = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _'_ _ _ '
          - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FILE_ _ = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _'_ _ _ '
          - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ARE MORE FILES TO BE ENTERED? NO

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we specify our file name as TESTIN. If you recall, this is the file dumped to diskette during the dump operation. Since this is the only file being dumped, move the cursor to the bottom of the screen and press the XMIT key.

After pressing XMIT, the restore operation is executed and, upon completion, a summary report is produced indicating that the job was terminated.

Note: When restoring files from tape, diskette, or sequential files on disk, the restore of files with the same prefix will terminate when the first file without that prefix is encountered on the input medium or at the end of the input file. The next file statement, if any, is then processed.

Screen 7 (HU34): Restoring All Files

```
                              RESTORE                                HU34

        SELECT TYPE SPACE ALLOCATION TO BE USED FOR ALL FILES: _ _ _

        ABSolute - SAME SPACE FILE OCCUPIED WHEN DUMPED
        RELocate - RELOCATE IF POSSIBLE
        LOGical -  SUFFICIENT SPACE TO ALLOCATE SPACE ALREADY
                     ASSIGNED TO LOGICAL PARTITIONS
        PREallocated - SPACE IS ALREADY ALLOCATED ON THE DISK
        BLANK    -  USE FIRST SUCCESSFUL OF THE FIRST THREE METHODS
                      OF ALLOCATION

      ********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP  ********
```

Screen HU34 asks you to select a global allocation parameter.

Performing a Multidisk Volume Restore Operation

Now, let's do a typical multidisk volume restore from tape operation.

Screen 1 (HU00BI02): Informational Message

```
                                                          HU00BI02
    A CONVERSATIONAL JOB (HU$RST) TO RESTORE FILES TO DISK(S) WILL BE
    INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM MODE FOR THE JOB
    TO BE SCHEDULED. IF YOU ENTERED HARDWARE UTILITIES THROUGH THE
    HU COMMAND YOU WILL BE IN SYSTEM MODE AFTER TRANSMITTING.
    IF YOU ENTERED THROUGH THE MENU COMMAND YOU ARE RESPONSIBLE FOR
    GOING INTO SYSTEM MODE.
                    *****  TRANSMIT TO CONTINUE  *****
```

This is an informational message screen. Read it and press XMIT key to continue.

Screen 2 (HU02): Input Device Type

```
                    RESTORE INPUT DEVICE INFORMATION          HU02

        ENTER INPUT DEVICE TYPE (TAPE,DSKT OR SPECIFIC DISK TYPE): TAPE

          ********   FUNCTION KEYS: F13=HELP, F14=EXIT HELP   ********
```

The device type is tape. Help is not required, so press the XMIT key.

Screen 3 (HU03): Input Volume Serial Number

```
                  RESTORE INPUT VOLUME INFORMATION          HU03

      ENTER INPUT VOLUME SERIAL NUMBER(S) OR IDENTITY OF GROUP OF VOLUMES:

        T07098_ - -    - - - - - -    - - - - - -    - - - - - -
        - - - - - -    - - - - - -    - - - - - -    - - - - - -
        - - - - - -    - - - - - -    - - - - - -    - - - - - -
        - - - - - -    - - - - - -    - - - - - -    - - - - - -

           ********   FUNCTION KEYS:  F13=HELP, F14=EXIT HELP    ********
```

On this screen, we identify the input device by listing volume serial numbers for all input volumes.

Screen 4 (HU05): Output Device Type

```
                   RESTORE OUTPUT DEVICE INFORMATION          HU05

         ENTER OUTPUT DISK DEVICE TYPE: 8494

         ********   FUNCTION KEYS: F13=HELP, F14=EXIT HELP   *********
```

We indicate our device type is an 8494 disk. Press the XMIT key.

Screen 5 (HU31): Output Serial Number

```
                RESTORE OUTPUT VOLUME INFORMATION              HU31

ENTER DISK IDENTIFIERS (D01-D16) AND OUTPUT VOLUME SERIAL NUMBER(S)
(DISK IDENTIFIERS ARE REQUIRED WHEN RESTORING FROM A TAPE CREATED
FROM MULTIPLE INPUT VOLUMES; THEY SHOULD NOT BE USED WHEN RESTORING
FROM A TAPE CREATED FROM A SINGLE DISK INPUT):

  D01 : D07090        D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

  D02 : D07099        D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

  D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _

  D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _   D_ _: _ _ _ _ _ _


    ********    FUNCTION KEYS:  F13=HELP, F14=EXIT HELP     ********
```

On this screen, we specify the disk identifier(s) and volume serial number(s) of the disk(s) to which the files are to be restored.

Screen 6 (HU32): File Options

```
                                RESTORE                        HU32

  WAIT FOR UNLOCK                                          WAIT=YES

  UNEXPIRED FILE CHECKING                                  UNXF=YES


  ******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we take the default values for all file options and press the XMIT key.

Our result is

- DMPRST will not start processing until the file becomes available.
- File expiration date is checked on the output volume.

Screen 7 (HU33): Specifying Output File Names

```
                           RESTORE                          HU33
ENTER
DNN WHERE NN IS THE IDENTIFIER OF THE DISK TO WHICH THE FILES
    ARE TO BE RESTORED

       .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP

         =FILENAME,ALLOCATION,NEWNAME

D01  FILE_ _=FILEA _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _,_ _ _,
             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

D02  FILE_ _=FILEB _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _,_ _ _,
             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

D_ _ FILE_ _=_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _,_ _ _,
             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

ARE MORE FILES TO BE ENTERED?  NO

*********FUNCTION KEYS: F13=HELP, F14=EXIT HELP *********
```

On this screen, we specified the output files. Since they are the only files being dumped, move the cursor to the bottom of the screen and press the XMIT key.

After pressing XMIT, the restore operation is executed and, upon completion, a summary report is produced indicating that the job was terminated.

12.3.4. Performing a List Operation

The list function consists of the following three screens:

- Screen 1 (HU00BIL) is an informational screen telling you that the interactive job you selected has been initiated.
- Screen 2 (HU35) asks you for the input device type.
- Screen 3 (HU36 or HU37) asks you for your input volume serial number(s) or group identity and the file name.

Screen 1 (HU00BIL): Informational Message

```
                                                          HU00BI02
A CONVERSATIONAL JOB (HU$LST) TO LIST THE NAMES OF FILES DUMPED
TO A SEQUENTIAL DISK FILE, A TAPE, OR A DISKETTE FROM DISK(S)
WILL BE INITIATED IN YOUR BEHALF.  YOU MUST BE IN SYSTEM MODE
FOR THE JOB TO BE SCHEDULED.  IF YOU ENTERED HARDWARE UTILITIES
THROUGH THE HU COMMAND, YOU WILL BE IN SYSTEM MODE AFTER TRANS-
MITTING.  IF YOU ENTERED THROUGH THE MENU COMMAND YOU ARE
RESPONSIBLE FOR GOING INTO SYSTEM MODE.
                  *****  TRANSMIT TO CONTINUE  *****
```

This is an informational message screen. Read it and press XMIT key to continue.

Screen 2 (HU35): List Input Device Information

```
                    LIST INPUT DEVICE INFORMATION                HU35

        ENTER INPUT DEVICE TYPE (TAPE, DSKT OR SPECIFIC DISK TYPE): _ _ _ _


           *******  FUNCTION KEYS:  F13=HELP,  F14=EXIT HELP  *******
```

Enter the type of input device. If you specify tape, Screen 3 is HU36; otherwise, Screen 3 is HU37.

Screen 3 (HU36): List Input Volume Information (Tape)

```
                    LIST INPUT VOLUME INFORMATION                HU36

        ENTER INPUT VOLUME SERIAL NUMBER(S)

          - - - - - -    - - - - - -    - - - - - -    - - - - - -
          - - - - - -    - - - - - -    - - - - - -    - - - - - -
          - - - - - -    - - - - - -    - - - - - -    - - - - - -
          - - - - - -    - - - - - -    - - - - - -    - - - - - -

        ********  FUNCTION KEYS: F13=HELP, F14=EXIT HELP  ********
```

For tape input, enter the volume serial number(s).

Screen 3 (HU37): List Input Volume Information (Diskette or Disk)

```
                LIST INPUT VOLUME INFORMATION                    HU37

ENTER INPUT VOLUME SERIAL NUMBER(S) OR IDENTITY OF GROUP OF VOLUMES:

   ------    ------    ------    ------
   ------    ------    ------    ------
   ------    ------    ------    ------
   ------    ------    ------    ------

ENTER Y IF THIS IS THE IDENTITY OF A GROUP OF VOLUMES: N

ENTER FILE NAME: ----------------------------

    ********     FUNCTION KEYS:  F13=HELP, F14=EXIT HELP    ********
```

12.4. Executing DMPRST in Volume Mode (Batch Environment)

As shown in Table 12-1, you can execute DMPRST in either volume or file mode. By executing DMPRST in volume mode, you can back up an entire disk volume or a truncated portion of that volume. On the other hand, if you want to create backups of individual files on your disk, you can run the DMPRST routine in file mode. Both modes permit you to choose a disk, tape, or data set label diskette as your backup medium. In 12.4, we will discuss volume mode while in 12.5 we will discuss file mode.

Since a printer is not automatically assigned in this operation as is the case in the interactive environment, you should assign one in your control stream. DMPRST will then give a listing of the // PARAM statements and all error messages. If no printer is assigned, only critical error messages will be displayed on your workstation screen.

Table 12-2 summarizes the // PARAM statements along with the appropriate LFD names and the type of operation that will be performed. A complete description of these statements is provided in 12.4.1 through 12.4.5.

Table 12-2. Volume Mode // PARAM Statements

Type of Operation	// PARAM Statement	LFD Name

Copy disk to disk	// PARAM IN=DISC
	// PARAM OUT=DISC
	// PARAM END=LAST
		DISCIN
		DISCOT

Dump disk to MIRAM disk file	// PARAM IN=DISC
	// PARAM OUT=SEQD
	// PARAM END=LAST
		DISCIN
		SEQDOT,,INIT

Dump disk to tape	// PARAM IN=DISC
	// PARAM OUT=TAPE
	// PARAM END=LAST
		DISCIN
		TAPEOT

Dump disk to streaming tape	// PARAM IN=DISC
	// PARAM OUT=TAPE,SLOW
	// PARAM END=LAST
		DISCIN
		TAPEOT

Dump disk to diskette	// PARAM IN=DISC
	// PARAM OUT=SEQD
	// PARAM END=LAST
		DISCIN
		SEQDOT,,INIT

Restore tape to disk	// PARAM IN=TAPE // PARAM OUT=DISC	TAPEIN DISCOT

Restore streaming tape to disk	// PARAM IN=TAPE,SLOW
	// PARAM OUT=DISC
	// PARAM END=LAST
		TAPEIN
		DISCOT

Restore diskette to disk	// PARAM IN=SEQD // PARAM OUT=DISC	SEQDIN DISCOT
Restore MIRAM disk file to disk	// PARAM IN=SEQD // PARAM OUT=DISC	SEQDIN DISCOT
Copy tape to tape	// PARAM IN=TAPE // PARAM OUT=TAPE	TAPEIN TAPEOT
Copy diskette to diskette	// PARAM IN=SEQD // PARAM OUT=SEQD	SEQDIN SEQDOT
Check file expiration date	// PARAM NOEXPCK	
Restart processing for dump/ restore to tape or diskette	// PARAM RESTART	

12.4.1. Performing a Disk Copy Operation in Volume Mode

During a disk copy operation, you are copying the contents of a disk to another disk of the same type. Since the input and the output in a disk copy operation are always a disk, specify // PARAM IN=DISC as your input parameter and // PARAM OUT=DISC as your output parameter. In addition, specify DISCIN on the input // LFD statement and DISCOT on the output // LFD statement as file names.

You can control the termination of the copy operation by specifying the // PARAM END statement. The // PARAM END=*ccc* statement terminates the copy operation after a specific cylinder (*ccc*) is copied. By specifying // PARAM END=LAST, you terminate the copy operation after the last allocated cylinder has been copied. You should specify a // PARAM END statement. If you omit it, the entire contents of the input disk is copied to the output disk, even if only a few cylinders contain user data.

Note: *If you use the // PARAM END=*ccc *statement,* ccc *must be specified in decimal.*

The following two examples are typical job streams used to perform a disk copy operation. In Example 1, the disk volume DISK01 is copied to the backup disk volume DISK02. Also, the copy operation is terminated after the last allocated cylinder has been copied. In Example 2, the resident volume OS3RES is copied to the backup volume RESBAK. The copy operation is terminated after the last allocated cylinder has been copied. Note that, even though a different disk type is used in each example, the // PARAM IN, // PARAM OUT, and // LFD names are always the same.

Example 1: Disk Volume Copy Operation Using 8419 Disks

```
// JOB DSKCOPY1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 51 // VOL DISK02 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM END=LAST
/&
// FIN
```

Example 2: Disk Volume Copy Operation Using 8417 Disks

```
// JOB DSKCOPY2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL OS3RES // LFD DISCIN
// DVC 51 // VOL RESBAK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM END=LAST
/&
// FIN
```

12.4.2. Performing a Dump Operation in Volume Mode

The dump operation enables you to dump a disk or portion of a disk to tape, diskette, or a MIRAM file on a disk of a different type from the original. You may use any tape, data set label diskette, or a MIRAM disk file as a backup in a dump operation. However, the parameters specified in your control stream depend upon the devices being used.

In a dump operation, the input is always from disk. Therefore, specify // PARAM IN=DISC as your input parameter and DISCIN as the file name on the // LFD job control statement of the input disk.

If you only want to dump a portion of a disk, include a // PARAM END=*ccc* statement in your job control stream. The // PARAM END=*ccc* statement terminates the dump operation after a specific cylinder (*ccc*) has been dumped. If you specify // PARAM END=LAST, the dump operation terminates after the last allocated cylinder has been dumped.

If you omit the // PARAM END parameter, the entire contents of the input disk is dumped to the output (including test pattern data written by the disk prep routine), even if only a few cylinders include user data.

The output in a dump operation may be a tape, data set label diskette, or a MIRAM file on a disk of a different type than the original. Your output parameter, however, depends on your output medium. Remember, the device you select for output must first be prepped.

Dumping to Tape in Volume Mode

When using magnetic tape as the output, specify // PARAM OUT=TAPE as your output parameter and TAPEOT as the file name on your tape // LFD job control statement. These same parameter specifications are applicable if your output tape is a streaming tape operating in default (100 ips) mode. If, however, you want the streaming tape to operate in slow (25 ips) mode, then you must include the SLOW parameter in the // PARAM statement (// PARAM OUT=TAPE,SLOW). Table 12-2 (see 12.4) lists the formats for the // PARAM statement when executing DMPRST in volume mode (batch). When dumping to a streaming tape, it is recommended that your system be dedicated to dump restore to ensure the streaming mode operation.

When using more than one tape as output, you can assign them in your control stream in one of two ways. One way is to use a single device assignment set (DVC-LFD sequence) having a file name of TAPEOT. Include one // VOL statement listing the volume serial numbers of all output tapes (see Example 1).

Another way is to supply a separate device assignment set for each output tape. Here, a separate device is assigned for each tape volume. Again, the file name on your tape // LFD statement is TAPEOT. However, 01 through 99 are attached to each tape file name, following the first one (i.e., // LFD TAPEOT01) (see Example 2).

The first record written to your magnetic tape is called the control record. The control record contains the following information in the order shown:

Bytes	Contents
0-1	Control record ID X'COCO'
2-5	System time (packed decimal)
6-9	System data (packed decimal)
10-15	Volume serial number of dumped disk pack
16-17	Unused
18-21	Device type of dumped disk pack
22	Device type of tape
23	Tape channel number
24-25	Unused
26	X'F' - shows file processing
27-30	Volume table of contents of dumped disk if file processing
31-79	Unused

The following two examples show multivolume dump operations using a magnetic tape:

Example 1: Multivolume Dump Using a Single Device Assignment Set for All Output Tapes

```
// JOB TAPDUMP1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL OS3RES // LFD DISCIN
// DVC 90 // VOL SPO366,SPO367,SPO368 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
/&
// FIN
```

The disk volume OS3RES is dumped to tape volumes SPO366, SPO367, and SPO368. Here, a single device assignment set (DVC-LFD sequence) assigns all three output tapes. Each tape volume is mounted separately on the same tape drive.

Example 2: Multivolume Dump Using a Separate Device Assignment Set for Each Output Tape

```
// JOB TAPDUMP2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL OS3RES // LFD DISCIN
// DVC 90 // VOL SPO366 // LFD TAPEOT
// DVC 91 // VOL SPO367 // LFD TAPEOT01
// DVC 92 // VOL SPO368 // LFD TAPEOT02
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
/&
// FIN
```

The disk volume OS3RES is dumped to tape volumes SPO366, SPO367, and SPO368. Here, each tape volume is assigned via an individual device assignment set and the tape volumes are mounted simultaneously on separate tape drives.

Dumping to Diskettes in Volume Mode

If you are using a diskette as output, specify // PARAM OUT=SEQD as your output parameter. DMPRST stores its control information and the data from the original disk in a MIRAM file on the backup diskettes.

When you dump files to a diskette, must be prepped as a data set label diskette with a block size of 256 or 128 bytes. You should also use the default FDATA=Y prep option to allocate the entire diskette as one file called DATA. Remember to use the name DATA in the LBL statement of your DMPRST job control stream.

If you want to use a file name other than DATA, you must prep the diskette using the FDATA=N prep option. Then you must allocate this MIRAM file in blocks. You do this through the interactive services ALLOCATE command or by using the // EXT job control statement with the BLK parameter. To determine the number of blocks required in the diskette file, use the following formula:

number of cylinders being dumped from disk **X** number of surfaces (tracks) per disk **X** track capacity (bytes per track) **=** total number of bytes being dumped from disk

total number of bytes being dumped from disk **÷** diskette block size (bytes per block) **=** number of diskette blocks required

To determine the number of cylinders being dumped from disk, use an interactive services VTOC command. The number of surfaces per disk and the track capacity depend on the disk type. The diskette block size must be 256 or 128 bytes. For these values, see the *Consolidated Data Management Programming Guide* (UP-9978).

Since most dump operations require multiple diskettes, it's much easier to use the default name DATA rather than change the name on each diskette.

The file name on your diskette // LFD statements is SEQDOT,,INIT. Unlike the file name specification for tape, only a single device assignment set (DVC-LFD sequence) may be provided, even if you are using more than one diskette. You include the INIT parameter in the LFD statement to initialize the diskette so DMPRST can dump to it. Remember, however, that you must not include the INIT parameter when performing the restore operation from that diskette.

All volumes must be listed on // VOL statements unless // VOL SCRATCH is used. For more information on the // VOL SCRATCH statement, see the *Job Control Language Programming Guide* (UP-9986).

The following example shows a typical job stream for a multivolume dump operation using diskettes:

```
// JOB DSKTDMP1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL OS3RES // LFD DISCIN
// DVC 130 // VOL DSKT01,DSKT02,DSKT03,DSKT04
// LBL DATA // LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
/&
// FIN
```

The disk volume OS3RES is dumped to a MIRAM file named DATA on diskette volumes DSKT01, DSKT02, DSKT03, and DSKT04. The MIRAM file DATA was allocated to diskettes as part of the prep procedure performed prior to the job run. Since a MIRAM file is required when dumping to diskette, the diskette must be prepped in the data set label mode.

Dumping to a MIRAM Disk File in Volume Mode

When you dump to a MIRAM disk file, your output disk may be a different type from the original, e.g., dumping from an 8417 disk to an 8419 disk. Before performing the dump operation, you must allocate a MIRAM file on the output disk for use by DMPRST.

Note: *The 8417 fixed-head disk can only be dumped in file mode.*

If one output disk is sufficient, allocate the MIRAM file by including the necessary job control statements in your DMPRST job control stream. However, for more than one output disk, you should allocate the MIRAM file in a separate job before running your DMPRST job. For more information on allocating files, see the *Job Control Language Programming Guide* (UP-9986).

Specify // PARAM OUT=SEQD as your output parameter and the file name on your output disk // LFD job control statement as SEQDOT,,INIT. As in the case of a diskette, only one device assignment set (DVC-LFD sequence) may be used.

You include the INIT parameter in the LFD statement to initialize the disk so DMPRST can dump to it. Remember, however, that you must not include the INIT parameter when performing the restore operation from that disk.

The following examples show dump operations using disks.

Example 1: Single-Volume Dump to Disk

```
// JOB DSKDUMP1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 51
// VOL BKUP01
// EXT MI,C,,CYL,800
// LBL PAYBACKUP
// LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
// PARAM END=LAST
/&
// FIN
```

The 8417 disk volume, DISK01, is dumped to the PAYBACKUP file on the 8419 disk volume, BKUP01. The dumped contents of the original volume is stored on the backup volume in a MIRAM file. Therefore, a MIRAM file is allocated for the PAYBACKUP file. Since only one backup volume is used, this file is allocated by specifying the necessary job control statements in the DMPRST job itself.

Example 2: Multivolume Dump to Disk

```
// JOB DSKDUMP2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL OS3RES // LFD DISCIN
// DVC 51 // VOL BKRES1,BKRES2
// LBL PACKSAVE // LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
/&
// FIN
```

In this example, an OS/3 resident volume is dumped to two 8419 disk volumes, BKRES1 and BKRES2. Here, as in Example 1, the dumped contents of the original disk are stored in a MIRAM file on the backup volume. Therefore, a MIRAM file named PACKSAVE is allocated on each backup volume. Since two backup volumes are used, the MIRAM files are allocated in separate jobs prior to the execution of the DMPRST job. Note that a single device assignment set is used for the backup volumes. (Only one disk drive is permitted; multimount disks are not allowed.)

12.4.3. Performing a Restore Operation in Volume Mode

When you perform a dump operation, the backup volume that is created is stored in a MIRAM file. Therefore, you cannot use the dumped backup volume on your system if your original data is lost. Before actually using any backup data on your system, you must restore it to the original disk or a disk of the same type as the original.

In a restore operation, the output is always a disk. Therefore, you should always specify // PARAM OUT=DISC as your output parameter and DISCOT as the file name on your output disk // LFD job control statement.

The input in a restore operation may be tape, data set label diskette, or a MIRAM disk file of a different type from the original. However, the input parameter used depends on your input medium.

Restoring from Tape in Volume Mode

When using magnetic tape as the input, specify // PARAM IN=TAPE as your input parameter and TAPEIN as the file name on your tape // LFD job control statement. These same parameter specifications are applicable if your input tape is a streaming tape operating in the default (100 ips) mode. If, however, you want the streaming tape to operate in the slow (25 ips) mode, then you must include the SLOW parameter in the // PARAM statement (// PARAM IN=TAPE,SLOW). Table 12-2 lists the formats for the // PARAM statement when executing DMPRST in volume mode (batch). It is recommended, when restoring from a streaming tape, that your system be dedicated to dump restore to ensure the streaming mode operation.

When using more than one tape as input, you can assign them in your control stream in one of two ways. One way is to use a single device assignment set (DVC-LFD sequence) having a file name of TAPEIN. Included in this set are // VOL statements listing the volume serial numbers of the restore operation input tapes.

Another way is to supply a separate device assignment set for each tape in your job. Here, a separate device is assigned for each tape volume. Again, the file name (// LFD) is TAPEIN. However, the numbers 01 through 99 are attached to each tape file name, following the first one (i.e., // LFD TAPEIN01).

Note: *If you perform a dump operation using separate device assignment sets and then restore using a single device assignment set, you'll receive an error message from data management. The method of assigning tapes used in the dump operation should also be used in the related restore operation.*

The following two examples show restore operations using magnetic tape:

Example 1: Multivolume Restore Using a Single Device Assignment Set for Tape

```
// JOB TAPERSTR1
// DVC 20 // LFD PRNTR
// DVC 90 // VOL SPO366,SPO367,SPO368 // LFD TAPEIN
// DVC 50 // VOL OS3RES // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
/&
// FIN
```

The tape volumes SPO366, SPO367, and SPO368, which contain the backup volume OS3RES, are restored to the original disk. Here, a single device assignment set assigns the tapes in this control stream. Each tape volume is mounted separately on the same tape drive.

Example 2: Multivolume Restore Using Separate Device Assignment Sets for Tape

```
// JOB TAPERSTR2
// DVC 20 // LFD PRNTR
// DVC 90 // VOL SPO366 // LFD TAPEIN
// DVC 91 // VOL SPO367 // LFD TAPEIN01
// DVC 92 // VOL SPO368 // LFD TAPEIN02
// DVC 50 // VOL OS3RES // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
/&
// FIN
```

The tape volumes SPO366, SPO367, and SPO368, which contain the backup volume OS3RES, are restored to the original disk. Here, each tape volume is assigned via an individual device assignment set. The tape volumes are mounted simultaneously on separate tape drives.

Restoring from Diskettes in Volume Mode

If your input is a diskette, specify // PARAM IN=SEQD as your input parameter, and SEQDIN as the file name on your diskette // LFD job control statement. You may only specify one device assignment set (DVC-LFD sequence) when multivolume diskettes are used.

The following example shows restore operations using diskettes:

```
// JOB DSKTRST
// DVC 20 // LFD PRNTR
// DVC 130 // VOL DSKT01,DSKT02,DSKT03,DSKT04
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL OS3RES // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
/&
// FIN
```

The diskettes created by the dump operation in 12.4.2 are restored to the original disk volume, OS3RES. Since the data on the diskettes is stored on a MIRAM file named DATA, the LBL name DATA must be included in the diskette device assignment set.

Restoring from a MIRAM Disk File in Volume Mode

Since the input in a restore operation is from a MIRAM disk file, specify // PARAM IN=SEQD as your input parameter and SEQDIN as the file name on your input disk // LFD job control statement.

Include the name of the input disk MIRAM file in your control stream. This is the same name that was assigned for the related dump operation. Remember, only a single device assignment set may be used to assign your input disk volumes.

The following two examples show restore operations using disk;

Example 1: Single-Volume Restore from Disk

```
// JOB DISKRST1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL BKUP01
// LBL PAYBACKUP // LFD SEQDIN
// DVC 51 // VOL DISK01 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
/&
// FIN
```

This backup disk volume, BKUP01, which was created by the dump operation shown in Example 1 in "Dumping to a MIRAM File in Volume Mode" in 12.4.2, is restored to the original disk volume, DISK01. Since the backup data is stored in a MIRAM file named PAYBACKUP, the LBL name PAYBACKUP is included in the disk device assignment set.

Example 2: Multivolume Restore from Disk

```
// JOB DISKRST2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL BKRES1,BKRES2
// LBL PACKSAVE // LFD SEQDIN
// DVC 51 // VOL D00001 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
/&
// FIN
```

The backup disk volumes, BKRES1 and BKRES2, which were created by the dump operation shown in Example 2 in "Dumping to a MIRAM Disk File in Volume Mode" in 12.4.2, are restored to the disk volume D00001. The disk-resident volume cannot be the output disk in a volume restore operation. Therefore, the restore is made to another disk of the same type (D00001) as the original. When the restore is complete, the volume serial number (VSN) on the output pack (D00001) will be OS3RES.

The backup data is stored in a MIRAM file called PACKSAVE on the backup volumes. Therefore, the LBL name PACKSAVE must be included in the disk device assignment set. Note that a single device assignment set is used for the backup volumes.

12.4.4. Performing a Tape Copy Operation in Volume Mode

In a tape copy operation, you are copying the contents of one tape to another. The tape being copied must be one that was created by a previous DMPRST operation in volume mode. The DMPRST routine will not copy a tape created in file mode. DMPRST cannot be used to copy multivolume tape files to tape.

Input and output in a tape copy operation are always tapes. Therefore, // PARAM IN=TAPE and // PARAM OUT=TAPE must be specified along with the file names TAPEIN for input and TAPEOT for output.

The following is an example of a typical job stream used to perform the tape copy operation:

```
// JOB TAPECOPY
// DVC 20 // LFD PRNTR
// DVC 90 // VOL PAY002 // LFD TAPEIN
// DVC 91 // VOL PAY003 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=TAPE
/&
// FIN
```

12.4.5. Performing a Diskette Copy Operation in Volume Mode

In a diskette copy operation, you are copying the contents of one set of diskettes to another. The diskettes being copied must be a set that was created by a previous DMPRST operation in volume mode.

Input and output in a diskette copy operation are always diskettes. Therefore, // PARAM IN=SEQD and // PARAM OUT=SEQD must be specified along with the file names SEQDIN for input and SEQDOT for output.

The following is an example of a typical job stream used to perform the diskette copy operation:

```
// JOB DSKTCOPY
// DVC 20 // LFD PRNTR
// DVC 130 // VOL COPY01,COPY02,COPY03
// LBL DATA // LFD SEQDIN
// DVC 131 // VOL COPY0A,COPY0B,COPY0C
// LBL DATA // LFD SEQDOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=SEQD
/&
// FIN
```

The contents of the diskette set, COPY01, COPY02, COPY03, are copied to the diskette set COPY0A, COPY0B, COPY0C. Since the file DATA was allocated to each diskette during prep, the LBL name DATA must be supplied in each device assignment set.

12.5. Executing DMPRST in File Mode (Batch Environment)

You may execute DMPRST in the file mode to create backups of individual files on your disk or disks. Choose disk, tape, or data set label diskette as your backup, depending on which DMPRST procedure is used. Multidisk volumes can be dumped to a tape. In a single-disk volume environment, you can copy, dump, and restore an entire volume of active files. As in volume mode, you should assign a printer to list the // PARAM and FILE statements and any error messages.

Table 12-3 summarizes the // PARAM and FILE statements along with the appropriate LFD names and the type of operation that will be performed. A complete description of these statements is provided in 12.5.1 through 12.5.5.

Table 12-3. File Mode // PARAM and FILE Statements

Type of Operation	// PARAM Statement	LFD Name

Copy disk to disk	// PARAM IN=DISC
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		DISCIN
		DISCOT

Dump disk to MIRAM disk file	// PARAM IN=DISC
	// PARAM OUT=SEQD
	// PARAM TYPE=FILE
		DISCIN
		SEQDOT,,INIT

Dump disk to tape	// PARAM IN=DISC
	// PARAM OUT=TAPE
	// PARAM TYPE=FILE
		DISCIN
		TAPEOT

Dump multiple disks to tape	// PARAM IN=DISC(D01,D02,...Dnn)	DISCIN01 DISCIN02 : DISCINnn

Dump disk to diskette	// PARAM IN=DISC
	// PARAM OUT=SEQD
	// PARAM TYPE=FILE
		DISCIN
		SEQDOT,,INIT

Restore tape to disk	// PARAM IN=TAPE
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		TAPEIN
		DISCOT

Restore tape to multiple disks	// PARAM OUT=DISC(D01,D02,...Dnn)	DISCOT01 DISCOT02 : DISCOTnn

Restore diskette to disk	// PARAM IN=SEQD
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		SEQDIN
		DISCOT

Restore MIRAM disk file to disk	// PARAM IN=SEQD
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		SEQDIN
		DISCOT

Copy diskette to diskette	// PARAM IN=SEQD
	// PARAM OUT=SEQD
	// PARAM TYPE=FILE
		SEQDIN
		SEQDOT

continued

Table 12-3. File Mode // PARAM and FILE Statements (cont.)

Type of Operation	// PARAM Statement	LFD Name
Process only active permanent files	// PARAM TYPE=FILE,ALL	
Restore from any diskette or tape volume	// PARAM TYPE=FILE,ANY	
Avoid wait for file to be available	// PARAM TYPE=FILE,NOWAIT	
Suppress file expiration date check	// PARAM NOEXPCK	
Restart processing for dump/restore to tape or diskette	// PARAM RESTART	

12.5.1. Performing a Disk Copy Operation in File Mode

When performing a disk copy operation in file mode, you copy the contents of individual files on a disk to another disk of the same type.

Input and output in a disk copy operation are always disks. Therefore, specify // PARAM IN=DISC as your input parameter and // PARAM OUT=DISC as your output parameter. In addition, you must specify the file names DISCIN on the input // LFD statement and DISCOT on the output // LFD statement.

Since you are copying files, specify // PARAM TYPE=FILE to inform DMPRST that file, rather than volume, processing is performed.

Associated with the // PARAM TYPE=FILE parameter are FILE statements. By including these statements as embedded data in your control stream, you can name which files you want copied by either file name or file prefix.

The FILE statements are free-formatted, meaning that the first character may start anywhere. However, you must place a blank character between the FILE statement and the first operand to act as a delimiter. If you specify a file name containing embedded blanks, enclose it in quotation marks (i.e., FILE "PAY BACKUP").

If you want to copy an entire volume of active permanent files, use the FILE,ALL statement. By active permanent files, we mean all files that have entries in the VTOC and are not job temporary files (new libraries and scratch files). Any files deleted, but not physically removed, are not copied. The VTOC is not copied. (In volume mode, everything is copied.)

The following example shows a disk copy operation in file mode:

```
// JOB DFILCPY
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 51 // VOL DISK02 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE MONEY
        FILE INTEREST
        FILE BILLS
        FILE.P BANK
/*
/&
// FIN
```

The files MONEY, INTEREST, and BILLS and all files with the prefix of BANK on disk volume DISK01 are copied to disk volume DISK02. Note that the FILE statements are included as embedded data in the control stream.

The following example shows a disk copy operation in file mode copying the entire volume of active files:

```
// JOB COPYALL
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 51 // VOL DISK02 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE,ALL
/&
// FIN
```

Notes:

1. *Job temporary files (run libraries and scratch files) are not processed in this file mode.*

2. *This file mode must be used when copying the entire contents from or to an 8417 disk with fixed-head assembly because volume mode processing does not support fixed-head assembly processing.*

Before DMPRST processes any files, it locks them. Therefore, if you are dumping or restoring a file currently being used in the system, DMPRST waits until the file is available before processing it. You can avoid this delay by specifying the NOWAIT parameter on the FILE statement. DMPRST, upon detecting NOWAIT, skips processing any of the unavailable files and goes to the next available file. All files skipped are listed on the printer so you can process them later.

```
// JOB COPYALL
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 51 // VOL DISK02 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE,NOWAIT
/$
        FILE CREDIT
        FILE ASSETS
        FILE INVENTORY
/*
/&
// FIN
```

12.5.2. Performing a Dump Operation in File Mode

The dump operation enables you to copy individual files or an entire volume of active permanent files from a disk to tape, diskette, or a MIRAM file on a disk of a different type from the original. You may use any tape, data set label diskette, or a MIRAM disk file as a backup. However, the parameters specified in your control stream depend on which devices you are using.

In a dump operation, the input is always from disk. (Multiple disks can be dumped to tape, as described later in this subsection.) Therefore, specify // PARAM IN=DISC as your input parameter, and DISCIN as the file name on your input disk // LFD job control statement.

When performing a dump operation in file mode, you dump individual files. To inform DMPRST that you are dumping files rather than an entire volume, include the // PARAM TYPE=FILE parameter in your control stream.

Associated with the // PARAM TYPE=FILE statement are FILE statements. You use these statements to specify either the file name or the file prefix (up to 16 bytes) of the files you want dumped and you include them in your control stream as embedded data.

The FILE statements are free-formatted, meaning that the first character may start anywhere. However, you must place a blank character between the FILE statement and the first operand to act as a delimiter. If you specify a file name containing embedded blanks, enclose it in quotation marks (i.e., FILE "PAY BACKUP").

Dumping a Single Disk to Tape in File Mode

If you are using magnetic tape as output, specify // PARAM OUT=TAPE as your output parameter, and TAPEOT as the file name on your tape's // LFD job control statement. When using more than one tape as output, you can assign them in your control stream in one of the following ways:

1. Use a single device assignment set (DVC-LFD sequence) having a file name of TAPEOT. Included in this set are // VOL statements listing the volume serial numbers of the backup tapes in the DMPRST job.

2. Supply a separate device assignment set for each tape in your job. Here, a separate device is assigned for each tape volume. Again, the file name is TAPEOT; however, the numbers 01 through 99 are attached to each tape file name, following the first one (i.e., // TAPEOT01).

Since you are executing DMPRST in file mode, include the // PARAM TYPE=FILE parameter in your job stream.

Files must be restored in the same order as they were dumped. Therefore, you should keep a record of the file order. The simplest way is to save the listings from your dump operations.

The following examples show file dump operations using magnetic tape.

Example 1: Multifile Dump to a Single-Volume Tape

```
// JOB TFILDMP1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 90 // VOL PAY002 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE
/$
        FILE CREDIT
        FILE ASSETS
        FILE.P HIST
/*
/&
// FIN
```

The disk files CREDIT and ASSETS and all files with the prefix HIST on volume DISK01 are dumped to tape volume PAY002.

Example 2: Multifile Dump to Multivolume Tapes Using a Single Device Assignment Set

```
// JOB TFILDMP2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL MYPACK // LFD DISCIN
// DVC 90 // VOL BACK01,BACK02,BACK03,BACK04
// LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE
/$
        FILE RPGII
        FILE MYCOBOLMASTER
        FILE PROCFILE
        FILE ''PAY BACKUP''
/*
/&
// FIN
```

The disk files RPGII, MYCOBOLMASTER, PROCFILE, and PAY BACKUP are dumped from MYPACK to tape volumes BACK01, BACK02, BACK03, and BACK04. Here, a single device assignment set (DVC-LFD sequence) is used to assign the tapes. Each tape volume is mounted separately on the same drive. Note that even though you are accessing files on disk, there is no // LBL name in the control stream. Since DMPRST doesn't use the data management facility to search for your files, no // LBL statement is needed. Also, the name on the fourth FILE statement, "PAY BACKUP", is enclosed in double quotes because of the embedded blank.

Example 3: Multifile Dump to Multivolume Tapes Using Separate Device Assignment Sets

```
// JOB TFILDMP3
// DVC 20 // LFD PRNTR
// DVC 50 // VOL MYPACK // LFD DISCIN
// DVC 90 // VOL BACK01 // LFD TAPEOT
// DVC 91 // VOL BACK02 // LFD TAPEOT01
// DVC 92 // VOL BACK03 // LFD TAPEOT02
// DVC 93 // VOL BACK04 // LFD TAPEOT03
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE
/$
        FILE RPGII
        FILE MYCOBOLMASTER
        FILE PROCFILE
        FILE ''PAY BACKUP''
/*
/&
// FIN
```

This job is the same as the job performed in Example 2. Here, however, the tape volumes are assigned using separate device assignment sets. The volumes are mounted simultaneously on separate tape drives.

Example 4: Entire Volume Dump of Active Permanent Files to Multivolume Tapes Using a Single Device Assignment Set

```
// JOB DUMPALL
// DVC 20 // LFD PRNTR
// DVC 50 // VOL MYPACK // LFD DISCIN
// DVC 90 // VOL BACK01,BACK02
// LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE,ALL
/&
// FIN
```

All active permanent files (entries in the VTOC) are dumped.

Dumping Multiple Disks to Tape in File Mode

If you are performing a multivolume dump to tape, all the volumes must be online on drives of the same disk type. Use the LFD names DISCIN01, DISCIN02,...DISCIN*nn* for the disks. The input parameter is // PARAM IN=DISC(D01,D02,...D*nn*), where the numerics of the D*nn* definitions correlate with those in the // LFD statements. The numerics need not be contiguous. The value of *nn* must be a two-digit number from 01 to 16.

The ALL specification cannot be used on the TYPE=FILE statement for a multidisk volume dump.

Note: *Tapes created with multidisk volume inputs cannot be restored by DMPRST on releases prior to Release 12.0 or by any release of SU@RST.*

The following statements would be required to dump files from three disk volumes (DISCAA, DISCBB, and DISCCC):

```
// DVC 50 // VOL DISCAA // LFD DISCIN01
// DVC 51 // VOL DISCBB // LFD DISCIN02
// DVC 52 // VOL DISCCC // LFD DISCIN03
   .
   .
   .
// PARAM IN=DISC(D01,D02,D03)
```

Associated with each D*nn* statement are DISC data statements. One of them must precede any FILE statements associated with that disk. Thus, the JCL to dump files from three disks would appear as follows:

```
// JOB MULTIDUMP
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISCAA // LFD DISCIN01
// DVC 51 // VOL DISCBB // LFD DISCIN02
// DVC 52 // VOL DISCCC // LFD DISCIN03
// DVC 90 // VOL TAPE00 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC(D01,D02,D03)
// PARAM OUT=TAPE
// PARAM TYPE=FILE
/$
      DISC D01
      FILE PAYROLL
      DISC D03
      FILE PAYROLL
      DISC D02
      FILE TAXES
      DISC D03
      FILE PENSION
/*
/&
```

Note: *File PAYROLL is a multivolume file and two DISC and FILE statements are required to dump it in its entirety.*

Dumping to Diskette in File Mode

If you are using a diskette as output, specify // PARAM OUT=SEQD as your output parameter. DMPRST stores its control information and any data from the original disk in a MIRAM file on the backup diskettes. You must prep each backup diskette as a data set label diskette with a record size of either 128 or 256 bytes. Since a default // LBL name of DATA is assigned when your diskette is prepped in the data set label mode, use the name DATA in your DMPRST control stream.

A physical image of the files dumped from disk is stored in the MIRAM file DATA on diskette. Since the dumped files must be restored in the same order as they were dumped, you should keep a record of the file order. The simplest way to do this is to save the listings from your file dump operations.

If you choose an output file other than DATA, first scratch the name DATA from your diskette and allocate a MIRAM file having the new name. Since most dump operations require multiple diskettes, it's much easier to use the default name DATA than to change the name on each diskette.

The file name on your diskette's // LFD statement is SEQDOT,,INIT. Unlike the file name specification for tape, only a single device assignment set (DVC-LFD sequence) may be provided, even if you are using more than one diskette. You include the INIT parameter in the LFD statement to initialize the diskette so DMPRST can dump to it. Remember, however, that you must not include the INIT parameter when performing the restore operation from that diskette.

Since you are executing DMPRST in file mode, include the // PARAM TYPE=FILE parameter in your job stream.

The following example shows a file dump to diskette:

```
// JOB DSKTFIL1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL MYPACK // LFD DISCIN
// DVC 130 // VOL SAVE01,SAVE02,SAVE03,SAVE04
// LBL DATA // LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
// PARAM TYPE=FILE
/$
        FILE INVENTORY
        FILE PARTS
        FILE INVOICES
/*
/&
// FIN
```

The files INVENTORY, PARTS, and INVOICES on the disk volume MYPACK are dumped to the MIRAM file DATA on diskette. The DATA file is a multivolume file consisting of the volumes SAVE01, SAVE02, SAVE03, and SAVE04.

Dumping to a MIRAM Disk File in File Mode

When dumping to a MIRAM disk file, your output disk may be a different type from the original. For example, dumping from an 8417 disk to an 8419 disk. Before you can perform the dump operation, you must allocate a MIRAM file on the output disk.

If a single output disk is sufficient, you may allocate the MIRAM file by including the necessary job control statements in your DMPRST control stream. However, for more than one output disk, allocate the MIRAM file in a separate job before running your DMPRST job. For more information on allocating files, see the *Job Control Language Programming Guide* (UP-9986).

A physical image of the files dumped is stored in the MIRAM file you allocated on the output. Since dumped files are restored in the same order as they are dumped, you should keep a record of the file order. The simplest way to do this is to save the listings from your dump operations.

You must supply // PARAM OUT=SEQD as your output parameter and SEQDOT,,INIT as the file name on your output disks' // LFD statement. As in the case of a diskette, only a single device assignment set (DVC-LFD sequence) can be used. You must list all volumes on the // VOL card or cards unless // VOL SCRATCH is used. For more information on the // VOL SCRATCH statement, see the *Job Control Language Programming Guide* (UP-9986). You include the INIT parameter in the LFD statement to initialize the disk so DMPRST can dump to it. Remember, however, that you must not include the INIT parameter when performing the restore operation from that disk.

Since you are executing DMPRST in file mode, include the // PARAM TYPE=FILE parameter in your job stream.

If you want to dump all the files on a disk, you may specify // PARAM TYPE=FILE,ALL as file mode parameter. This parameter causes all the files on the original disk to be dumped. You don't need to include any FILE statements in your job stream when using the ALL option.

The // PARAM TYPE=FILE,ALL parameter allows you to dump the entire contents of an 8417 fixed-head volume file by file. Use this parameter when you want to dump an 8417 fixed-head disk volume, since this type disk can't be dumped in volume mode.

The following three examples show dump operations in file mode using disks:

Example 1: Multifile Dump to Single-Volume Disk

```
// JOB DFILDMP1
// DVC 20 // LFD PRNTR
// DVC 50 // VOL D00309 // LFD DISCIN
// DVC 51 // VOL BACKUP
// EXT MI,C,,CYL,100
// LBL FILESAVE
// LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
// PARAM TYPE=FILE
/$
        FILE INTERSTPRG
        FILE MORTGAGEPRG
        FILE BALANCE
/*
/&
// FIN
```

The disk files INTERESTPRG, MORTGAGEPRG, and BALANCE are dumped to the MIRAM file, FILESAVE, on the disk volume BACKUP. FILESAVE is allocated on the output disk using the EXT statement.

Example 2: Multifile Dump to Multivolume Disk

```
// JOB DFILDMP2
// DVC 20 // LFD PRNTR
// DVC 50 // VOL USA444 // LFD DISCIN
// DVC 51 // VOL SAVE01,SAVE02
// LBL RELEASED // LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
// PARAM TYPE=FILE
/$
          FILE ACCOUNTING
          FILE COBOLPROG
          FILE MARKETPROG
          FILE ''PRICE INDEX''
          FILE TAXPROG
          FILE.P INV
/*
/&
// FIN
```

The files ACCOUNTING, COBOLPROG, MARKETPROG, PRICE INDEX, and TAXPROG, and all files prefixed with INV, are dumped to the MIRAM file (RELEASED) on two 8419 disks. RELEASED is allocated on disk volumes SAVE01 and SAVE02. Since two backup volumes are used in this job, the MIRAM file is allocated in separate jobs prior to the execution of the DMPRST job. Note that a single device assignment set is used for the backup volumes.

Example 3: File-by-File Dump of an 8417 Fixed-Head Disk

```
// JOB FIXDUMP3
// DVC 20 // LFD PRNTR
// DVC 50 // VOL FIXVOL // LFD DISCIN
// DVC 51 // VOL FIX001,FIX002
// LBL DISKBACKUP // LFD SEQDOT,,INIT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=SEQD
// PARAM TYPE=FILE,ALL
/&
// FIN
```

All active permanent files on the 8417 fixed-head disk, FIXVOL, are dumped to disk volumes FIX001 and FIX002. Notice that no FILE statements are included because the // PARAM TYPE=FILE,ALL parameter is specified. This parameter must be used when dumping the entire contents of an 8417 fixed-head disk.

12.5.3. Performing a Restore Operation in File Mode

When you dump files from a disk, the backup created is stored on a MIRAM file or on tape. Therefore, you cannot use the backup on your system in the event your original data is lost. Before actually using any backup data on your system, you must restore it to the original disk or a disk of the same type as the original.

In a restore operation, the output is always a disk. Therefore, you must specify // PARAM OUT=DISC as your output parameter, and DISCOT as the file name on your disk's // LFD statement.

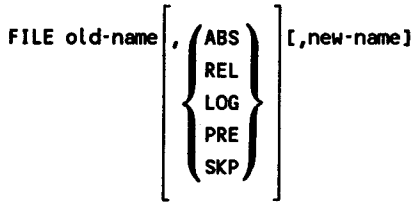
Since you are restoring files, include the // PARAM TYPE=FILE parameter in your job stream. This parameter informs DMPRST that file, rather than volume, processing is performed.

Associated with the // PARAM TYPE=FILE parameter are FILE statements. Like the FILE statements used in file dump operations, they name the files you want processed. In a restore operation, however, they also allow you to rename files and specify the type of processing you want performed.

The FILE statements are free-formatted, meaning that the first character may start anywhere. However, you must place a blank between the FILE statement and the first operand, and a comma between the operands to act as a delimiter. If you specify a file name containing embedded blanks, enclose it in double quotation marks.

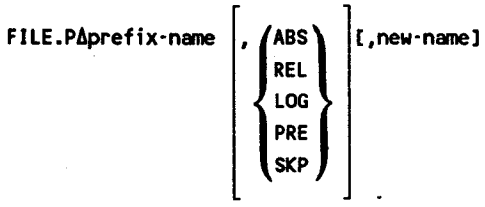
Two formats of the FILE statement are used in a restore operation:

Format 1



The old-name parameter specifies the name of the file you are restoring to disk.

Format 2



The prefix-name parameter specifies the prefix name of all the files to be restored. Regardless of the format, when restoring more than one file, you must restore the files in the order that they were dumped.

Using the Allocation Parameter to Control Restore Processing

The second parameter in the FILE statement is the allocation parameter. This optional parameter enables you to control file processing. The allocation parameters are

- ABS Locates a file on the same absolute extents. If a file cannot be allocated to the same extents, an error message is issued and DMPRST processes the next FILE statement.
- REL Relocates a file. If relocation to the same size file is not possible, a file large enough to hold the original file is allocated. If this is unsuccessful, an error message is issued and DMPRST processes the next FILE statement.
- LOG Relocates a file and deletes all unassigned space for that file. LOG deletes only space that is not assigned to a partition control appendage (PCA).
- PRE Relocates a file in a space that was preallocated. This file space can be located anywhere on the disk.
- SKP Suppresses the restore of a file. SKP is used only in diskette and tape restore operations. It must be used when restarting a tape restore; it is *not* required for a normal tape restore.

You can only specify one allocation parameter in each FILE statement. Enter it after the old-name parameter, with a comma separating the two.

When you want to perform standard restore processing, omit the allocation parameter. However, if you omit the allocation parameter and specify the new-name parameter, substitute a comma for the missing allocation parameter.

During standard restore processing, DMPRST scratches a file if it already exists on the output disk and makes up to three attempts to reallocate it on the disk.

When reallocating the file, DMPRST first attempts to allocate the same absolute extents occupied by the original file. If this is unsuccessful, the file relocation facility attempts to allocate the same size file at a different location. If this second try is also unsuccessful, the file relocation facility determines how much of the original space is actually assigned. It then tries to allocate a file large enough to hold the assigned space from the original file. If all three attempts fail, an error message is issued to the system console.

Using the File Prefix Parameter

The file prefix parameter gives you the capability to restore the first series of consecutive files having a certain prefix in the file name. Suppose you want to restore all files having the file name prefix of OLD. In addition, the restored files are to be located on the same absolute extents. The following example shows the restore operation using the prefix name and ABS parameters:

```
// LFD UPDATE
// DVC 20 // LFD PRNTR
// DVC 130 // VOL SAVE01,SAVE02,SAVE03
// LFD SEQDIN
// DVC 50 // VOL NEVOL1,NEVOL2,NEVOL3
// LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE.P OLD,ABS
/*
/&
// FIN
```

Note: When restoring files from tape, diskette, or sequential files on disk, the restore of files with the same prefix will terminate when the first file without that prefix is encountered on the input medium or at the end of the input file. The next file statement, if any, is then processed.

Using the New-Name Parameter to Rename Files

The third parameter in the FILE statement is the new-name parameter. You use it to change the name of a file being restored to disk. For example, suppose you want to restore the diskette file MYFILE to a file named PAYROLL on disk. In order to copy MYFILE to PAYROLL, you specify PAYROLL as the new-name parameter in the FILE statement with the old-name MYFILE. In this case, DMPRST first scratches any file having the name PAYROLL from the output disk. Next, DMPRST allocates a file large enough for MYFILE. The old file, MYFILE, bearing the new name PAYROLL, is then written to the allocated space on the output disk.

The following example shows a restore operation using the new-name parameter:

```
// JOB RENAME
// DVC 20 // LFD PRNTR
// DVC 130 // VOL SAVE01,SAVE02,SAVE03
// LFD SEQDIN
// DVC 50 // VOL BAKVOL // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE OLDNAME,REL,NEWNAME
        FILE OLDACCOUNTS,,NEWACCOUNTS
/*
/&
// FIN
```

The files are restored to disk with the new file name specified by the new-name parameter. For the first file (OLDNAME), DMPRST scratches any file having the name NEWNAME from the output disk. DMPRST then allocates a file large enough for the file OLDNAME. The old file OLDNAME bearing the new file name NEWNAME is then written to the allocated space on the output disk. The same procedure is followed for the second FILE statement. Here, however, a comma is substituted for the missing allocation parameter. As in all restore operations, the files are restored in the same order they were dumped.

Global File Allocation Parameters

If you are restoring all files in a single-disk volume environment, you can specify a global allocation parameter (REL, LOG, PRE, and ABS) on the TYPE=FILE,ALL parameter and it will apply to all the files being restored. The function of these global specifications is the same as for the allocation parameters specified on a FILE statement (see "Using the Allocation Parameter to Control Restore Processing" earlier in this subsection).

Restoring from Tape to a Single-Disk Volume in File Mode

If you are using magnetic tape as input, specify // PARAM IN=TAPE as your input parameter, and TAPEIN as the file name on your tape's // LFD statement. When using more than one tape as your input, you can assign them in your control stream in one of two ways.

One way uses a single device assignment set (DVC-LFD sequence), which has a file name of TAPEIN. Included in this set are // VOL statements listing the volume serial numbers of the restore operation's input tapes.

Another way is to supply a separate device assignment set for each tape. Here, a separate device is assigned for each tape volume. Again, the file name on your // LFD statement is TAPEIN. However, the numbers 01 through 99 are attached to each tape file name, following the first one (i.e., FILE // LFD TAPEIN01).

Since you are executing DMPRST in file mode, include // PARAM TYPE=FILE in your job stream. Also, supply the appropriate FILE statements as embedded data. Remember, files must be restored in the same order they were dumped. If you want to restore the entire volume of active files, use // PARAM TYPE=FILE,ALL.

Notes:

1. *Tapes created in multidisk volume format must be restored using the JCL described in "Restoring Multiple Disks from Tape in File Mode" later in this subsection, even when restoring to only one of the disks.*

2. *If you perform a tape dump operation using separate device assignment sets and then restore using a single device assignment set, you will receive an error message from data management. The method of assigning tapes used in the dump operation should also be used in the related restore operation.*

The following three examples show restore operations using magnetic tape.

Example 1: Multifile Restore from Single-Volume Tape File

```
// JOB TFILRST1
// DVC 20 // LFD PRNTR
// DVC 90 // VOL PAY002 // LFD TAPEIN
// DVC 50 // VOL DISK01 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE CREDIT,ABS
        FILE ASSETS,REL
/*
/&
// FIN
```

The files that were dumped to tape by the dump operation in Example 1 of "Dumping a single Disk to Tape in File Mode" in 12.5.2 are restored to the disk volume DISK01. Note that the files are restored in the same order in which they were dumped.

The first file, CREDIT, is restored to the same absolute extents that it originally occupied. This function is controlled by the allocation parameter, ABS. The second file, ASSETS, is restored to a different location on disk by the REL allocation parameter.

Example 2: Multivolume Restore from Multivolume Tapes Using a Single Device Assignment Set

```
// JOB RENAME
// DVC 20
 // LFD PRNTR
// DVC 90 // VOL BACK01,BACK02,BACK03,BACK04
// LFD TAPEIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE RPGII,REL,RPGII02
        FILE MYCOBOLMASTER,ABS
        FILE PROCFILE,,PROCFILE02
        FILE ''PAY BACKUP''
/*
/&
// FIN
```

The files that were dumped in Example 2 of "Dumping a Single Disk to Tape in File Mode" in 12.5.2 are restored to the disk volume, MYPACK. Since the tapes were assigned in the dump operation using a single device assignment set, they are assigned in the restore operation in the same manner.

Note that the files are restored in the same order in which they were dumped. The first FILE statement relocates the RPGII file to a new location on the output disk and changes the file name to RPGII02. The second FILE statement restores the file, MYCOBOLMASTER, to the same absolute extents it occupied on the original disk. The third FILE statement changes the name, PROCFILE, to PROCFILE02. Because no allocation parameter is provided, a comma is inserted in its place.

Example 3: Multifile Restore from Multivolume Tapes Using Separate Device Assignment Sets

```
// JOB TFILRST3
// DVC 20 // LFD PRNTR
// DVC 90 // VOL BACK01 // LFD TAPEIN
// DVC 91 // VOL BACK02 // LFD TAPEIN01
// DVC 92 // VOL BACK03 // LFD TAPEIN02
// DVC 93 // VOL BACK04 // LFD TAPEIN03
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE RPGII,REL,RPGII01
        FILE MYCOBOLMASTER,ABS
        FILE PROCFILE,,PROCFILE02
        FILE ''PAY BACKUP''
/*
/&
// FIN
```

This job stream is the same as the job stream in Example 2. In this example, however, each tape volume is assigned via an individual device assignment set. The tape volumes are mounted simultaneously on separate tape drives.

Restoring Multiple Disks from Tape in File Mode

If you are restoring from a tape created from multidisk inputs, use the LFD statements TAPEIN and DISCOT01, DISCOT02,...DISCOT*nn*.

The input parameter is // PARAM IN=TAPE and the output parameter is // PARAM OUT=DISC(D01,D02,...D*nn*), where the numerics of the D*nn* statements must correlate with those in the LFD statements. The numerics associated with each disk must be the same as those used on the LFD and D*nn* statements when the input tape was created. The numerics need not be contiguous.

DISC data statements are associated with each D*nn* statement. One of them must precede any FILE statements associated with a disk.

Files that are dumped from two different disks cannot be restored to one disk, nor can files dumped from one disk be restored to two different disks. The D*nn* identifier is always associated with the file on the output tape.

You cannot specify ALL on the TYPE=FILE statement when doing a multidisk volume restore, nor can you use the global allocation parameters.

The JCL required to restore the PAYROLL file dumped in the example in "Dumping Multiple Disks to Tape in File Mode" earlier in this subsection is as follows:

```
// JOB RESTORE
// DVC 20 // LFD PRNTR
// DVC 90 // TAPE00 // LFD TAPEIN
// DVC 50 // VOL DISCAA // LFD DISCOT01
// DVC 51 // VOL DISCCC // LFD DISCOT03
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC(D01,D03)
// PARAM TYPE=FILE
/$
      DISC D01
      FILE PAYROLL
      DISC D03
      FILE PAYROLL
/*
/&
```

Restoring from a Diskette in File Mode

If you are using diskette as input, specify // PARAM IN=SEQD as your input parameter, and SEQDIN as the file name on your diskette's // LFD statement. Only specify a single device assignment set (DVC-LFD sequence) when multivolume diskettes are used.

Since you are executing DMPRST in file mode, include the // PARAM TYPE=FILE parameter in your job stream. Also, include the necessary FILE statements as embedded data in your job stream. Remember that the files must be restored in the same order as they were dumped.

Unlike the tape restore operation, when restoring from diskettes you must provide a file statement for every file that was dumped in the related dump operation. In a tape restore, if you want to prevent a file from being restored, simply omit the FILE statement for that file. In a diskette restore operation, however, to prevent a file from being restored, specify the SKP parameter on the file's FILE statement (i.e., FILE MYFILE,SKP).

In a multivolume diskette restore operation, you can start restoring from a volume other than the first volume by specifying the ANY parameter on the // PARAM TYPE=FILE statement. When ANY is specified, DMPRST searches the mounted diskette for the file name specified on the FILE statement. Any files preceding the specified file on the volume are skipped.

The following two examples show typical restore operations using diskettes:

Example 1

```
// JOB STDRST01
// DVC 20 // LFD PRNTR
// DVC 130 // VOL SAVE01,SAVE02,SAVE03,SAVE04
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE INVENTORY
        FILE PARTS,SKP
        FILE INVOICES
/*
/&
// FIN
```

All the files that were dumped to diskette by the dump operation in 12.5.2, except PARTS, are restored to the disk volume, MYPACK. Notice that because the files are restored from diskette, all the files that were dumped are listed even though the PARTS file isn't being restored. The SKP parameter in the second FILE statement prevents PARTS from being restored.

Since the files INVENTORY, PARTS, and INVOICES are stored in the MIRAM file (DATA) on diskette, the LBL name DATA is included in the diskette's device assignment set. Note that the files are restored in the same order they were dumped.

The only parameter used in the FILE statements is the old-name parameter. Since the allocation parameter is omitted from these FILE statements, standard restore processing is performed (see "Using the Allocation Parameter to Control Restore Processing" earlier in this subsection).

Example 2

```
// JOB RESTART
// DVC 20 // LFD PRNTR
// DVC 130 // VOL,,SAVE03,SAVE04
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE,ANY
/$
        FILE INVOICES
/*
/&
// FIN
```

In this restore operation, the parameter ANY is specified on the // PARAM TYPE statement enabling DMPRST to begin restoring from a volume other than the first one. DMPRST checks the file name on the FILE statement as the starting point for the restore operation. Any files preceding the starting point (INVOICES) are skipped.

Note: When diskette volumes are being skipped, the omitted volume serial numbers must be indicated by commas.

Restoring from a MIRAM Disk File in File Mode

Since the input disk in a restore operation is a different type from the original, specify // PARAM IN=SEQD as your input parameter, and SEQDIN as the file name on your input disk's // LFD statement.

You must include the name of the input disk's MIRAM file in your control stream. This is the same name that was assigned for the related dump operation. Remember, only a single device assignment set may be used to assign your input disk volumes.

Since you are executing DMPRST in file mode, include the // PARAM TYPE=FILE statement in your job stream. You must also supply the appropriate FILE statements as embedded data. Remember, the files must be restored in the same order in which they were dumped.

The following example shows a restore operation using a disk:

```
// JOB STDRST02
// DVC 20 // LFD PRNTR
// DVC 50 // VOL BACKUP // LBL FILESAVE // LFD SEQDIN
// DVC 51 // VOL D88088 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE INTERESTPRG
        FILE MORTGAGEPRG
        FILE BALANCE
/*
/&
// FIN
```

The files that were dumped to disk by the dump operation in Example 1 of "Dumping to a MIRAM Disk File in File Mode" in 12.5.2 are restored to the disk volume D88088. This volume is not the original volume (D00309), but it's the same type disk as the original.

Since the allocation parameter is omitted from the FILE statements, standard restore processing is performed. Remember, the files must be restored in the same order in which they were dumped.

12.5.4. Performing a Diskette Copy Operation in File Mode

You can copy the contents of a set of diskettes created in file mode to another set of diskettes. Unlike the disk copy operation, however, you cannot copy individual files. Instead, the entire contents of the diskette are copied.

Input and output in a diskette copy operation are always diskettes. Therefore, // PARAM IN=SEQD and // PARAM OUT=SEQD must be specified along with the file names SEQDIN for input and SEQDOT for output. Also, since you are performing the diskette copy operation in file mode, include the // PARAM TYPE=FILE parameter in your job stream.

The following is an example of a typical job stream used to perform the diskette copy operation:

```
// JOB COPYDSKT
// DVC 20 // LFD PRNTR
// DVC 130 // VOL COPY01,COPY02,COPY03
// LBL DATA // LFD SEQDIN
// DVC 131 // VOL COPY0A,COPY0B,COPY0C
// LBL DATA // LFD SEQDOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=SEQD
// PARAM TYPE=FILE
/&
// FIN
```

The contents of the diskette set, COPY01, COPY02, COPY03, are copied to the diskette set, COPY0A, COPY0B, COPY0C.COPY02. Since the DATA file was allocated to each diskette during prep, the LBL name, DATA, must be supplied in each device assignment set.

12.5.5. Copying Files in a Single-Disk Environment

You can make copies of a file on the same disk using the new-name parameter of the FILE statement. Here, the copy you create is written on the same disk as the original file. The new file, however, is assigned a new file name.

Since the same disk is used as both the input and output volume, the same device must be assigned having the // LFD names of DISCIN and DISCOT. Also, specify // PARAM IN=DISC and // PARAM OUT=DISC.

The new-name parameter of the file statement must be specified when you're copying files on the same disk, since two files with the same name can't exist on the same disk volume. With the new-name parameter, you can copy the file and give it a new name.

For example, suppose you are copying the MYFILE file on your disk. In order to copy MYFILE, specify the name MYFILE02 as the new-name parameter on the FILE statement.

In this case, DMPRST first scratches any file having the name MYFILE02 from the disk. Next, DMPRST allocates a file large enough for MYFILE. The old file, MYFILE, is then copied to the allocated space and given the new name, MYFILE02. Now, two identical files having different file names exist on the same disk.

The following example shows a single-volume file copy operation:

```
// JOB FILCOPY
// DVC 20 // LFD PRNTR
// DVC 50 // VOL DISK01 // LFD DISCIN
// DVC 50 // VOL DISK01 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE MYFILE,,MYFILE02
/*
// FIN
```

A copy of the MYFILE file on disk volume DISK01 is written to another location on the same disk. Note that the same disk is assigned as input and output.

Here, DMPRST first scratches any file on the disk having the name MYFILE02. Next, DMPRST allocates a file large enough for MYFILE. The old file, MYFILE, is then copied to the allocated space and given the name MYFILE02.

12.6. Restarting a Dump/Restore Operation

You can use the DMPRST routine to restart a diskette or tape dump or restore operation that terminated prematurely. This saves you from rerunning the entire job. This restart function, however, can only be performed when dumping from a disk to diskette or tape, or when restoring from diskette or tape to disk.

You can restart a diskette or tape DMPRST operation in either volume or file mode. In the following paragraphs, we will discuss the procedures for restarting diskette and tape DMPRST operations and will emphasize the differences between the two modes.

12.6.1. Restarting a Diskette Dump/Restore Operation

The first step in restarting a DMPRST job is to mount the correct diskette volume. You must mount a diskette that was completely processed before the original job terminated. If you don't know which volume was completed last, select a volume you are sure was completed.

For example, suppose you are restoring the file named MYFILE from diskette volumes A, B, C, D, E, F, and G to disk, and you aren't sure whether volume D was completed at the time the job was terminated. In this case, you should mount either volume B or C, since you know these volumes were completed. Remember, however, that you cannot mount the first volume of a diskette file when performing a restart. So, in this example, the first volume of MYFILE, volume A, can never be used to restart.

The second step in restarting your DMPRST job is to revise the statements in your original job stream. You will later run this revised job stream to restart your job.

Consider the // VOL statements. Because you are not using all of the original volumes in your restart operation, make changes to the // VOL statements in your control stream. The volume serial numbers of the diskettes not being used in the restart operation must be replaced by commas. So, if you originally specified

```
// VOL A,B,C,D,E,F,G
```

and you're restarting the job with volume C, you would change your // VOL statement to the following:

```
// VOL ,,C,D,E,F,G
```

Note that volumes A and B in this example are replaced by commas. You can also replace the original // VOL statements with a single // VOL SCRATCH statement. While this statement eliminates the need to specify any volume serial numbers in your restart control stream, it suppresses data management's checking for the proper volume serial numbers. Therefore, ensure that the operator knows exactly what volumes to mount and the proper sequence in which to mount them. For more information on the // VOL statement and the // VOL SCRATCH statement, see the *Job Control Language Programming Guide* (UP-9986).

Before restarting your job, add the statement // PARAM RESTART to the set of PARAM statements in your revised job stream. This statement activates the DMPRST restart function.

At this point, if you are running DMPRST in volume mode, you are ready to initiate the restart function by running your revised job stream. However, if you are restarting a restore operation in file mode, consider the following: When you are restarting a restore operation in file mode, include the same FILE statements in your restart job that were used in the terminated restore job. So, if your original restore job contained FILE statements using the SKP parameter (see 12.5.3), include these statements in the restart job.

The following examples are used to show a restart procedure. The first example is the control stream for the original job. The second example is the control stream used to perform the actual restart.

Example 1: Original Diskette File Restore Operation

```
// JOB DSKTRST
// DVC 20 // LFD PRNTR
// DVC 130 // VOL DSKT01,DSKT02,DSKT03,DSKT04,DSKT05,DSKT06
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE INVENTORY
        FILE PARTS,SKP
        FILE INVOICES
        FILE CREDITS
        FILE PAYMENTS,SKP
/*
/&
// FIN
```

Example 2: Restart Control Stream for Original Job

```
// JOB DSKTRST
// DVC 20 // LFD PRNTR
// DVC 130 // VOL ,,,,DSKT04,DSKT05,DSKT06
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM RESTART
// PARAM TYPE=FILE
/$
        FILE INVENTORY
        FILE PARTS,SKP
        FILE INVOICES
        FILE CREDITS
        FILE PAYMENTS,SKP
/*
/&
// FIN
```

The restart control stream in Example 2 is basically the same as the original control stream. In the original job, however, DMPRST was processing DSKT05 when it terminated. Therefore, DSKT04, the last volume completely processed, is mounted first for the restart job. Notice that the volume serial numbers of the diskettes not being used in the restart job are replaced with commas. The remainder of the control stream in Example 2 is identical to the original, except for the // PARAM RESTART statement that is added to activate the restart function.

12.6.2. Restarting a Tape Dump/Restore Operation

You can restart a tape dump or restore operation that terminated prematurely, unless the tape was created with a release prior to Release 12.0.

When restarting a tape operation, refer to the output of the original job. DMPRST issues messages to indicate when a tape is being read or written. It also issues a message when it is finished reading (closing) a tape. Mount a tape that DMPRST has already written to or read from; or, if restoring, mount the next volume after one that was closed. The first volume of the operation can never be mounted for a restart; the original job must be rerun.

The original job control stream must be altered for the restart. If the original // VOL statement for the tapes appears as

```
// VOL TAPE01,TAPE02,TAPE03
```

and DMPRST started to read (write) tape02, the statement must be changed as follows:

```
// VOL ,,TAPE02,TAPE03
```

Note that an extra comma must be entered.

A // PARAM RESTART statement must also be added to the job stream.

At this point, if you are running DMPRST in volume mode, you are ready to restart by running your revised job stream. However, if you are running in file mode and you are restarting a restore operation, some changes may be required for the FILE statements if you are not restoring all files. A FILE statement with a SKP parameter (see "Restoring from a Diskette in File Mode" in 12.5.3) must be added for any file that is not being restored and that precedes one on the tape that is being restored.

The following example shows a restart for a tape restore. The first control stream is for the dump that created the tape input for the restore. The second is the control stream for the restore. The third is for the restart.

The following is the control stream to dump the files:

```
// JOB TAPEDUMP
// DVC 20 // LFD PRNTR
// DVC 90 // VOL TAPE01,TAPE02,TAPE03 // LFD TAPEOT
// DVC 50 // VOL MYPACK // LFD DISCIN
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
      FILE INVENTORY
      FILE PARTS
      FILE INVOICES
      FILE CREDITS
      FILE PAYMENTS
/*
/&
// FIN
```

The following is the control stream for the original restore job (Note that the INVENTORY and PARTS files are not being restored.):

```
// JOB TAPERSTR
// DVC 20 // LFD PRNTR
// DVC 90 // VOL TAPE01,TAPE02,TAPE03 // LFD TAPEIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
      FILE INVOICES
      FILE CREDITS
      FILE PAYMENTS
/*
/&
// FIN
```

The following is the restart control stream:

```
// JOB TAPERSTR
// DVC 20 // LFD PRNTR
// DVC 90 // VOL ,,TAPE02,TAPE03 // LFD TAPEIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
// PARAM RESTART
/$
      FILE INVENTORY,SKP
      FILE PARTS,SKP
      FILE INVOICES
      FILE CREDITS
      FILE PAYMENTS
/*
/&
// FIN
```

12.7. Checking for File Expiration Date

DMPRST routine automatically checks for file expiration dates on your output volume thus saving you the time and expense of keeping outdated files. DMPRST checks the file expiration date by comparing that date with the system date. If the date has not expired, a message is displayed asking you to process the file, ignore the date, or terminate the job. The expiration date function is used in both volume and file modes when copying or restoring files.

To suppress the file expiration date checking feature, specify the // PARAM NOEXPCK statement. The following example is a typical control stream using the expiration date checking function in file mode. In this example, the file expiration date function is done automatically (omit the // PARAM NOEXPCK) in restoring disk files from diskette. In addition, specify the FILE PREFIX statement to restore all files having the prefix MAST.

```
// JOB DSKTRST
// DVC 20 // LFD PRNTR
// DVC 130 // VOL DSKT01,DSKT02,DSKT03,DSKT04,DSKT05,DSKT06
// LBL DATA // LFD SEQDIN
// DVC 50 // VOL MYPACK // LFD DISCOT
// EXEC DMPRST
// PARAM IN=SEQD
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE INVENTORY
        FILE.P MAST
        FILE INVOICES
        FILE CREDITS
/*
/&
// FIN
```

12.8. Listing Files on an Input Medium

You can print a list of the files backed up on a tape, a diskette, or a MIRAM file on disk by specifying LIST on the TYPE=FILE parameter.

The following example shows a list operation from tape:

```
// JOB LIST
// DVC 20 // LFD PRNTR
// DVC 90 // VOL TAPE1 // LFD TAPEIN
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM TYPE=FILE,LIST
/&
```

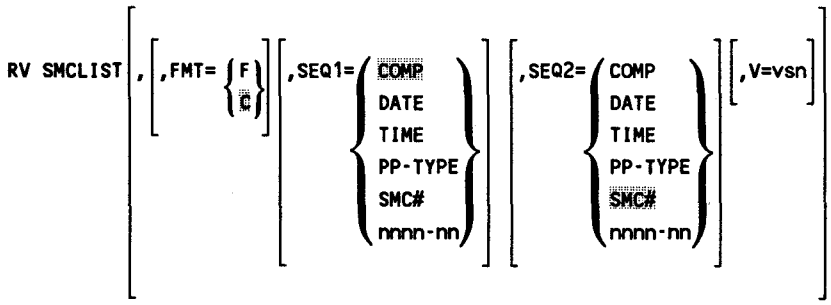
Section 13
List Software Maintenance Corrections (SMCLIST)

13.1. SMCLIST Function

You can use the SMCLIST canned job control stream to print a listing of the software maintenance corrections contained in the SMCLOG file. This listing may be printed in either a full or a condensed format. Also, by specifying certain parameters, you can produce listings sorted by SMC number, component number, program-product-type number, date, and time.

13.2. Executing SMCLIST

The format of the SMCLIST canned job control stream is



where:

FMT= F / C
: Specifies the format of the listing being produced.

FMT=F
: Specifies that a full listing is printed.

FMT=C
: Specifies that a condensed listing is printed.

A full listing is a listing sorted primarily by component number and then by SMC number. It gives more information about each SMC than a condensed listing, such as the regenerations an SMC requires or the method used to install it.

Since you don't always need as much information as the full listing shows, we also provide a condensed listing. A condensed listing contains only SMC numbers in ascending order and an indication of whether any SMCs were backed out, replaced, or were not installed because of an error during installation. (See Figures 13-1 and 13-2 for an example of each listing.)

The default for the FMT parameter is C for condensed. Unless you specify FMT=F on your SMCLIST run command, you will always receive a condensed listing of the SMCs in the SMCLOG file.

SEQ1=

Specifies the primary sorting key to be used.

SEQ1=COMP

Specifies the component number.

SEQ1=DATE

Specifies the date that the SMCs were applied.

SEQ1=TIME

Specifies the time that the SMCs were applied.

SEQ1=PP-TYPE

Specifies the program-product-type number.

SEQ1=SMC#

Specifies the SMC number.

SEQ1=nnnn-nn

Specifies that only those SMCs with a program-product-type number of *nnnn-nn* will be printed.

If you omit the SEQ1 keyword, the full listing is sorted primarily by component number.

SEQ2=

Specifies the secondary sorting key to be used.

SEQ2=COMP

Specifies the component number.

SEQ2=DATE

Specifies the date that the SMCs were applied.

SEQ2=TIME

Specifies the time that the SMCs were applied.

SEQ2=PP-TYPE

Specifies the program-product-type number.

SEQ2=SMC#
Specifies the SMC number.

SEQ2=nnnn-nn
Specifies that only those SMCs with a program-product-type number of *nnnn-nn* will be printed.

If you omit the SEQ2 keyword, the secondary sorting key is the SMC number.

V=vsn
Specifies the volume serial number for a system release pack other than RES.

Notes:

1. *The SEQ1 specification may not be the same as the SEQ2 specification.*
2. *If* nnnn-nn *is specified for the SEQ1 or SEQ2 parameter, any SMCs that were replaced do not appear in the listing.*

The condensed listing in Figure 13-2 shows all of the SMCs contained in the SMCLOG file. It is sorted by SMC number and shows all the SMCs in ascending order. Some of the SMCs show codes after them indicating that they were

- Replaced by Unisys because they produced adverse effects on OS/3. Any SMCs of this kind show a code of -R after them, for example, C082187-R.
- Backed out by you under the direction of a Unisys representative because they produced adverse effects on your particular system. Any SMCs of this kind show a code of -B after them, for example, C082005-B.
- Not applied to your system because of an error during installation. These SMCs are followed by a code of -E, for example, C082058-E.

Note: *An additional code of -I may also appear on your condensed listing. This code applies to non-SMC records and indicates that Unisys recorded information about that record in the SMCLOG file to be used by the SMC job stream. You can ignore any records containing this code.*

```
OS/3 SMC LOG FILE DISPLAY    RELEASE-ID=12.0.0  -1    FORMAT=F    SEQ1=COMP    SEQ2=SMC#    07/25/88    17:3*    PAGE=0001

 SMC                    PP-TYPE                  REQUIRED          APPLICATION
NUMBER      COMP        NUMBER      SYSTEM       RE-GENS      DATE     TIME   METHOD   CHARACTER   SMC/SMP STATUS
  (1)        (2)          (3)         (4)          (5)          (6)              (7)        (8)          (9)
C071187     A000        6210-00     8.0ONLY                  12/17/80 15:45  SMC                  NOT BACKED-UP
C071957     A000        6210-00     9.0ONLY                  12/17/80 15:54  SMC                  NOT BACKED-UP
C072168     A000        6210-00     8G0NLY                   12/17/80 15:57  SMC                  NOT BACKED-UP
C072238     A000        6210-00                              01/10/81 16:36  SMC                  NOT BACKED-UP
C072280     A000        6210-00                              12/17/80 15:59  SMC                  NOT BACKED-UP
C072360     A000        6210-00                              01/10/81 16:33  SMC                  NOT BACKED-UP
C072470     A000        6210-00                              01/10/81 16:37  SMC                  NOT BACKED-UP
C072509     A000        6210-00                              01/10/81 16:41  SMC                  NOT BACKED-UP
C072524     A000        6210-00                              01/26/81 19:29  SMC                  NOT BACKED-UP
C072539     A000        6210-00                              01/26/81 19:52  SMC                  NOT BACKED-UP
C072642     A000        6210-00                              02/10/81 17:42  SMC                  NOT BACKED-UP
C072667     A000        6210-00                              02/10/81 18:45  SMC                  NOT BACKED-UP
C072715     A000        6210-00                              02/10/81 19:01  SMC                  NOT BACKED-UP
C072782     A000        6210-0.                  YES         03/07/81 15:08  SMC                  NOT BACKED-UP
C072800     A000        6210-00                              02/10/81 13:43  SMC                  NOT BACKED-UP
T070011     A000        6210-00     COMMON                   07/09/81 13:22  SMC                  **PUN PROCESSOR ERROR
C072219     A010        6216-00                              12/17/80 16:02  SMC                  NOT BACKED-UP
C072298     A010        6216-0.                              12/17/80 16:00  SMC                  NOT BACKED-UP
C072427     A010        6216-00                              01/26/81 18:51  SMC                  NOT BACKED-UP
C072476     A010        6216-0.                              01/10/81 16:43  SMC                  NOT BACKED-UP
C072479     A010        6216-0.                              01/10/81 16:44  SMC                  NOT BACKED-UP
C072487     A010        6216-00                              01/10/81 16:46  SMC                  NOT BACKED-UP
C072527     A010        6216-00                              01/10/81 16:48  SMC                  NOT BACKED-UP
C072532     A010        6216-00                              01/23/81 14:05  SMC                  NOT BACKED-UP
C072249     A011        6210-00                              12/17/80 16:11  SMC                  NOT BACKED-UP
C072577     A011        6210-0.                              01/26/81 21:31  SMC                  NOT BACKED-UP
C072610     A011        6210-00                              01/26/81 23:52  SMC                  NOT BACKED-UP
C072393     A012        6214-00                              01/10/81 16:49  SMC                  NOT BACKED-UP
C072630     A012        6214-00                              01/27/81 00:36  SMC                  NOT BACKED-UP
C072361     A014        6210-00                              12/17/80 16:13  SMC                  NOT BACKED-UP
C072425     A014        6210-00                              01/10/81 16:51  SMC                  NOT BACKED-UP
C072432     A014        6210-00                              01/10/81 16:52  SMC                  NOT BACKED-UP
C072854     A014        6210-00                              03/31/81 12:41  SMC                  NOT BACKED-UP
C072871     A014        6210-00                              03/31/81 12:46  SMC                  NOT BACKED-UP
C072395     A015        6210-00                              01/10/81 16:54  SMC                  NOT BACKED-UP
C072471     A015        6210-00                              01/10/81 16:56  SMC                  NOT BACKED-UP
C072510     A015        6210-00                              01/10/81 16:57  SMC                  NOT BACKED-UP
C072679     A015        6210-0.                              02/10/81 18:47  SMC                  NOT BACKED-UP
C072806     A015        6210-00                  YES         03/07/81 15:12  SMC                  NOT BACKED-UP
C073071     A015        6210-00     COMMON                   07/09/81 12:40  SMC                  NOT BACKED-UP
C072278     A017        6210-00                              01/10/81 16:58  SMC                  NOT BACKED-UP
C072521     A019        6216-00                              01/10/81 17:00  SMC                  NOT BACKED-UP
C072585     A019        6216-0.                              01/26/81 22:25  SMC                  NOT BACKED-UP
C071956     A020        6210-00                              80/80/80 80:80  SMC                  **PROGRAM ERROR
C072115     A020        6210-00                              12/17/80 16:20  SMC                  NOT BACKED-UP
C072210     A020        6210-00                              12/17/80 16:22  SMC                  NOT BACKED-UP
C072233     A020        6210-00                              12/17/80 16:24  SMC                  NOT BACKED-UP
C072275     A020        6210-00                              12/17/80 16:25  SMC                  NOT BACKED-UP
C072295     A020        6210-00                              12/17/80 16:27  SMC                  NOT BACKED-UP
C072334     A020        6210-00                              12/17/80 16:29  SMC                  NOT BACKED-UP
```

Figure 13-1. Sample of Full SMC Listing (Part 1 of 2)

```
OS/3 SMC LOG FILE DISPLAY   RELEASE-ID= 12.0.0  -1   FORMAT=F    SEQ1=COMP    SEQ2=SMC#    07/25/88    17:34    PAGE=0002

 SMC                    PP-TYPE                  REQUIRED          APPLICATION
NUMBER      COMP        NUMBER      SYSTEM       RE-GENS      DATE     TIME   METHOD   CHARACTER   SMC/SMP STATUS
   ①          ②           ③          ④            ⑤          ⑥        ⑥       ⑦          ⑧          ⑨
C0072349    A020        6210-00                              12/17/80 16:31  SMC                    NOT BACKED-UP
C0072508    A020        6210-00                              01/10/81 17:01  SMC                    NOT BACKED-UP
C0072525    A020        6210-00                              01/26/81 19:35  SMC                    NOT BACKED-UP
C0072561    A020        6210-00                              01/26/81 20:53  SMC                    NOT BACKED-UP
C0072705    A020        6210-00                    YES       02/11/81 12:11  SMC                    NOT BACKED-UP
C0072324    A021        6210-00                              12/17/80 16:33  SMC                    NOT BACKED-UP
C0072347    A022        6210-00                              12/17/80 16:35  SMC                    NOT BACKED-UP
C0072387    A022        6210-00                              01/10/81 17:02  SMC                    NOT BACKED-UP
C0072531    A022        6210-00                              01/26/81 19:41  SMC                    NOT BACKED-UP
C0071490    A023        6210-00                              01/10/81 17:04  SMC                    NOT BACKED-UP
C0071577    A023        6210-00                              01/10/81 17:06  SMC                    NOT BACKED-UP
C0072234    A023        6210-00                              12/17/80 16:37  SMC                    NOT BACKED-UP
C0072272    A023        6210-00                              12/17/80 16:38  SMC                    NOT BACKED-UP
C0072410    A023        6210-00                              01/26/81 18:39  SMC                    NOT BACKED-UP
C0072597    A023        6210-00                              01/26/81 22:49  SMC                    NOT BACKED-UP
C0072151    A024        6210-00                              12/17/80 16:40  SMC                    NOT BACKED-UP
C0072156    A024        6210-00                              12/17/80 16:41  SMC                    NOT BACKED-UP
C0072400    A024        6210-00                              01/10/81 17:08  SMC                    NOT BACKED-UP
C0072401    A024        6210-00                              01/23/81 13:18  SMC                    NOT BACKED-UP
C0072459    A024        6210-00                              01/23/81 13:15  SMC                    NOT BACKED-UP
C0072452    A024        6210-00                              01/10/81 17:22  SMC                    NOT BACKED-UP
C0072462    A024        6210-00                              01/10/81 17:25  SMC                    NOT BACKED-UP
C0072580    A024        6210-00                              01/26/81 22:31  SMC                    NOT BACKED-UP
C0072352    A025        6210-00                              12/17/80 16:43  SMC                    NOT BACKED-UP
C0072487    A025        6210-00                              01/10/81 17:27  SMC                    NOT BACKED-UP
C0072577    A025        6210-00                              01/26/81 21:56  SMC                    NOT BACKED-UP
C0072517    A026        6210-00                              01/10/81 17:28  SMC                    NOT BACKED-UP
C0072720    A027        6210-00                              02/10/81 19:09  SMC                    NOT BACKED-UP
C0072730    A027        6210-00                              02/10/81 19:12  SMC                    NOT BACKED-UP
C0072105    A050        6210-00                              12/17/80 16:45  SMC                    NOT BACKED-UP
C0072173    A050        6210-00                              12/17/80 16:47  SMC                    NOT BACKED-UP
C0072350    A050        XXXX-XX                              01/23/81 14:07  SMC                    NOT BACKED-UP
C0072575    A050        6210-00                              01/26/81 21:43  SMC                    NOT BACKED-UP
C0072179    A060        6210-00                              12/18/80 14:11  SMC                    NOT BACKED-UP
C0072558    A060        6210-00                              01/26/81 20:35  SMC                    NOT BACKED-UP
C0072451    A090        6210-00                              01/10/81 17:31  SMC                    NOT BACKED-UP
C0072419    A110        6233-00                              01/10/81 17:33  SMC                    NOT BACKED-UP
C0072567    A110        6233-00                              01/26/81 21:16  SMC                    NOT BACKED-UP
C0072185    A120        6219-00                              12/18/80 14:12  SMC                    NOT BACKED-UP
C0072190    A120        6219-00                              12/18/80 14:15  SMC                    NOT BACKED-UP
C0072202    A120        6219-00                              12/18/80 14:18  SMC                    NOT BACKED-UP
C0072277    A120        6219-00                              12/18/80 14:20  SMC                    NOT BACKED-UP
C0072308    A120        6219-00                              12/18/80 14:23  SMC                    NOT BACKED-UP
C0072327    A120        6219-00                              12/18/80 14:24  SMC                    NOT BACKED-UP
C0072369    A120        6219-00                              12/18/80 14:27  SMC                    NOT BACKED-UP
C0072625    A120        6219-00                              01/27/81 00:29  SMC                    NOT BACKED-UP
C0072656    A120        6219-00                              02/10/81 18:02  SMC                    NOT BACKED-UP
C0072572    A121        6220-00                              01/26/81 21:37  SMC                    NOT BACKED-UP
C0072589    A121        6220-00                              01/26/81 22:36  SMC                    NOT BACKED-UP
C0072599    A121        6220-00                              01/26/81 23:21  SMC                    NOT BACKED-UP
```

Figure 13-1. Sample of Full SMC Listing (Part 2 of 2)

Field	Description
SMC NUMBER	Software maintenance correction number assigned.
COMP	Software component number (internal use only).
PP-TYPE NUMBER	Program product type number; e.g., control system (6210), etc.
SYSTEM	No entry indicates SMC is common to both models 6 and 8; 80 indicates SMC pertains only to models 8 through 20; 90 indicates SMC pertains only to model 6.
REQUIRED RE-GENS	YES indicates some SYSGEN was required; no entry indicates SYSGEN was not required.
APPLICATION DATE - TIME	Date and time SMC or SMP were applied.
METHOD	SMC applied via the librarian; SMP applied via dump/restore.
CHARACTER	No entry indicates SMC/SMP was required; OPTIONAL indicates SMC/SMP was not required.
SMC/SMP STATUS	NOT BACKED UP indicates SMC/SMP cannot be deleted but was successfully applied. NOT APPLIED,DEPENDENCY indicates other SMCs not contained on the volume were needed; SMC/SMP not applied. **SMC LOOPED indicates SMC not applied because specified time limit was exceeded for applying SMC. **SMC EXECUTION ERROR indicates SMC was not applied due to abnormal termination. ** RUN PROCESSOR ERROR indicates SMC not applied due to SMC not being scheduled.

```
OS/3 SMC LOG FILE  V=12.0.0     RELEASE-ID= 12.0.0 -A       FORMAT=C  SEQ1=COMP      SEQ2=SMC#        02/04/88      15:14      PAGE=0001

                        R - REPLACED      B - BACKED OUT      I - INFORMATION ONLY      F - OTHER ERROR

C0R1817   C0R1819   C0R1912   C0R2081   C0R2086   C0R2089   C0R2091   C0R1799   C0R1842-E C0R2041   C0R2044   C0R2056
C0R1957   C0R2029   C0R2045   C0R2067   C0R2134   C0R1815   C0R1821   C0R1836   C0R1857   C0R1925   C0R2118   C0R2132
C0R1906   C0R2028   C0R2000   C0R1919   C0R2034   C0R2035   C0R2068   C0R2113-R C0R1976   C0R1910-B C0R2042   C0R1928
C0R2107   C0R1885   C0R1887   C0R1893   C0R2027   C0R0757   C0R0150   C0R1790   C0R1813   C0R1830   C0R1856   C0R1865
C0R1866   C0R1896   C0R1915   C0R2006   C0R2046   C0R1779   C0R2122   C0R2008   C0R1835   C0R1924   C0R2088-B C0R2090
C0R2112   C0R1998   C0R2187-R C0R2270   C0R1999   C0R2017   C0R1804   C0R2049   C0R2050   C0R1801   C0R1926   C0R2032
C0R2071   C0R1907   C0R1760   C0R1859   C0R2014   C0R2098   C0R2156   C0R1769   C0R1870   C0R1759   C0R1809   C0R1883
C0R2025   C0R2064   C0R2152   C0R2234   C0R2271   C0R1710   C0R1780   C0R1903   C0R1785   C0R1757   C0R1852   C0R1690
C0R1959   C0R1776   C0R1869   C0R1917   C0R2005-B C0R1916   C0R1920   C0R2093   C0R2126   C0R2065   C0R1644   C0R1810
C0R2009   C0R2136   C0R2074   C0R1824   C0R0080   C0R2043   C0R2057   C0R2072   C0R2054   C0R2080   C0R2031   C0R1873
C0R1913   C0R2003   C0R1884   C0R2058-E C0R2059   C0R1775   C0R1827   C0R1849   C0R1891   C0R1892   C0R2030   C0R1798
C0R1847   C0R1888   C0R2026   C0R2021   C0R2022   C0R2047   C0R1863   C0R2019   C0R2020   C0R2051   C0R2052   SM08A
BACKUP -I C0R2092   C0R0576   XESC-EN-I X460-00-I X60C-10-I X600-20-I X600-30-I X600-50-I X600-60-I X600-80-I X700-00-I
X80C-CC-I 6130-00-I 6130-03-I 6201-00-I 6201-03-I 6210-00-I 6211-00-I 6212-C0-I 6213-00-I 6214-00-I 6215-00-I 6216-00-I
6217-00-I 6218-00-I 6219-00-I 6220-00-I 6221-00-I 6222-00-I 6222-01-I 6223-00-I 6224-00-I 6225-00-I 6226-00-I 6228-00-I
6229-01-I 6229-02-I 6229-03-I 6230-00-I 6231-00-I 6232-00-I 6233-00-I 6247-00-I 6247-01-I 6247-02-I 6248-00-I 6248-01-I
6248-02-I 6248-03-I 6248-04-I 6248-05-I 6248-06-I 6248-07-I 6254-00-I 6255-00-I C0R1853   C0R2150
```

Figure 13-2. Sample of Condensed SMC Listing

Section 14
Diskette Copy Routine (SU$CPY)

The diskette copy routine (SU$CPY) can generate up to three copies of an 8420 diskette, whether it is data set label or format label, single or double density. The SU$CPY routine also gives you the option of verifying individual 8420 diskettes once you've copied them. All you have to do to execute SU$CPY is key in a simple command at the system console or workstation. Once you've entered this command, a series of messages appear on the screen requesting you to select certain options.

14.1. Executing SU$CPY

Mount the diskette you are copying (input) in a drive with a lower device-id number than the output diskette's drive.

The format of the command used to execute SU$CPY is

```
RV SU$CPY,,N={1}
             {2}
             {3}
```

where:

```
N={1}
  {2}
  {3}
```

Specifies the number of copies you're making.

During the execution of the SU$CPY routine, you are asked to reply to the following messages:

Message 1

```
jji=NUMBER OF COPIES REQUIRED OF OUTPUT 2,3 DEFAULT=1
```

Reply

Press the XMIT key to default to the number of copies you originally requested in the RV SU$CPY command. If you want to change the number of copies you originally requested, enter the job number and message-id (*jji*) and then the number of the copies wanted (1, 2, or 3). Press the XMIT key. The number specified overrides the one that was originally specified in the RV SU$CPY command. Message 2 appears on the screen after you transmit your reply.

Message 2

```
jji=TYPE OF OPERATION: COPY, VERIFY
```

Reply

You should reply by entering the job number and message-id and one of the following specifications:

C
: Indicates that you are making copies.

V
: Indicates that you are verifying a diskette.

If no reply is given to this message and you press the XMIT key, the C option is assumed and message 3 appears on the screen.

Message 3

```
jji=N=NEXT DISKETTE MOUNTED OR DEFAULT=FINAL DISKETTE
```

Reply

If more diskettes are to be copied, remove the current diskette, mount the next diskette to be copied, and reply by keying in N along with the job number and message-id. This resumes the message sequence starting at message 2.

To terminate the job, press the XMIT key instead of replying to the message.

If you want to verify a copy, key in the N parameter and press XMIT. This resumes the message sequence starting at message 2. Then you can simply key in V and press the XMIT key to verify the copy. After verification, message 3 again appears on the screen.

Note: *Only one diskette can be verified each time you execute SU$CPY. So, if you are making more than one copy in a single execution of SU$CPY, only the first copy can be verified. To verify the remaining copies, you must run separate SU$CPY operations using the V (verify) option for each individual copy.*

14.2. Programming Examples

The following are examples of system console listings showing the execution of the SU$CPY routine:

Example 1: Diskette Copy

```
1.  RV SU$CPY
2.   SU$CPY     (2)      (3)      (4)      (5)      (6)      (7)
3.  1L JC08  USING   DEV=FFF TYPE=PRNTR
4.  1M*JC09  MOUNT   DEV=320 VSN=IN        DEV=321 VSN=OUT      GO?
5.  GO SU$CPY
6.  1N JC01  JOB SU$CPY   EXECUTING JOB STEP SU$CPY00 #001
7.  1P?NUMBER OF COPIES REQUIRED OF OUTPUT 2,3 DEFAULT=1
8.  1P
9.  Q?  TYPE  OF  OPERATION:  COPY,VERIFY
10. 1Q
11. 1A?N=NEXT DISKETTE MOUNTED OR DEFAULT=FINAL DISKETTE
12.  1A
    1B      END OF STANDALONE COPY ROUTINE
       (1)      (2)      (3)      (4)      (5)      (6)      (7)
    1C JC02  JOB SU$CPY    TERMINATED NORMALLY
    BR CN
```

Here, you are making a single copy of an 8420 diskette. The RV SU$CPY command on line 1 executes the SU$CPY routine, and line 2 shows the job slot in which the copy routine is running. Note that no N= specification is given in the RV SU$CPY command to specify the number of output copies. Since no number is specified, the default of 1 is used.

The next four lines (lines 3, 4, 5, and 6) are messages showing the devices being used and indicating the start of the routine's execution message sequence. The message in line 7 is the first message of the sequence. It requests you to enter the number of output copies. In this example, no reply is given in line 8 and the default of 1 is used.

The XMIT key is then pressed and the message in line 9 appears requesting you to specify either C (for copy) or V (for verify). In line 10, no reply is given and the XMIT key is pressed. This causes the default C to be initiated. The diskette is then copied and the next request message is generated in line 11.

The final message of the sequence (line 11) requests you to either mount another diskette, generate another TYPE OF MESSAGE request (line 9), or terminate the job. Here, because no further copying or verifying is desired, nothing is entered in line 12 and the XMIT key is pressed, initiating the default and terminating the job.

Example 2: Diskette Verify

```
1.  RV SU$CPY
2.   SU$CPY     (2)      (3)      (4)      (5)      (6)      (7)
3.  1E JC08  USING   DEV=FFF TYPE=PRNTR
4.  1F*JC09  MOUNT   DEV=320 VSN=IN        DEV=321 VSN=OUT      GO?
5.  GO SU$CPY
6.  1G JC01  JOB SU$CPY   EXECUTING JOB STEP SU$CPY00 #001
7.  1H?NUMBER;OF;COPIES;REQUIRED;OF;OUTPUT;2,3;DEFAULT=1
8.   1H
9.  1J?  TYPE  OF  OPERATION:  COPY,VERIFY
10.  1J;V
11. 1K?N=NEXT DISKETTE MOUNTED OR DEFAULT=FINAL DISKETTE
12.  1K
    1L      END OF STANDALONE COPY ROUTINE
      (1)      (2)      (3)      (4)      (5)      (6)      (7)
    1M JC02  JOB SU$CPY    TERMINATED NORMALLY
    BR CN
```

The console listing in this example is exactly the same as the one shown in the diskette copy operation in Example 1. However, because this is a verify operation, the reply to the TYPE OF OPERATION request (line 9) is V (for verify). This reply is shown in line 10.

Example 3: Diskette Copy and Verify

```
1.  RV SU$CPY
2.   SU$CPY     (2)      (3)      (4)      (5)      (6)      (7)
3.  1Q JC08  USING   DEV=FFF TYPE=PRNTR
4.  1A*JC09  MOUNT   DEV=320 VSN=IN        DEV=321 VSN=OUT      GO?
5.  GO SU$CPY
6.  1B JC01  JOB SU$CPY   EXECUTING JOB STEP SU$CPY00 #001
7.  1C?NUMBER OF COPIES REQUIRED OF OUTPUT 2,3 DEFAULT=1
8.   1C
9.  1D?  TYPE  OF  OPERATION:  COPY,VERIFY
10.  1D
11. 1E?N=NEXT DISKETTE MOUNTED OR DEFAULT=FINAL DISKETTE
12.  1E N
13. 1F?  TYPE  OF  OPERATION:  COPY,VERIFY
14.  1F V
15.  G?N=NEXT DISKETTE MOUNTED OR DEFAULT=FINAL DISKETTE
16.  1G
    1H      END OF STANDALONE COPY ROUTINE
      (1)      (2)      (3)      (4)      (5)      (6)      (7)
    1J JC02  JOB SU$CPY    TERMINATED NORMALLY
    BR CN
```

Example 3 is similar to the diskette copy operation in Example 1. Here, however, a copy and verify operation is performed in the same execution. The first part of the control stream (lines 1 through 10) performs the diskette copy.

The difference between this example and Example 1 begins in line 11 with the request to either mount another output disk, generate a TYPE OF OPERATION request, or terminate the job. In this example, the letter N is keyed in and the XMIT key is pressed (line 12). This generates another TYPE OF OPERATION request as shown in line 13.

In line 14, a reply of V (for verify) is entered and the XMIT key is pressed. This verifies the same output disk that was created in the copy operation and generates the message shown in line 15.

Since no further copies are being made or verified, the default is used as the reply to this message (line 16), the XMIT key is pressed and the job is terminated.

Section 15
System Utility Copy Routines

15.1. 8419 Disk Copying (SU$C19)

Use the SU$C19 disk copy routine to simultaneously create and verify up to six copies of a single 8419 disk, regardless of the disk's contents. The verification procedure ensures that your disk was copied correctly by comparing the contents of the input with that of the output. You can also use the SU$C19 routine in a separate operation to verify copies that were made by a previous SU$C19 copy operation or a disk copy operation performed using the dump/restore routine. You can execute the SU$C19 routine either interactively from a workstation or in a batch environment. Either method achieves the same results; however, the interactive method is easier to use.

15.1.1. Executing SU$C19 in an Interactive Environment

Interactive processing of the SU$C19 routine consists of a question and answer session (dialog). Before executing SU$C19, you must have successfully logged on to the system via the LOGON command. After logging on, key in HU in system mode and press the XMIT key. After pressing the XMIT key, a menu screen appears.

```
                        HARDWARE UTILITIES          HU00C

    1. DUMP FILES FROM A DISK
    2. RESTORE FILES TO A DISK
    3. LIST FILES ON A BACKUP MEDIUM
    4. COPY FILES FROM DISK TO DISK
    5. COPY AND/OR VERIFY 8419 DISK
    6. COPY AND/OR VERIFY 8416/8418 DISK
    7. COPY AND/OR VERIFY 8430/8433 DISK
    8. NONE OF THESE

                        ENTER SELECTION 5
```

Note: *Selections 6 and 7 are applicable only to model 8 through 20 systems and do not appear in the menu screen (HU00A) displayed on model 3 through 6 systems.*

Select 5 from the menu screen to start the execution of the SU$C19 routine. After pressing XMIT, the following informational message screen appears:

```
                                                          HU00BI03
A CONVERSATIONAL JOB (HU$CPY) TO COPY FILES FROM ONE DISK TO
ANOTHER WILL BE INITIATED IN YOUR BEHALF. YOU MUST BE IN SYSTEM
MODE FOR THE JOB TO BE SCHEDULED. IF YOU ENTERED HARDWARE
UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN SYSTEM MODE
AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU COMMAND YOU
ARE RESPONSIBLE FOR GOING INTO SYSTEM MODE.
              *****  TRANSMIT TO CONTINUE  *****
```

After reading the message screen indicating that a copy job was initiated, press XMIT. Screen 1 is displayed.

Screen 1 (HU13): Number of Copies, Starting/Ending Addresses, and Verification Option

```
                          COPY-VERIFY                  HU13

COPIES  -  UP TO 6                          COPY=1

BEGIN ADDRESS (CCC)                         BGAD=000

END ADDRESS (CCC)                           EDAD=808

ONLY VERIFY                                 OVEF=NO

******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

Looking at screen 1, we used the default 1 for the number of copies, took the default for the beginning and ending addresses (cylinder 000 for beginning and 808 for ending), and entered NO indicating that we are in a "copy" operation and not a "verify-only" operation. Since help is not required, press XMIT.

Screen 2 (HU14): File Options

```
                               COPY              HU14

UNEXPIRED FILE CHECKING                    UNXF=YES

VERIFY                                     VEFY=YES

******** FUNCTION KEYS: F13=HELP,F14=EXIT HELP ********
```

On this screen, we want to enter unexpired file checking and output verification. The unexpired file checking (UNXF=YES, default) checks the output volume for file expiration dates. If the date has not expired, a message is displayed in system mode of the workstation asking you to either skip the file and continue processing or terminate. Up to 10 files at a time can be skipped. (The expiration option must not be used with unprepped output disk packs. To do so results in a space management error.) We specified VEFY=YES for file verification (overriding the default). Verification ensures that an exact copy of your input is produced. Any errors detected will be printed using the print option (next screen).

Screen 3 (HU15): Print Option

```
                          COPY-VERIFY              HU15

PRINT ALL ERRORS                                   PRNT=YES
```

Since we specified the verify option on the last screen, we recommend that you take the default YES to print any errors detected during the copy process. Press XMIT.

Screen 4 (HU01): Input and Output Device Information

```
                COPY INPUT-OUTPUT DEVICE INFORMATION        HU01

ENTER INPUT VOLUME SERIAL NUMBER: REL120

ENTER OUTPUT VOLUME SERIAL NUMBERS

                  D00410
```

Here, we enter our volume serial numbers (DISKIN for INPUT and DISK01, DISK02, and DISK03 for three copies of our output). Press XMIT to complete the copy operation. After pressing XMIT, the workstation becomes free for other uses and the HU25 HARDWARE UTILITIES information screen appears.

```
                     HARDWARE UTILITIES          HU25
                THE INTERACTIVE INTERFACE FOR THIS PROGRAM
            HAS NOW BEEN COMPLETED. THIS WORKSTATION WILL BE
            AVAILABE FOR USE WHILE THE PROGRAM PROCESSES AS
            YOU REQUESTED.
```

When the job is finished, an end-of-job message is displayed on our workstation screen, showing the parameter selected and the UPSI byte setting. If the setting is other than X'00', errors occurred and are reported.

```
***SYSTEM UTILITY COPY ROUTINE FOR DISKS  8419 **** **** **

// PARAM COPY=1,BGAD=000,EDAD=808,OVEF=NO,UNXF=YES,VEFY=YES

SC50 UPSI SETTING  X'00'
```

Note: After copying, our output volume serial number is the same as our input volume serial number (REL120). If we want the original serial number (D00410), then we can execute the CGV canned control stream to change the volume serial number (see Section 9).

15.1.2. Executing SU$C19 in a Batch Environment

When executing SU$C19 in a batch environment, you must supply the appropriate job control statements defining your devices. All the options available to you in an interactive environment are supported in batch.

SU$C19 Organization

SU$C19 has seven parameters that control its operation. All of them have default values and are associated with the // PARAM statement. If you choose all the defaults, however, omit the // PARAM statement from your control stream.

The format of the SU$C19 // PARAM statement is

```
// PARAM [ COPY= {n} ][ ,VEFY= {NO } ] [ PRNT= { NO } ] [ ,OVEF= {NO } ][ ,BGAD= {ccc} ]
                  {1}        {YES}          { YES}          {YES}          {000}
      [ ,EDAD= {ccc} ][ ,UNXF= {YES} ]
               {808}          {NO }
```

Use the COPY parameter to indicate how many copies of your input disk are to be made or verified when using the OVEF parameter. The value of *n* may vary from 1 to 6. If you omit the COPY parameter, one copy is specified by default.

Use the VEFY parameter in conjunction with the COPY parameter to verify the contents of each new output disk. If you specify VEFY=YES, each copy is verified against the input. If you omit the VEFY parameter, no verification is performed.

Use the PRNT parameter to print any incorrect records found during verification. If an error is detected, both the input and output records are listed with their corresponding disk address. A sample verification printout, showing input and output records, is shown in Figure 15-1. If you omit the PRNT parameter, messages are displayed on the system console, instead of the printer.

Figure 15-1. Sample Verification Printer Output

Use the OVEF parameter to indicate a verify-only operation is performed on an output disk created by a previous copy operation. When OVEF=YES is specified, no copies are made. However, the COPY parameter or its default is used to indicate the number of output disk volumes being verified. The OVEF parameter may be used in conjunction with the BGAD and EDAD parameters, but cannot be specified if VEFY=YES is specified in the same control stream. The default of the OVEF parameter is NO.

Use the BGAD parameter to indicate the beginning cylinder address, in decimal, for duplication or verification. The address you specify must be the cylinder number (*ccc*). If you omit the BGAD parameter, the beginning address is cylinder 000.

Use the EDAD parameter to indicate the ending cylinder address, in decimal, for duplication or verification. The address you specify must be the cylinder number (*ccc*). If you omit the EDAD parameter, the ending address is 808.

Use the UNXF parameter to check for the file expiration date. If the file date has not expired, a message is displayed asking you either to continue processing or to skip the file and continue to the next file (providing you are processing more than one file). Using the UNXF parameter eliminates the time and expense of keeping outdated files and also eliminates the overlaying of files that should not be destroyed.

Note: The SU$C19 program will not allow you to copy less than one cylinder of data.

SU$C19 Interfacing with Job Control

Specify DISCIN on the // LFD job control statement for your input disk, and DISCOT on the // LFD job control statement for your output disk. If you are copying onto more than one disk, DISCOT01 through DISCOT05 must be specified for each disk, following the first one.

Programming Examples

The following control streams show some typical examples of how to use SU$C19:

Example 1

```
// JOB STDCOPY
// DVC 50 // VOL DSK001 // LFD DISCIN
// DVC 51 // VOL DSK002 // LFD DISCOT
// EXEC SU$C19
/&
// FIN
```

This is a basic 8419 disk copy operation with no verification. Since no printer is specified in your device assignments, no printer output is available. Notice that, since no parameters are specified, the // PARAM statement is omitted from your control stream. Here, a single copy is made.

Example 2

```
// JOB VFYCOPY
// DVC 50 // VOL DSK001 // LFD DISCIN
// DVC 51 // VOL DSK002 // LFD DISCOT
// DVC 20 // LFD PRNTR
// EXEC SU$C19
// PARAM VEFY=YES,PRNT=YES
/&
// FIN
```

This is an 8419 disk copy operation with verification. Since you are using the print option (PRNT=YES), the device assignment set for the printer must be specified. The file name PRNTR must appear in the // LFD statement for the printer. If you specify verification (VEFY=YES) and do not specify a printer, your job is executed, but errors are displayed on the system console.

Example 3

```
// JOB COPYSIX
// DVC 50 // VOL DSK000 // LFD DISCIN
// DVC 51 // VOL DSK001 // LFD DISCOT
// DVC 52 // VOL DSK002 // LFD DISCOT01
// DVC 53 // VOL DSK003 // LFD DISCOT02
// DVC 54 // VOL DSK004 // LFD DISCOT03
// DVC 55 // VOL DSK005 // LFD DISCOT04
// DVC 56 // VOL DSK006 // LFD DISCOT05
// DVC 20 // LFD PRNTR
// EXEC SU$C19
// PARAM COPY=6,VEFY=YES,PRNT=YES,UNXF=YES
/&
// FIN
```

Here, you are copying an 8419 input disk to six 8419 output disks with verification, printing, and file expiration date checking. Note the required file names (DISCOT through DISCOT05) specified on the // LFD job control statements.

15.2. 8416/8418 Disk Copying (SU$C16)

Use the SU$C16 disk copy routine to simultaneously create and optionally verify up to seven copies of a single 8416 or 8418 disk, regardless of the disk's contents. The verification process ensures that your disk was copied correctly by comparing the contents of the input with that of the output. You can use this routine in a separate operation to verify copies that were made by a previous SU$C16 disk copy operation or a disk copy operation using the dump/restore routine.

You can execute the SU$C16 routine either interactively from a workstation or in a batch environment. Both methods achieve the same results.

15.2.1. Executing SU$C16 in an Interactive Environment

Interactive processing of the SU$FC16 routine consists of a question and answer session (dialog). Before executing SU$C16, you must have successfully logged on to the system via the LOGON command. After logging on, key in HU in system mode and press XMIT. After pressing XMIT, the following menu screen appears:

```
                                HARDWARE UTILITIES          HU000

    1. DUMP FILES FROM A DISK
    2. RESTORE FILES TO A DISK
    3. LIST FILES ON A BACKUP MEDIUM
    4. COPY FILES FROM DISK TO DISK
    5. COPY AND/OR VERIFY 8419 DISK
    6. COPY AND/OR VERIFY 8416/8418 DISK
    7. COPY AND/OR VERIFY 8430/8433 DISK
    8. NONE OF THESE

    ENTER SELECTION      6
```

Select 5 from the menu screen to start execution of the SU$C16 routine

After pressing XMIT, the following HU00CI05 information screen is displayed:

```
                                HU00CI05

    A CONVERSATIONAL JOB (HU$C16) TO COPY THE CONTENTS OF ONE 8416/
    8418 DISK TO ONE OR MORE DISKS OR TO VERIFY THE CONTENTS OF ONE
    OR MORE DISKS HAS BEEN INITIATED IN YOUR  BEHALF. YOU MUST BE
    IN SYSTEM MODE FOR THE JOB TO BE SCHEDULED. IF YOU ENTERED
    HARDWARE UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN SYSTEM
    MODE AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU
    COMMAND YOU ARE RESPONSIBLE FOR GOING INTO SYSTEM MODE.

                    ***** TRANSMIT TO CONTINUE *****          X
```

After reading the message screen indicating that a copy job was initiated, press XMIT. The following screen appears:

Screen 1 (HU09): Indicating the Disk Device Type and Volume Serial Number

```
                        COPY INPUT DEVICE INFORMATION          HU09

    ENTER SPECIFIC DISK DEVICE TYPE:  8416

    ENTER VOLUME SERIAL NUMBER:  123456

    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

Looking at Screen 1, we entered 8416 for the disk device type and 123456 for the volume serial number. The device type entered on this screen must be one of two disks: 8416 or 8418.

Note: Regarding the screens, all the default values are shaded while the user entries are shown in reverse lettering (white characters on black background).

Since help is not required, press XMIT and the following screen appears:

Screen 2 (HU13): Indicating Number of Copies, Begin/End Addresses, and Verification Option

```
                              COPY-VERIFY                    HU13

COPIES - UP TO 7                                        COPY= 3

BEGIN ADDRESS (CCC)                                     BGAD=000

END ADDRESS (CCC)                                       EDAD=193

ONLY VERIFY                                             OVEF=NO

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

Looking at Screen 2, we entered 3 for the number of copies (overriding the default of 1). We took the default values, which appear on the screen, for the beginning and ending addresses (cylinder 000 for beginning and 193 for ending), as well as the default option NO, indicating that we are doing a "copy" operation and not a "verify-only" operation. Since help is not required, press XMIT.

Screen 3 (HU14): Indicating the File Options

```
                              COPY           HU14

UNEXPIRED FILE CHECKING                 UNXF=YES

VERIFY                                  VERY=YES

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we want unexpired file checking and output verification. We entered YES for output verification and allowed the unexpired file checking option (UNXF) to default to YES. Unexpired file checking checks the output volume for file expiration dates. If the date has not expired and your workstation is in system mode, your screen displays a message. This message asks you either to terminate or to skip the file and continue processing. You can skip up to 10 files at a time. (The expiration option must not be used with unprepped output disk packs. To do so results in a space

management error.) Verification ensures that an exact copy of your input is produced. Any errors detected are printed by using the print option (next screen).

Screen 4 (HU15): Indicating the Print Option

```
                          COPY-VERIFY                        HU15

PRINT ALL ERRORS                                          PRNT=YES
```

Since we specified the verify option on the last screen, we recommend that you take the default YES to print any errors detected during the copy process. Press XMIT.

Screen 5 (HU11): Specifying Output Device Information

```
                     COPY OUTPUT DEVICE INFORMATION          HU11

ENTER SPECIFIC DISK DEVICE TYPE: 8416

ENTER VOLUME SERIAL NUMBER(S):

  DISK01   DISK02   DISK03

******** FUNCTION KEYS: F13=HELP, F14=EXIT HELP ********
```

On this screen, we entered 8416 as our output disk device type. Next, we entered our output volume serial numbers. The number of underscored input fields requesting volume serial numbers that appear on the screen are equal to the number of copies specified on Screen 2 (HU13). All fields that appear must be filled in. Since we specified three copies, fields for three volume serial numbers appear on the screen. We entered volume serial numbers DISK01, DISK02, and DISK03 for the three copies of our output. Press XMIT to complete the copy operation. After you press XMIT, the workstation becomes free for other uses and the following HU25 screen appears. When the job is finished, your workstation screen displays an end-of-job message.

```
                          HARDWARE UTILITIES          HU25

    THE INTERACTIVE INTERFACE FOR THIS PROGRAM
HAS NOW BEEN COMPLETED. THIS WORKSTATION WILL BE
AVAILABLE FOR USE WHILE THE PROGRAM PROCESSES AS
YOU REQUESTED.
```

15.2.2. Executing SU$C16 in a Batch Environment

When executing SU$C16 in a batch environment, you must supply the appropriate job control statements defining your devices. All the options available to you in an interactive environment are supported in batch.

SU$C16 Organization

SU$C16 has seven parameters that control its operation. All these parameters have default values and are associated with the // PARAM statement. However, if you choose all of the defaults, you must still have the // PARAM statement present in your control stream.

The format of SU$C16 is

```
// PARAM [COPY= {n}  ][,VEFY= {NO }][ ,PRNT= {NO }][ ,OVEF= {NO }]
         [    {1}  ]        {YES}        {YES}        {YES}

      [,BGAD= {ccc16  }][,EDAD= {ccc16                          }][ ,UNXF ={NO }]
              {cccch16}         {cccch16                        }        {YES}
              {cccchrr16}       {cccchrr16
              {000001   }       {193628 for 8416 and 8418 low
                                {327628 for 8418 high
```

Use the COPY parameter to indicate how many copies of your input disk pack are to be made. The value of *n* may be from 1 to 7. If you omit COPY, one copy is made.

Use the VEFY parameter to verify your new output. If VEFY=YES is specified, each copy is verified against the input. If you omit VEFY, then no verification is performed.

Use the PRNT parameter to print any records found to be in error. If an error is detected, both the input and output records are listed with its corresponding output disk address (*ccchrr*) in hexadecimal. You would normally choose PRNT=YES if you had chosen VEFY=YES. If you omit PRNT, only fatal errors are written on the system console.

Use the BGAD parameter to indicate the starting address of your copy in hexadecimal. The address may be a cylinder number (ccc_{16}), a cylinder/head number ($ccch_{16}$), or a cylinder/head/record number ($ccchrr_{16}$). If you omit BGAD, the beginning address is cylinder 000, head 0, and record 01.

Use the OVEF parameter to indicate that a verify-only operation is to be performed. No copy will be made. If you omit OVEF, one or more copies will be made, with or without specification, depending on other parameter specifications. OVEF may be used in conjunction with BGAD and EDAD.

Use the EDAD parameter to indicate the ending address of your copy in hexadecimal (ccc_{16}, $ccch_{16}$, or $cchrr_{16}$). If you omit EDAD, the default values for the hexadecimal ending addresses are as follows:

Disk Subsystem	Cylinder	Head	Record
8416	193	6	28
8418 low	193	6	28
8418 high	327	6	28

By using both the BGAD and EDAD parameters, you can copy only one record from your input pack to your output pack. It is worth remembering that the input starting and ending addresses are the same as your output addresses, and any information contained in your specified output area is destroyed and the new information is written in that area. For example, if you were copying from cylinders 10 through 20, any information residing in cylinders 10 through 20 in your output pack is destroyed and the new information is written in that area.

Use the UNXF parameter to check for the file expiration date of any file on the output disks. If the file date has not expired, a message is displayed that asks you either to continue processing or to skip the file and continue to the next file (providing you are processing more than one file). Using the UNXF parameter eliminates the time and expense of keeping outdated files. It also eliminates the overlaying of files that should not be destroyed.

SU$C16 Interfacing with Job Control

The file name DISCIN must be specified on the // LFD job control statement for your input disk pack. The file name DISCOT must be specified on the // LFD job control statement for your output disk pack. If you are copying onto more than one disk pack, DISCOT01 through DISCOT06 must be specified for each disk pack being copied.

Executing SU$C16

In a multijob environment, the use of SU$C16 can cause space allocated to a file to be lost; files may be added to or portions deleted, or a disk pack may be incorrectly copied. These situations can occur when another job is updating the VTOC of a disk pack at the same time SU$C16 is copying that VTOC to another disk pack, or when SU$C16 is copying a file while another job is extending or scratching that file. Therefore, the job executing SU$C16 must be the only job running when the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonshareable by the operator. This ensures that an exact copy will be made by SU$C16.

Programming Examples

The following control streams show some typical examples of how to use SU$C16:

Example 1

```
1        10    16                                                    72
// JOB STDCOPY
// DVC 60  // VOL DSP001 // LFD DISCIN
// DVC 61  // VOL DSP002 // LFD DISCOT
// EXEC SU$C16
// PARAM
/&
// FIN
```

Note that even though you did not specify any parameters, the // PARAM statement still appears in your control stream. This is the basic disk copy operation. The printer is not specified; therefore, no printer output is available.

Example 2

```
1        10    16                                                    72
// JOB VPCOPY
// DVC 50 // VOL DSP001 // LFD DISCIN
// DVC 51 // VOL DSP002 // LFD DISCOT
// DVC 20 // LFD PRNTR
// EXEC SU$C16
// PARAM VEFY=YES,PRNT=YES
/&
// FIN
```

Since you are using the print option, the device assignment set for the printer must be specified. The file name PRNTR must be specified on the // LFD job control statement. The verification and printing are specified. If you do not specify a printer, your job is executed but only fatal errors detected are displayed on the system console.

Example 3

```
1        10    16                                                    72
// JOB COPY7
// DVC 50 // VOL DSP001 // LFD DISCIN
// DVC 51 // VOL DSP002 // LFD DISCOT
// DVC 52 // VOL DSP003 // LFD DISCOT01
// DVC 53 // VOL DSP004 // LFD DISCOT02
// DVC 54 // VOL DSP005 // LFD DISCOT03
// DVC 55 // VOL DSP006 // LFD DISCOT04
// DVC 56 // VOL DSP007 // LFD DISCOT05
// DVC 57 // VOL DSP008 // LFD DISCOT06
// DVC 20 // LFD PRNTR
// EXEC SU$C16
// PARAM COPY=7,VEFY=YES,PRNT=YES,UNXF=YES
/&
// FIN
```

Here, you are copying your input disk pack to seven output packs with verification, printing, and file expiration date checking. Note the required file names specified on the // LFD job control statements.

Example 4

```
1        10    16                                                    72
// JOB COPYTRK
// DVC 20 // PRNTR
// DVC 60 // VOL DSP001 // LFD DISCIN
// DVC 61 // VOL DSP002 // LFD DISCOT
// DVC 62 // VOL DSP003 // LFD DISCOT01
// EXEC SU$C16
// PARAM COPY=2,BGAD=0054,EDAD=0054,VEFY=YES
/&
// FIN
```

This example makes two copies of one track and verifies the copies. There is no printer output and verification of the output is written to the console.

Example 5

```
1        10    16                                                    72
// JOB COPYCYL
// DVC 60 // VOL DSP001 // LFD DISCIN
// DVC 61 // VOL DSP002 // LFD DISCOT
// EXEC SU$C16
// PARAM BGAD0=7A,EDAD=07A
/&
// FIN
```

This example makes one copy of one cylinder. There is no printer output and no verification.

15.3. 8430/8433 Disk Copying (SU$CSL)

Use the SU$CSL disk copy routine to simultaneously create and optionally verify up to seven copies of a single 8430 or 8433 disk, regardless of the disk's contents. The verification process ensures that your disk was copied correctly by comparing the contents of the input with that of your output. You can also use this routine in a separate operation to verify copies that were made by a previous SU$CSL disk copy operation or a disk copy operation using the dump/restore routine.

You can execute SU$CSL either interactively from a workstation or in a batch environment. Both methods achieve the same results.

15.3.1. Executing SU$CSL in an Interactive Environment

Interactive processing of the SU$CSL routine consists of a question and answer session (dialog). Before executing SU$CSL, you must have successfully logged on to the system via the LOGON command. After logging on, key in HU in system mode and press XMIT. After pressing XMIT, the following menu screen appears:

```
                    HARDWARE UTILITIES     HU00C

          1. DUMP FILES FROM A DISK
          2. RESTORE FILES TO A DISK
          3. LIST FILES ON A BACKUP MEDIUM
          4. COPY FILES FROM DISK TO DISK
          5. COPY AND/OR VERIFY 8419 DISK
          6. COPY AND/OR VERIFY 8416/8418 DISK
          7. COPY AND/OR VERIFY 8430/8433 DISK
          8. NONE OF THESE

                    ENTER SELECTION 7
```

Select 7 from the menu screen to start execution of the SU$CSL routine. After you press XMIT, the following HU00CI06 informational screen appears:

```
                                                          HU00CI06

A CONVERSATIONAL JOB (HU$CSL) TO COPY THE CONTENTS OF ONE
8430/8433 DISK TO ONE OR MORE DISKS OR TO VERIFY THE CONTENTS OF
ONE OR MORE DISKS HAS BEEN INITIATED IN YOUR BEHALF.
YOU MUST BE IN SYSTEM MODE FOR THE JOB TO BE SCHEDULED. IF YOU
ENTERED HARDWARE UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN
SYSTEM MODE AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU
COMMAND YOU ARE RESPONSIBLE FOR GOING INTO SYSTEM MODE.

                  *****  TRANSMIT TO CONTINUE  *****        X
```

After reading the message screen indicating that a copy job was initiated, press XMIT. Screen 1 is displayed.

Screen 1 (HU10): Indicating Disk Device Type and Volume Serial Number

```
                    COPY INPUT DEVICE INFORMATION          HU10

ENTER SPECIFIC DISK DEVICE TYPE:   8433

ENTER VOLUME SERIAL NUMBER:   ABC123

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

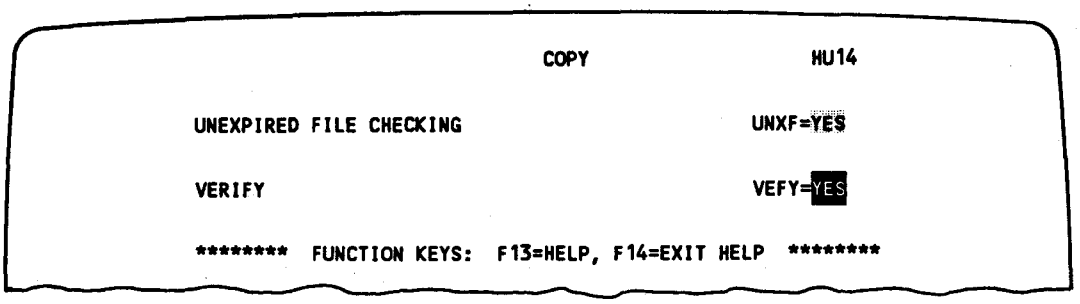
Looking at Screen 1, we entered 8433 for the disk device type and ABC123 for the volume serial number. The device type entered on this screen can be an 8430 or 8433 disk pack. Since help is not required, press XMIT and the following screen appears:

Screen 2 (HU13): Indicating Number of Copies, Start/End Addresses, and Verification Option

```
                              COPY-VERIFY                  HU13

COPIES - UP TO 7                                    COPY=5

BEGIN ADDRESS (CCC)                                 BGAD=000

END ADDRESS (CCC)                                   EDAD=327

ONLY VERIFY                                         OVEF=NO

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

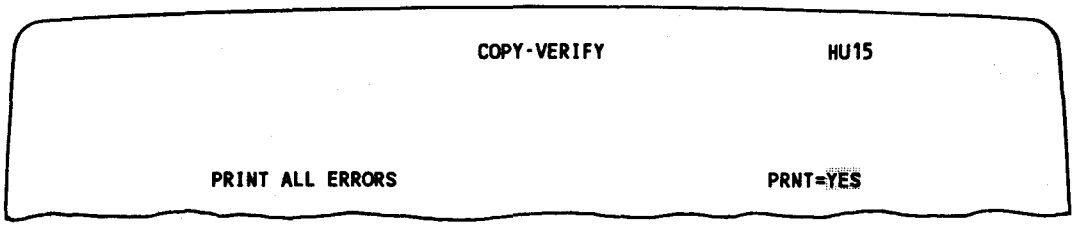
Looking at Screen 2, we entered 3 for the number of copies (overriding the default of 1), took the default for the beginning and ending addresses (cylinder 000 for beginning and 327 for ending), and took the default option NO, indicating that we are doing a "copy" operation and not a "verify-only" operation. Since help is not required, press XMIT.

Screen 3 (HU14): Indicating the File Options



On this screen, we want unexpired file checking and output verification. We entered YES for output verification and allowed the unexpired file checking option (UNXF) to default to YES. Unexpired file checking checks the output volume for file expiration dates. If the date has not expired and you are in system mode, the workstation screen displays a message. This message asks you either to terminate or to skip the file and continue processing. You can skip up to 10 files at a time. (The expiration option must not be used with unprepped output disk packs. To do so results in a space management error.) Verification ensures that an exact copy of your input is produced. Any errors detected are printed by using the print option (next screen).

Screen 4 (HU15): Indicating the Print Option



Since we specified the verify option on the last screen, we recommend that you take default YES to print any errors detected during the copy process. Press XMIT.

Screen 5 (HU11): Specifying Output Device Information

```
                    COPY OUTPUT DEVICE INFORMATION          HU11

ENTER SPECIFIC DISK DEVICE TYPE:  8433

ENTER VOLUME SERIAL NUMBER(S):

    DISK04      DISK05      DISK06

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we entered 8433 as our output disk device type. Next, we entered volume serial numbers DISK04, DISK05, and DISK06 for the three copies of our output. The number of underscored input fields requesting volume serial numbers that appear on the screen are equal to the number of copies specified on Screen 2. Press XMIT to complete the copy operation. After you press XMIT, the workstation becomes free for other uses and the following informational message screen (HU25) appears. When the job is finished, your workstation screen displays an end-of-job message.

```
                          HARDWARE UTILITIES      HU25

        THE INTERACTIVE INTERFACE FOR THIS PROGRAM
    HAS NOW BEEN COMPLETED. THIS WORKSTATION WILL BE
    AVAILABLE FOR USE WHILE THE PROGRAM PROCESSES AS
    YOU REQUESTED.
```

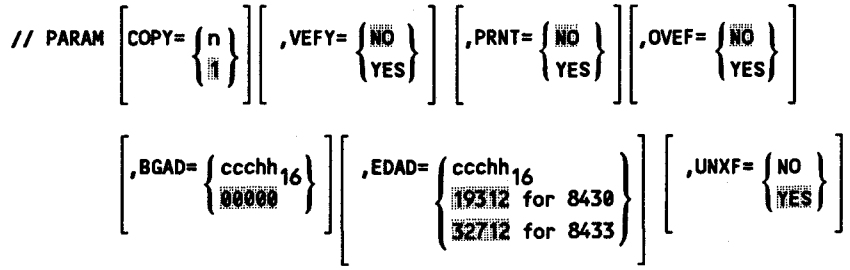
15.3.2. Executing SU$CSL in a Batch Environment

When executing SU$CSL in a batch environment, you must supply the appropriate job control statements defining your devices. All the options available to you in an interactive environment are supported in batch.

SU$CSL Organization

SU$CSL has seven parameters that control its operation. All these parameters have default values and are associated with the // PARAM statement as was the case with SU$C16. However, if you choose all the defaults, you must still have the // PARAM statement present in your control stream. SU$CSL checks the file control block to determine the type of disk pack being copied.

The format of SU$CSL is



Use the COPY parameter to indicate how many copies of your input disk pack are to be made. The value of *n* may be from 1 to 7. If you omit COPY, one copy is made.

Use the VEFY parameter to verify your new output. If you are making more than one copy, each copy is verified against the input. If you omit VEFY, no verification is performed.

Use the PRNT parameter to print any records found to be in error. If an error is detected, both the input and output record is listed with its corresponding output disk address (*ccchhrr*), in hexadecimal. You would normally choose PRNT=YES if you had chosen VEFY=YES. If you omit PRNT, only fatal errors are written on the system console.

Use the OVEF parameter to indicate that a verify-only operation is to be performed. No copy will be made. If you omit OVEF, one or more copies will be made, with or without verification, depending on other parameter specifications. OVEF may be used in conjunction with BGAD and EDAD.

Use the BGAD parameter to indicate the starting address of the first track to be copied, in hexadecimal. The address you specify must be the cylinder and head number (*ccchh*). If you omit BGAD, the beginning address is cylinder 000, head 00.

Use the EDAD parameter to indicate the ending address of the last track to be copied in hexadecimal. As explained in the BGAD parameter, the address you specify must be the cylinder and head number (*ccchh*). If you omit EDAD, the values will be defaulted for each device as shown in the format.

Since the programming logic of SU$CSL is similar to that of SU$C16, the input addresses are the same as your output addresses and any information contained in your specified output area is destroyed and the new information is written in that area.

Use the UNXF parameter to check for the file expiration date. If the file date has not expired, a message is displayed that asks you either to continue processing or to skip the file and continue to the next file (providing you are processing more than one file). Using the UNXF parameter eliminates the time and expense of keeping outdated files. It also eliminates the overlaying of files that should not be destroyed.

SU$CSL Interfacing with Job Control

The file name DISCIN must be specified on the // LFD job control statement for your input disk pack. The file name DISCOT must be specified on the // LFD job control statement for your output disk pack. If you are copying onto more than one disk pack, then DISCOTO1 through DISCOTO6 must be specified for each additional disk pack being copied.

Executing SU$CSL

In a multijob environment, the use of SU$CSL can cause space allocated to a file to be lost, files may be added to or portions deleted, or a disk pack may be incorrectly copied. These situations can occur when another job is updating the VTOC of a disk pack at the same time SU$CSL is copying that VTOC to another disk pack, or when SU$CSL is copying a file while another job is extending or scratching that file. Therefore, the job executing SU$CSL must be the only job running when the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonshareable by the operator. This ensures that an exact copy will be made by SU$CSL.

Programming Examples

The following control streams show some typical examples of how to use SU$CSL:

Example 1

```
1         10      16                                                    72
// JOB COPY8433
// DVC 50 // VOL DSP001 // LFD DISCIN
// DVC 51 // VOL DSP002 // LFD DISCOT
// EXEC SU$CSL
// PARAM
/&
// FIN
```

Here is a basic 8433 disk copy. There is no printing or verification. Your starting address is cylinder 000, head 00, while your ending address is cylinder 327, head 12.

Example 2

```
1         10      16                                                    72
// JOB COPY8433
// DVC 50 // VOL DSP001 // LFD DISCIN
// DVC 51 // VOL DSP002 // LFD DISCOT
// DVC 20 // LFD PRNTR
// EXEC SU$CSL
// PARAM PRNT=YES,VEFY=YES,EDAD=050
/&
// FIN
```

Here, you are only copying up to cylinder X'50', as indicated by EDAD=050. You are also using the verification and print options, making sure the information written is the same as that read.

Example 3

```
1         10      16                                                    72
// JOB COPY8430
// DVC 50 // VOL DSP001 // LFD DISCIN
// DVC 51 // VOL DSP002 // LFD DISCOT
// DVC 52 // VOL DSP003 // LFD DISCOT01
// DVC 20 // LFD PRNTR
// EXEC SU$CSL
// PARAM COPY=2,VEFY=YES,PRNT=YES,BGAD=010,EDAD=100,UNXF=YES
/&
// FIN
```

Here, you are using all the parameters. You are making two copies by using the 8430 disk packs. Your copy starts at cylinder X'10' and ends at cylinder X'100' with verification and printing of any errors being detected, as well as file expiration date checking.

Section 16
System Utility Symbiont

The system utility symbiont (SL$$SU) is a multipurpose utility that allows you to perform various functions using cards, tapes, disks, or diskettes. Table 16-1 breaks down the different functions to the media associated with them.

Table 16-1. SL$$SU Functions

Function Code	Function Performed
	Card Functions
CC	Reproduces cards punched in Hollerith code.
CCB	Reproduces cards punched in binary and Hollerith code.
CCS	Reproduces and resequencing source programs.
CT	Writes card to tape in unblocked format.
CTR	Writes card to tape in blocked format.
CP	Lists cards.
CH	Lists cards containing compressed mode.
JCP	Punches cards from the system console.
TT	Copys a tape to another tape.
TH	Prints a tape in character and hexadecimal format.
THR	Prints a tape in character, hexadecimal, deblocked format.
TP	Prints a tape containing only standard characters.
TPR	Prints a tape in character and deblocked format.
TRS	Locates a specific record on tape.
TC	Punches cards from tape.

continued

Table 16-1. SL$$SU Functions (cont.)

Function Code	Function Performed
	Card Functions (cont.)
INT	Preps a tape.
FSF	Forward spaces to a specific file.
BSF	Backward spaces to a specific file.
FSR	Forward spaces to a specific record.
BSR	Backward spaces to a specific record.
WTM	Writes tape marks.
REW	Rewinds a tape.
RUN	Rewinds a tape with interlock.
ERG	Erases a portion of a tape.
	Disk/Format Label Diskette Functions
AVX	Displays available disk extents on the console screen
DD	Prints a disk in unblocked format.
DDR	Prints a disk in reblocked format.
SVT	Prints a short-format VTOC file.
VTP	Prints the volume table of contents of a disk.
	Diskette Functions
DD	Prints a diskette in unblocked format.
VTP	Prints the data set labels of a diskette.

Section 17
Buffer Analysis Routine

The buffer analysis routine prints information about dynamic buffers, using the YDUMP file as input.

Format

```
RU BUFLIST,,V=vsn,BUF=(B)
                      (S)
                      (L)
                      (E)
                      (D)
```

where:

V=vsn
: Specifies the volume serial number of the resident device. The volume serial number of the YDUMP file is the default.

BUF=
: Specifies the type of buffer analysis report to be printed.

  BUF=B
  : Specifies the *brief* option, which prints totals only. This is the default.

  BUF=S
  : Specifies the *short* option, which prints information only for those lists that contain buffers.

  BUF=L
  : Specifies the *long* option, which prints information about every buffer on a list as well as the total bytes used by allocated and free buffers.

  BUF=E
  : Specifies the *extended* option, which is the same as the long (L) option, but also includes information about memory usage, starting with the first buffer area and scanning to the end of memory.

  BUF=D
  : Specifies the *debug* option, which is the same as the long (L) option, but also prints selected data bases in hexadecimal format. This option is useful for debugging the program.

Appendix A
Canned Job Control Streams

A.1. Referencing the Canned Job Control Streams

The following canned job control streams provide you with a more convenient method of performing some system utilities without the need of punching the parameters and job control statements normally required to run them. The utilities reside in the system load library file (YLOD), and their corresponding job control streams reside in the system job control stream library file (YJCS). The utilities are initiated from the system console by keying in their associated job control stream name.

Table A-1 shows the job names associated with the utilities, the functions performed, and the manuals where they can be found.

Table A-1. Canned Job Control Streams

Job Name	Function	Described in Document
CGV/CHGVSN	Changes a volume serial number on a previously prepped disk.	System Service Programs (SSP) Operating Guide (UP-8841)
DCOP	Copies SYSRES to another disk of the same type.	Installation Guide (UP-8839)
DRDP	Prints directory partition of a librarian disk file.	System Service Programs (SSP) Operating Guide (UP-8841)
DUMPLOG	Dumps job or console log records to disk.	Spooling and Job Accounting Concepts and Facilities (UP-8869)
DUMPLOGT	Dumps job or console log records to tape.	Spooling and Job Accounting Concepts and Facilities (UP-8869)
JBLOG	Produces a job accounting report with SYSLOG residing on disk.	Spooling and Job Accounting Concepts and Facilities (UP-8869)

continued

Table A-1. Canned Job Control Streams (cont.)

Job Name	Function	Described in Document
COPYREL	Copies SYSRES files to a new SYSRES volume.	Installation Guide (UP-8839)
IMPLDSKT	Creates an IMPL diskette.	Installation Guide (UP-8839)
JBLOGT	Produces a job accounting report with SYSLOG residing on tape.	Spooling and Job Accounting Concepts and Facilities (UP-8869)
LISTRES	Prints directory for SYSRES modules.	System Service Programs (SSP) Operating Guide (UP-8841)
MODLST	Lists the contents of the system libraries.	System Service Programs (SSP) Operating Guide (UP-8841)
SCLIST	Lists the shared code (YSCLOD) modules.	Supervisor Technical Overview (UP-8831)
SETREL	Preps and allocates RELEASE/SYSRES files.	Installation Guide (UP-8839)
SMCLIST	Lists software maintenance corrections.	System Service Programs (SSP) Operating Guide (UP-8841)
SYSDUMP	Prints a complete system dump from SYSRES or another system disk.	Dump Analysis User Guide/Programmer Reference (UP-8837)
SYSDUMPO	Prints a complete system dump or a portion of a system dump from SYSRES or another system disk.	Dump Analysis User Guide/Programmer Reference (UP-8837)

A.2. Copying Release or SYSRES Libraries (COPYREL and SETREL)

For convenience, we show the formats of the COPYREL and SETREL canned job control streams. You use SETREL first to prep your disk and assign the system files that will reside on the disk after you execute COPYREL. Not only are COPYREL and SETREL useful during your system installation, they can be used any time you need to copy all the RELEASE or SYSRES volume libraries to a backup disk of any type.

The format of COPYREL is

```
RVΔCOPYREL,,V=vsn,T=( 16 ),[,S=first-file][,L=last-file][,CAT=Y][,SEC=Y]
                    | 17 |
                    | 18 |
                    | 19 |
                    | 30 |
                    | 33 |
                    | 70 |
                    | 94 |
```

The format of SETREL is

```
RVΔSETREL,,V=vsn,T=( 16 ),P=prep-type[,CR=NO][,R=n]
                   | 17 |
                   | 18 |
                   | 19 |
                   | 30 |
                   | 33 |
                   | 70 |
                   | 94 |
```

For a complete description of COPYREL and SETREL, see the current version of the *Installation Guide* (UP-8839).

Appendix B
Code Set Components

B.1. Introduction

A code set defines the types of records needed to make up a particular type of module or a module group. It also defines the required sequence for these records. The recognized code sets are summarized below and are described in detail in B.2.

A. Grouped Code Sets

- 1 beginning-of-group demarcator, type A0
- 1 or more source, macro/jproc, object, or load module code sets
- 1 end-of-group demarcator, type A8
- 1 EOF code sentinel, type A1

B. Program Source and Macro/Jproc Source Module Code Sets

- 1 header, type A3 or A4
- 1 or more source items, type 24 or 25

C. Object Code Sets

- 1 header, type 80
- 1 or more linkage editor control statements, type 40 (optional)
- 1 or more CSECT, types 08, 09, 0A, 0B
- 1 or more ESD, types 04, 06, 07 (optional)
- 1 or more text, type 02
- 1 transfer, type 03
- 1 or more linkage editor control statements, type 40 (optional)
- 1 or more ISD records, type 0C

D. Load Code Sets

- 1 header, type 90 or B0 (root phase definition)
- 1 or more SENTRY, type C4 (optional)
- 1 or more sets of resource and SEXTERN records, type C8 and C6 (optional)
- 1 or more text, type 12 or 32
- 1 transfer, type 13
- 1 or more sets phase definition (type 90 or B0), text (type 12 or 32), and transfer (type 13) records, depending on the number of phases in the load module (optional)
- 1 or more ISD records, type 1C

B.2. Description

B.2.1. Grouped Code Sets

Modules in a file can be grouped by inserting beginning-of-group and end-of-group demarcator records within the file. A single group can contain modules of assorted types. The use of groups is optional and determined by the user.

The librarian is used to insert the beginning-of-group and end-of-group records in the file. Once the groups are created, the librarian can also be used to manipulate the module code sets on a group-by-group basis.

Groups may be nested to any number of levels. (Figure B-1 illustrates nested groups.) The formats for a beginning-of-group, end-of-group, and end-of-file sentinel records are given in Tables B-1 through B-3.

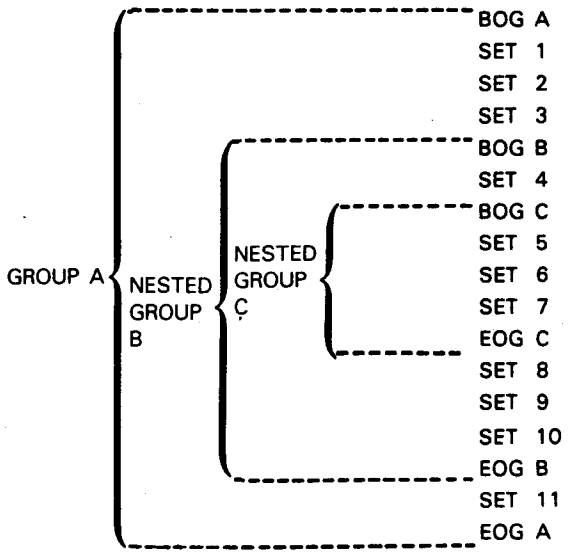


Figure B-1. Example of Nested Group Code Sets

Table B-1. Beginning-of-Group (BOG) Header Record Format

Byte Position	Field	Contents
0	Length prefix	38 (binary format)
1	Type prefix	$A0_{16}$
2-9	Group name	Symbolic name of logical group of code sets contained within the group starting with this record (left-justified and space-filled)
10-39	Comments	Up to 30 bytes of comments to identity the group

Table B-2. End-of-Group (EOG) Trailer Record Format

Byte Position	Field	Contents
0	Length prefix	8 (binary format)
1	Type prefix	$A8_{16}$
2-9	Group name	Symbolic name of logical group of code sets contained within the group starting with this record (left-justified and space-filled)

Table B-3. End-of-File (EOF) Sentinel Record Format

Byte Position	Field	Contents
0	Length prefix	20 (binary format)
1	Type prefix	$A0_{16}$
2-13	Unused	00_{16}
14-21	Name	ENDLIBΔΔ

B.2.2. Source Module Code Sets

A source module code set is composed of source module code statements. These may be macro definitions, own-code specifications, jprocs written in job control language, or source statements for a specific language processor. The formats for source module header and source statement records are given in Tables B-4 through B-6.

Table B-4. Source Module Code Header Record Format

Byte Position	Field	Contents
0	Length prefix	56 (binary format)
1	Type prefix	$A3_{16}$ or $A4_{16}$
2, 3	Unused	00_{16}
4	Flag	Bit 0 (80_{16}) Set to indicate module has been corrected. Bit 1 (40_{16}) Reserved Bit 2 (20_{16}) Set to indicate that module is deleted.
5-13	Unused	00_{16}
14-21	Module name	Symbolic name of the source code set started by record (left-justified and space-filled)
22-24	Date	In the form as it appears in the preamble
25-26	Time	Hour-minute (packed decimal less zone field)
27	Unused	00_{16}
28-57	Comments	Up to 30 bytes of comments to identify the source module

Table B-5. Source Module Code Statement Record Format

Byte Position	Field	Contents
0	Length prefix	Variable: 2 + length
1	Type prefix	24_{16}
2-81	Source record	Source statement

Table B-6. Compressed Source Module Code Statement Record Format

Byte Position	Field	Contents
0	Length prefix	Variable: 2 + compressed source length
1	Type prefix	25_{16}
2-81	Source record	Compressed source statement

B.2.3. Object Code Sets

An object module code set consists of text and relocation data output by a particular language processor. It can also contain control statement records used by the linkage editor to generate load code. Object module records are variable-length records and are packed as densely as possible within a given library block. The desired record sequence is:

- Object module header record
- Control statement records*
- All control section records (must precede associated text and entry ESDs)
- All ESD records (Names must be unique.)
- All ISD records**

* Control statement records are generated by certain language processors and may be used to designate control information necessary to a subsequent linkage editor run.

** ISD records are also generated by certain language processors and are used by JOBDUMP to produce a formatted dump if an abnormal termination occurs in your load module.

- All text/RLD records
- Object module transfer record
- Control statement records

These records are described in Tables B-7 through B-16.

Table B-7. Object Code Header Record Format

Byte Position	Field	Contents
0	Length prefix	55 (binary format)
1	Type prefix	80_{16}
2	ESID	00_{16}
3	Unused	Bit 0 (80_{16}) Set to indicate that the module has been patched. Bit 1 (40_{16}) Reserved Bit 2 (20_{16}) Set to indicate that module has been deleted. Bit 3-6 Not used Bit 7 (01_{16}) Set to indicate that the module is reentrant.
5-8	Address	Assembled or compiled origin of object module
9-12	Module length	Total number of bytes required for object module.
13-20	Module name	Symbolic name of object module started by this record (left-justified and space-filled).
21-23	Date	In the same form used in the preamble
24, 25	Time	Hour-minute (packed decimal less zone field)
26	Unused	00_{16}
27-56	Comments	Up to 30 bytes of comments to identify object module

Table B-8. Object Code Control Section Record Format

Byte Position	Field	Contents
0	Length prefix	19 (binary format)
1	Type prefix	08_{16}, 09_{16}, $0A_{16}$, or $0B_{16}$ (see Table B-10)
2	ESID	External symbol identification assigned to this control or common section.
3, 4	Flag bytes	8000_{16} indicates a deferred length specified in the transfer record of this object module; ignore bytes 9 through 12.
5-8	Section address	Compiled address of the start of this control or common section
9-12	Section size	Total length in bytes of this control or common section
13-20	Section name	Symbolic name of the control or common section (left-justified and space-filled)

Table B-9. Possible Control Section Record Types

Type of Control Section	Record Type	Record Length	Field Contents				
			2	3,4	5-8	9-12	13-20
Name control section	08	19	ESID	0000_{16} or 8000_{16}	Address	Length	Control section name
Unnamed control section	09		"	"	"	"	Blanks (40_{16})
Named common section	0A		"	"	"	"	Common section name
Unnamed common section	0B		"	"	"	"	Blanks (40_{16})

Table B-10. Object Code ESD Record Format

Byte Position	Field	Contents
0	Length prefix	15 (binary format)
1	Type prefix	04_{16}, 06_{16}, or 07_{16} (see Table B-12)
2	ESID	External symbol identification assigned to this ESD reference
3, 4	Unused	00_{16}
5-8	Relative Address	Processor-generated address or value assigned to this ESD reference
9-16	ESD name	Symbolic name of the ESD reference

Table B-11. Possible ESD Record Types

ESD Type	Record Type	Record Length	Field Contents			
			2	3, 4	5-8	9-16
ENTRY	04	15	ESID	0000_{16}	Assembled address	Symbol
EXTRN	06		"	"	" "	"
V-CON	07		"	"	" "	"

Table B-12. Object Code ISD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable
1	Type prefix	$0C_{16}$
2	ESID	External symbol identification of CSECT assigned to the ISD
3	Flag	Bits 0 and 1 unused Bit 2 set to indicate Type 3 ISD Bit 3 set to indicate Type 4 ISD (comment) Bit 4 through 7 unused
4	Flag	Unused
5-8	Compile origin	Processor-generated address assigned to this ISD
9-246	Attributes	Symbolic name and attributes of the ISD item

Table B-13. Object Code Text/RLD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	02_{16}
2	ESID	External symbol identification with which text data in this record is associated
3	Text length	Number of bytes less 1 of text data in this record
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5	Flag	01_{16} if patched text item
6-8	Relative address	Processor-assigned relative address of first byte of text data in this record

continued

Table B-13. Object Code Text/RLD Record Format (cont.)

Byte Position	Field	Contents
9 through 9 + text length	Text data	Instructions and/or data generated by a processor and relative to the ESID specified
9 + text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify the various preceding text data in this record (see Table B-14)

Table B-14. Relocation Mask Format

Byte Position	Field	Contents
0	ESID	External symbol identification of the external reference whose subsequent value will be used to modify the addressed field
1	Flag	Designator byte reflecting type, size, and position of the modification field (see Figure B-2)
2	Address	Relative record pointer indicating the most significant (leftmost) byte of text data at which the modification is to begin (first text byte, 0; 2nd byte, 1, etc.)

Notes:

1. Each RLD data field in a given text record is composed of 3 bytes of relocation information designating the field size, field position, and associated external index relevant to the modification of the addressed data bytes in this text record. The field may be positively or negatively relocated at link-edit time and can be modified by one or more relocation masks. The text and its associated relocation masks always must appear within the same logical record.

2. Load module relocation masks are identical, except that the ESID field represents the phase number assigned to the definition referenced by the address constant in the linked load module.

Figure B-2 is a sample relocation mask field.

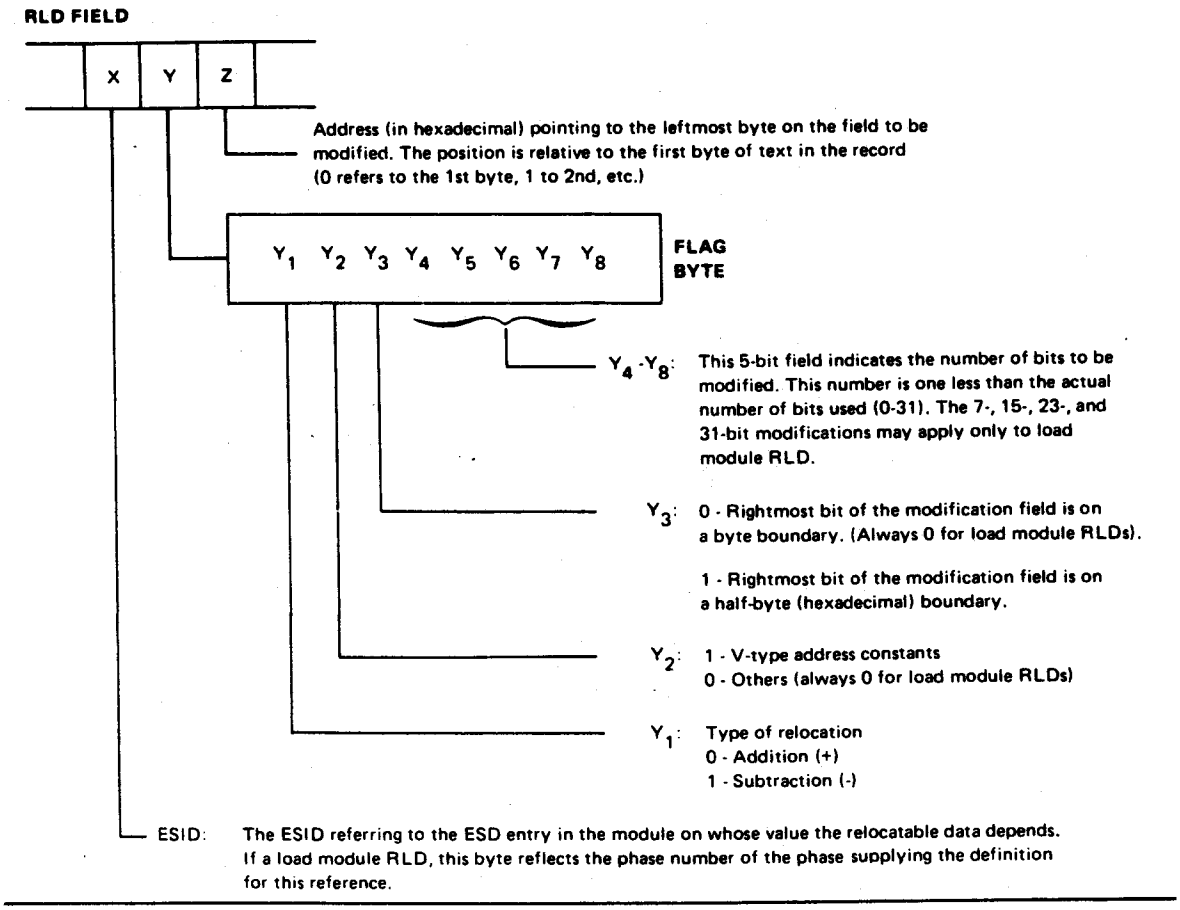


Figure B-2. Relocation Mask Field

Table B-15. Object Code Transfer Record Format

Byte Position	Field	Contents
0	Length prefix	11 + RLD (binary format)
1	Type prefix	03_{16}
2	ESID	External symbol identification assigned to the transfer reference
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5	Flag	80_{16} if deferred length is present in bytes 6-8. 40_{16} if transfer record does not terminate object module (1 or more control statements follow).
6-8	Deferred length	One CSECT or common section (named, unnamed, or blank) may have its respective record flagged to indicate that the object module transfer record specifies the actual length.
9-12	Transfer address	Processor-generated object module transfer address
13-13 + RLD length	RLD data	Relocation data to modify transfer address

Table B-16. Object Code Control Statement Record Format

Byte Position	Field	Contents
0	Length prefix	80 (binary format)
1	Type prefix	40_{16}
2-81	Control statement	Source control statement

Note: Any control statements appearing in an object module must directly follow a header record or a transfer record. The latter case is indicated by the appropriate setting of the flag byte in the transfer record.

B.2.4. Load Code Sets

A load module code set is produced by the linkage editor and is loaded into the system by the system load facility at the time of program execution. A load program may be composed of one or more phases, or program segments. The sequence of records in each phase of a load module is

- Phase definition record
- One or more SENTRY records (optional)
- One or more resource records (optional)
- One or more SEXTRN records (optional)
- One or more ISD records (optional)
- One or more text/RLD records
- Transfer record

Every load program contains an initial phase called the root phase. (Automatically included modules also become resident in the root phase.) Each phase segment contains its own transfer record, which identifies the end of the phase and possibly the starting address for program execution. The formats for load module code set records are given in Tables B-17 through B-21.

Table B-17. Load Code Phase Definition Record Format

Byte Position	Field	Contents
0	Length prefix	67 (binary format)
1	Type prefix	90_{16}
2	Phase number	Linkage-editor-assigned phase number of this phase
3, 4	Flag	Byte 3 Bit 0 (80_{16}) Set in root phase header to indicate clear module partition as defined in bytes 27 through 30. Bit 1 (40_{16}) Set to indicate that the load module calls reentrant code.

continued

Table B-17. Load Code Phase Definition Record Format (cont.)

Byte Position	Field	Contents
		Bit 2 (20_{16}) Set to identify the load module as reentrant. Bit 3 (10_{16}) Set to identify the load module as base 0 shared code. Bit 4 (08_{16}) Set to identify the load module as key 0 shared code. Bit 5-7 Reserved Byte 4 Bit 0 (80_{16}) Set to indicate that module has been patched. Bit 1 (40_{16}) Reserved Bit 2 (20_{16}) Set to indicate module has been deleted. Bit 3-7 Not used
5-8	Phase load address	Linkage-editor-assigned relative origin of this phase
9-12	Phase length	Total number of bytes required for this phase segment; value represents the highest zero relative address assigned to this phase.
13-20	Phase name	Symbolic name assigned to this loadable phase segment
21-23	Date	Month-day-year (packed decimal less zone field)
24, 25	Time	Hour-minute (packed decimal less zone field)
26	SENTRY count	Number of SENTRY records contained in the load module

continued

Table B-17. Load Code Phase Definition Record Format (cont.)

Byte Position	Field	Contents
27-30	Module length	Total number of bytes required for loading the module; value represents the highest zero relative address assigned to the load module.
31-38	Alias-phasename	Symbolic name assigned to this loadable phase segment by the linkage editor OVERLAY or REGION control statement that created the phase
39-68	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the load module segment

Table B-18. Load Module Shared Code Record Format

Byte Position	Field	Contents		
		Resource Records	SEXTRN Records	SENTRY Records
0	Length prefix	15 (binary format)	15 (binary format)	15 (binary format
1	Type prefix	$C8_{16}$	$C6_{16}$	$C4_{16}$
2	Number	Resource number	SINDEX number	SENTRY number
3, 4	Unused			
5-8	Length	Resource size	Byte 5 has resource number Bytes 6-8 unused	Link address
9-16	Name	Resource name (left-justified and zero-filled)	SEXTRN name (left-justified and blank-filled)	SENTRY name (left-justified and blank-filled)

Table B-19. Load Code ISD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable
1	Type prefix	1C
2	Phase number	Linkage-editor-assigned phase number of this phase
3	Flag	Bit 0 set to indicate Type 1 ISD (CSECT) Bit 1 set to indicate Type 2 ISD (comment) Bit 2 set to indicate Type 3 ISD Bit 3 set to indicate Type 4 ISD (comment) Bits 4-7 unused
4	Flag	Unused
5-8	Link origin	Linkage-editor-assigned relative origin for this ISD record
9-12	Compile origin	Language-processor-generated address to the ISD record
13-16	Size	Size of this ISD record
17-250	Attributes	Symbolic name and attributes of this ISD record

Table B-20. Load Code Text/RLD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	12_{16}
2	Phase number	Linkage-editor-assigned phase number of text data in this data
3	Text length	Number of bytes less 1 of text data in this record

continued

Table B-20. Load Code Text/RLD Record Format (cont.)

Byte Position	Field	Contents
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5	Flag	01_{16} if a patched text item
6-8	Load address	Linkage-editor-assigned phase segment load address assigned to the first byte of text data in this record
9 through 9 + text length	Text data	Instructions or data to be loaded relative to the load address
9 + text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify text in the record (see Table B-14)

Table B-21. Load Code Transfer Record Format

Byte Position	Field	Contents
0	Length prefix	11 + RLD data length (binary format)
1	Type prefix	13_{16}
2	Phase number	Linkage-editor-assigned phase number of this phase
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5-8	Unused	00_{16}
9-12	Transfer address	Linkage-editor-assigned phase segment transfer address
13 through 13 + RLD	RLD data	Relocation data used to modify the transfer address

B.2.5. Block Load Code Sets

Unlike standard load modules which use two file partitions, block load modules use three partitions. Partition 1 contains directory records just like those for standard load modules. Partition 3 contains the actual block module text records in sequential load order. These text records contain unstructured, contiguous text data, free of any control information. This data is binary zero-filled. The data records in partition 2 define the boundaries of each phase in partition 3.

Tables B-22 through B-27 describe the formats for the records within a lock load code set in sequential order as they occur in the file.

Table B-22. Partition 1 - Directory Entry

Byte Position	Field
0-7	Symbolic name
8	Type flag ($B0_{16}$)
9-11	Block relative pointer
12	Record relative pointer

Table B-23. Partition 2 - Block Load Module Header Record

Byte Position	Field	Contents
0	Length prefix	75 (binary format)
1	Type prefix	$B0_{16}$
2	Phase number	Linkage-editor-assigned phase number of this phase
3, 4	Flag	Byte 3 Bit 0 (80_{16}) Set in root phase header to indicate clear module partition as defined in bytes 27 through 30.

continued

Table B-23. Partition 2 - Block Load Module Header Record (cont.)

Byte Position	Field	Contents
		Bit 1 (40_{16}) Set to indicate that load module calls reentrant code.
		Bit 2 (20_{16}) Set to identify load module as reentrant.
		Bit 3 (10_{16}) Set to identify load module module as base 0 shared code.
		Bit 4 (08_{16}) Set to identify load module as key 0 shared code.
		Bit 5-7 Reserved
		Byte 4
		Bit 0 (80_{16}) Set to indicate that module has been patched.
		Bit 1 (40_{16}) Reserved
		Bit 2 (20_{16}) Set to indicate module has been deleted.
		Bit 3-7 Not used
5-8	Phase load address	Linkage-editor-assigned relative origin of this phase
9-12	Phase length	Total number of bytes required for this phase segment; value represents highest relative zero address assigned to this phase.
13-20	Phase name	Symbolic name assigned to this loadable phase segment
21-23	Date	In the form as it appears in the preamble
24, 25	Time	Hour-minute (packed decimal less zone field)
26	SENTRYs count	Number of SENTRYs recorded

continued

Table B-23. Partition 2 - Block Load Module Header Record (cont.)

Byte Position	Field	Contents
27-30	Module length	Total number of bytes required for loading the module; value represents highest relative zero address assigned to the load module
31-38	Alias-phasename	Symbolic name assigned to this loadable phase segment by the linkage editor OVERLAY or REGION control statement that created the phase
39-68	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the load module segment
69-71	Block number	Pointer to text block (beginning of this phase in partition 3)
72-74	Block number	Pointer to first text or transfer block of this phase in partition 2
75	Displacement	Pointer to first text or transfer record of this phase in partition 2
76	Checksum	XOR of first byte of each text block of partition 3

Table B-24. Partition 2 - Block Load Module RLD Record

Byte Position	Field	Contents
0	Length prefix	1 + number of RLD times 5 (binary format)
1	Type prefix	32_{16}
2	Length of RLDs	Number of RLD masks times 5
3(3 + n x 5-1)	RLD masks	Five-byte RLD masks (see Table B-25)

Table B-25. RLD Mask

Byte Position	Contents
0	Phase number (in load module RLD mask)
1	Bits (in load module RLD masks)
2-4	Load module relative address

Table B-26. Partition 2 - Block Load Module Nonphase Text/RLD Record

Byte Position	Field	Contents
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	12_{16}
2	Phase number	Linkage-editor-assigned phase number of text data in this record
3	Text length	Number of bytes less 1 of text data in this record
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5	Flag	01_{16} if a patched text item
6-8	Load address	Linkage-editor-assigned phase segment load address assigned to the first byte of text data in this record
9 through 9 + text length	Text data	Instructions and/or data to be loaded relative to the load address
9 + text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify text in the record (see Table B-14)

Note: Nonphase text records are present in block load modules when text/RLD items are detected that are not part of a given phase. Such text/RLD items outside the phase being loaded are to be loaded at the same time.

Table B-27. Partition 2 - Block Load Module Transfer Record

Byte Position	Field	Contents
0	Length prefix	11 + RLD data length (binary format)
1	Type prefix	13_{16}
2	Phase number	Linkage-editor-assigned phase number of text phase
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (multiple of 3 bytes)
5-8	Unused	00_{16}
9-12	Transfer address	Linkage-editor-assigned phase segment transfer address
13 through 13 + RLD length	RLD data	Relocation data used to modify the transfer address

Appendix C
SAT Library Structure

C.1. Library Blocks

SAT library space is allocated in fixed-length 256-byte blocks formatted as described in Table C-1.

Table C-1. Library Block Format

Byte Position	Field	Contents
0 - 2	Block number	Starting with 1 for the initial block, this is the logical block sequence number.
3	Block length	This is a binary value less than or equal to 251, indicating the number of bytes of relevant record data within the body of this block.
4	Unused	
5 - end of block	Variable records	Variable-length records comprising the body of data contained in this block (up to 251 bytes).

C.2. Library Records

Library records within blocks are variable-length records formatted as described in Table C-2.

Table C-2. Library Record Format

Byte Position	Field	Contents
0	Record length	Binary value (less than or equal to 249) indicating the length of the record data.
1	Type	Hexadecimal value indicating the record type (see Table C-3).
2 - end of record	Variable-length record data	Body of the particular record (up to 249 bytes of data)

Table C-3. Record-Type Byte Descriptions

Type Value (Hexadecimal)	Description
00	Nullified item records
02	TEXT/RLD records in object modules
03	Transfer records in object modules
04	Standard ENTRY records
06	Standard EXTRN records
07	V-CON records
08	Named CSECT records
09	Unnamed CSECT records
0A	Named common records
0B	Unnamed common records
0C	Object code ISD records

continued

Table C-3. Record-Type Byte Descriptions (cont.)

Type Value (Hexadecimal)	Description
12	TEXT/RLD records in load modules
13	Transfer records in load modules
16	Load code ISD records
1C	Load code ISD records
24	Program source or macro/jproc source module records
25	Compressed source code item
32	Blocked text or RLD records
40	Control statement records
80	Object module header records
90	Load module header/phase header records
A0	Beginning-of-group demarcator records
A1	End-of-file sentinel records
A2	Macro/jproc name header records (in directory only)
A3	Macro/jproc module header records
A4	Program source module header records
A8	End-of-group demarcator records
B0	Blocked load module header/phase header records
C4	Shared code ENTRY (SENTRY) records
C6	Shared code EXTRN (SEXTRN) records
C8	Resource records

C.3. Disk Directory

C.3.1. Directory Blocks

Table C-4 illustrates the space allocation for a directory block. Each disk file contains as many directory blocks as necessary to accommodate all the directory records needed to index the file.

Table C-4. Disk Directory Block Format

Byte Position	Field	Contents
0 - 2	Block number	This is the logical block sequence number.
3	Block length	This is a binary value equal to 251.
4	Checksum	This is a binary value reflecting an exclusive OR of the bytes in the block.
5 - 247	Directory records	These are 19 directory records, each 13 bytes long (see Table C-5 for format).
248 - 251	Unused	

C.3.2. Directory Records

Each directory record is 13 bytes long and is formatted as described in Table C-5.

Table C-5. Directory Record Format

Byte Position	Field	Contents
0-7	Symbolic name	This is an identifier for the module or demarcator in the data area which is indexed by this directory record.
8	Type	This corresponds to the record type of the demarcator record in the data area. (See Table C-6 for possible values.)
9-11	Block pointer	This is the logical block sequence number for the data block containing the demarcator record.
12	Record pointer	This is the number of bytes from the beginning of the data block to the beginning of the demarcator record indexed by this directory record.

Notes:

1. The block pointer and record pointer values are computed as the data partition is constructed.
2. The block pointer and record pointer values for a macro module header always identify the beginning of the proc module, regardless of where the name directive is contained within the body of the module.

Table C-6. Directory Record-Type Values

Type Value (Hexadecimal)	Description
00	Nullified item
04	ENTRY name (object module)*
08	CSECT name (object module)*
80	Object module header
90	Phase header (load module)
A0	Beginning-of-group demarcator
A1	End-of-file sentinal record
A2	Macro/jproc name*
A3	Macro/jproc module header
A4	Program source module header
A8	End-of-group demarcator
B0	Block module header record

* Multiple duplicate names can appear in library file directory.

Appendix D
File Transfer Utility (PCTRAN)

The file transfer utility (PCTRAN) permits transfer of data files, library modules, and source program files between disk or diskette on a Unisys personal computer (PC) and disk, diskette, or tape on an OS/3 system. Three user-friendly input screens are used to specify the desired file transfer activity, as well as the file names and media location used by both the host computer and the PC. PCTRAN can also be executed in batch mode using an MS-DOS® command (CMD) file.

D.1. Hardware Requirements

- PC/HT,™ PC/micro IT,™ or PC/IT™
- UNISCOPE® Communications Board (F4213-02), or Key Ring Adapter (F5091-00) and appropriate cable (F8487-00 or F8487-01), or USERNET™ UNISCOPE Protocol Communications Server (3147-xx)

D.2. Software Requirements

- MS-DOS Operating System, Release 2.1 or higher
- Synchronous Terminal Emulation Program (STEP) (F8721-00) with upgrade kit (F6166-00), or USERNET UNISCOPE Protocol Communications Server (F8743-xx)
- OS/3 Communications Software (ICAM)

MS-DOS is a registered trademark of Microsoft Corporation.

PC/HT, PC/micro IT, PC/IT, and USERNET are trademarks of Unisys Corporation.

UNISCOPE is a registered trademark of Unisys Corporation.

D.3. Setup Procedures

Load the Synchronous Terminal Emulation Program (STEP) into the PC and generate the proper UNISCOPE (U20) configuration for the PC. This procedure is described in the *Unisys Personal Computer Guide to Unisys Terminal Emulator User Guide* (UP-11886).

The OS/3 host system must have an ICAM generation with a line and U20 terminal defined with the same RID/SID as was specified in the PC configuraiton. The PC can be configured as a remote terminal emulating a workstation or as a remote workstation. See the *ICAM Programming Reference Manual* (UP-9749) for OS/3 ICAM generation procedures.

Some examples of OS/3 ICAM generation are:

CCA Specifications

```
REMO    LOCAP TYPE=(DMI),IAS=(YES,OFF),MODE=SYSTEM
```

Line and Terminal Specifications

Remote Terminal Emulating a Workstation

```
LN10    LINE DEVICE=(UNISCOPE),TYPE=(2400,SWCH,SYNC,UNAT),ID=10,    X
            CHAN=15,INPUT=(YES)
TM11    TERM FEATURES=(U20,1920),ADDR=(21,51),TCTUPD=YES,           X
            LOW=MAIN,MED=MAIN,HIGH=MAIN,AUX1=(COP,73)
TM12    TERM FEATURES=(U20,1920),ADDR=(21,52),TCTUPD=YES,           X
            LOW=MAIN,MED=MAIN,HIGH=MAIN,AUX1=(COP,73)
```

Remote Workstation

```
LN20    LINE DEVICE=(RWS),TYPE=(2400,SWCH,UNAT),ID=7,CHAN=15,       X
            INPUT=(YES)
        PGROUP PGID=23
TM21    TERM FEATURES=(U20,1920,,,PRIMARY),ADDR=(23,51),            X
            LOW=MAIN,MEDIUM=MAIN,HIGH=MAIN,AUX1=(COP,73)
TM22    TERM FEATURES=(U20,1920,,,SECONDARY),ADDR=(23,52),          X
            LOW=MAIN,MEDIUM=MAIN,HIGH=MAIN,AUX1=(COP,73)
```

Note: The AUX1 device is not necessary for this utility and is used only when you want to use the PC printer as an auxiliary printer on a workstation.

D.4. Operating Procedures

During the file transfer process (PCTRAN), you can press any function key to terminate the file transfer and initiate program restart. In the normal operating mode, the transfer terminates when the input end-of-file is detected on either the host file or the PC file. To use PCTRAN

1. Make sure that the OS/3 ICAM is ready, and that the communications line to be used is up and available.

2. Load the PC using the proper PC configuration generated in the setup procedure. For example, if the configuration is defined as RMT, the load entry is

```
STEP RMT
```

3. After the PC successfully connects, the word Poll flashes on the right side of line 25.

   Note: Sign on is not required if the PC is configured as a remote workstation and the line definition in the ICAM generation specifies DEVICE=(RWS); go to step 5.

4. Sign on to OS/3 using the $$SON command as follows:

```
$$SON TM11REMO
```

   where TM11 is the terminal-ID and REMO is the CCA LOCAP name defined in the ICAM generation.

5. The OS/3 logo is displayed. You can now log on in the normal manner. After you log on, perform step 5a or 5b to enter system mode to initialize PCTRAN.

   a. If the PC is operating as a remote terminal emulating a workstation, simultaneously press the ALT and 1 keys to put the PC in emulated system mode.

   b. If the PC is operating as a remote workstation, press the ALT and 4 keys simultaneously to put the PC in system mode.

6. Enter the following command to initialize PCTRAN:

```
PCTRAN  INDEX=1-8,CMD=drive:filename.ext
```

   where:

   INDEX
   Specifies the terminal index number that determines the logical terminal to be used during file transfers and batch mode command file processing. This value must be set to the same display number that is used to initiate PCTRAN.

CMD

Indicates the PC drive and DOS file name of the command file that contains PCTRAN input information for batch operation. See D.5 for PC CMD file usage.

The file transfer utility loads and the mode screen is displayed (see Figure D-1). Examples for each mode follow. (See the *General Editor (EDT) Operating Guide* (UP-9976) for additional information.)

```
* OS/3  PC  FILE  TRANSFER  UTILITY *

                Version 2R0

   1 : Receive Character Data File from Host
   2 : Transmit Character Data File to Host
   3 : Receive Hexify Data File from Host
   4 : Transmit Hexify Data File to Host
   5 : Terminate PCTRAN

   Select mode : (1)

NOTE:  Depress any function key to terminate transfer and restart.
```

Figure D-1. Mode Screen (Screen 1)

Example 1: Receive Character Data File from Host

1. Select mode (1) to Receive Character Data File from Host (see Figure D-1). Press the XMIT key to get the host screen (see Figure D-2).

```
*  O S / 3   P C   F I L E   T R A N S F E R   U T I L I T Y *

                RECEIVE DATA FROM HOST

Enter OS/3 file specifications in the following format:
          module-name,filename,vsn,SAT=YES
                        OR
          FILE=filename,VSN=vsn,RCFM=VAR
(file=test,vsn=res                                              )
(                                                               )
See general editor description of file specifications.
Sample Keywords: MODULE,FILE,VSN,TYPE,DEVICE,SAT

Note : The OS/3 file default parameter values are as follows:
     DEV=DISK            SAT=NO          RCFM=FIX
     EXTEND=YES          RCB=NO          RCSZ=256
     INC=1               INIT=NO         SIZE=2
```

Figure D-2. Host Screen (Screen 2)

2. Enter the appropriate host file name, volume serial number (VSN), and other parameter options to define the host file. Minimum requirements for a file definition are the file name and volume serial number. Figure D-2 shows that file name (test) and VSN (res) are specified.

3. After the host file has been opened, the PC screen (see Figure D-3) is displayed.

```
*  O S / 3  P C  F I L E  T R A N S F E R  U T I L I T Y *

                RECEIVE DATA FROM HOST

     Enter personal computer file specifications :

                PC drive:filename
                (A:testfile.dta )

          NOTE: File is in the form filename.extension.
```

Figure D-3. PC Screen (Screen 3)

4. Type in the appropriate PC specifications to define the PC file. Figure D-3 shows that drive (a) is specified to contain the PC file name (testfile) and the extension (dta). Press XMIT to transfer the host file (test) to the PC file (testfile.dta) located on drive (A).

5. When the file transfer is completed, the mode screen (see Figure D-4) is displayed indicating the total record count received from the host file. You can terminate PCTRAN or select another file transfer operation. Select mode 2, Transmit Character Data File to Host. Press XMIT to begin the next file transfer.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                    Version 2R0

    1 : Receive Character Data File from Host
    2 : Transmit Character Data File to Host
    3 : Receive Hexify Data File from Host
    4 : Transmit Hexify Data File to Host
    5 : Terminate PCTRAN
    Select mode : (2)

NOTE : Depress any function key to terminate transfer and restart.

RECEIVE RECORD COUNT = 000050
```

Figure D-4. Mode Screen Redisplayed at End of First Successful File Transfer

Example 2: Transmit Character Data File to Host

1. Figure D-5 shows the following parameter options selected for the host file:

   a. The record format (rcfm) is defined as variable to omit trailing blanks from the last block transferred.

   b. The record size (rcsz) is 168 characters.

   c. The record control byte (rcb=yes) precedes each record.

   d. Option (key1=1:7) specifies that characters 1 through 7 of the record are key fields.

   Note: Specify SAT=YES when you transfer a SAT module to a new host file; otherwise, the module is in a MIRAM file format (default).

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                TRANSMIT DATA TO HOST

 Enter OS/3 file specifications in the following format:
          module-name,filename,vsn,SAT=YES
                        OR
          FILE=filename,VSN=vsn,RCFM=VAR
(file=testout,vsn=res,rcfm=var,rcsz=168,rcb=yes,key1=1:7          )
(                                                                 )
See general editor description of file specifications.
Sample Keywords: MODULE,FILE,VSN,TYPE,DEVICE,SAT

Note : The OS/3 file default parameter values are as follows:
    DEV=DISK            SAT=NO        RCFM=FIX
    EXTENDED=YES        RCB=NO        RCSZ=256
    INC=1               INIT=NO       SIZE=2
```

Figure D-5. Host Screen Redisplayed

2. When the file is opened, the PC screen is displayed again (see Figure D-6). Select the PC drive that contains the file you want to transfer to the host; in this example, drive (A) and file name (testfile.dta). Press XMIT to execute the file transfer.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                TRANSMIT DATA TO HOST

 Enter personal computer file specifications :

          PC drive:filename
          (A:testfile.dta )

   NOTE: File is in the form filename.extension.
```

Figure D-6. PC Screen Redisplayed

3. When the second transfer is completed, the mode screen (see Figure D-7) is again displayed with a new total record count of the records transferred.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                    Version 2R0

    1 : Receive Character Data File from Host
    2 : Transmit Character Data File to Host
    3 : Receive Hexify Data File from Host
    4 : Transmit Hexify Data File to Host
    5 : Terminate PCTRAN

    Select mode : (3)

NOTE: Depress any function key to terminate transfer and restart.

RECEIVE RECORD COUNT = 000050
```

Figure D-7. Mode Screen Redisplayed at End of Second Successful File Transfer

Example 3: Receive Hexify Data File from Host

1. The host file may contain data such as field character control (FCC), null characters, etc., that need to be conserved and transferred back to the host intact. Select mode 3, Receive Hexify Data File from Host (see Figure D-7), to retain all control characters in the host file. Press XMIT to get the host screen.

2. In Figure D-8, the module (testtext) located in file (datafile) on the host screen, is specified for transfer to the PC. Press XMIT to open the files on the host.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                RECEIVE DATA FROM HOST

Enter OS/3 file specifications in the following format:
        module-name,filename,vsn,SAT=YES
                    OR
        FILE=filename,VSN=vsn,RCFM=VAR
(testtext,datafile,testvsn                                    )
(                                                             )
See general editor description of file specifications.
Sample Keywords: MODULE,FILE,VSN,TYPE,DEVICE,SAT

Note: The OS/3 file default parameter values are as follows:
    DEV=DISK          SAT=NO          RCFM=FIX
    EXTEND=YES        RCB=NO          RCSZ=256
    INC=1             INIT=NO         SIZE=2
```

Figure D-8. Host Screen Redisplayed

3. When the host file is opened, the PC screen (See Figure D-9) is displayed.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                RECEIVE DATA FROM HOST

Enter personal computer file specifications :

          PC drive:filename
          (B:hexdata      )

 NOTE: File is in the form filename.extension.
```

Figure D-9. PC Screen Redisplayed

4. Enter the PC drive to receive the host file (B) and the file name (hexdata). Press XMIT to execute the transfer of the host hexify file to the PC.

5. The mode screen (see Figure D-10) is displayed at the end of the transfer and indicates that 75 records were received by the PC.

```
*  OS/3  PC  FILE TRANSFER UTILITY*

                   Version 2R0

   1 : Receive Character Data File from Host
   2 : Transmit Character Data File to Host
   3 : Receive Hexify Data File from Host
   4 : Transmit Hexify Data File to Host
   5 : Terminate PCTRAN

   Select mode : (4)

NOTE:  Depress any function key to terminate transfer and restart.

RECEIVE RECORD COUNT = 000075
```

Figure D-10. Mode Screen Redisplayed at End of Third Successful File Transfer

6. Select mode 4, Transmit Hexify Data File to Host, and press XMIT to begin the next transfer operation.

Example 4: Transmit Hexify Data File to Host

1. The host screen (See Figure D-11) shows that the host module (stepmod) is stored in the SAT file (srcfil) located on the RES OS/3 disk pack. If the module name and SAT parameter are not included in the host file specification, a MIRAM file (srcfil) is stored on the RES OS/3 disk pack.

```
* OS/3  PC  FILE TRANSFER UTILITY *

                TRANSMIT DATA TO HOST

Enter OS/3 file specifications in the following format:
         module-name,filename,vsn,SAT=YES
                        OR
        FILE=filename,VSN=vsn,RCFM=VAR
(stepmod,srcfil,res,sat=yes                                    )
(                                                              )
See general editor description of file specifications.
Sample Keywords: MODULE,FILE,VSN,TYPE,DEVICE,SAT

Note: The OS/3 file default parameter values are as follows:
    DEV=DISK            SAT=NO          RCFM=FIX
    EXTEND=YES          RCB=NO          RCSZ=256
    INC=1               INIT=NO         SIZE=2
```

Figure D-11. Host Screen Redisplayed

2. Press XMIT to open the host file and get the PC screen (see Figure D-12). The PC loadable file (step.com) is selected for transfer to the host module (stepmod) from PC drive (A). Press XMIT to begin the transfer of data to the host.

   Note: You can receive the host module (stepmod) by another PC at a later time by selecting mode 3, Receive Hexify Data File from Host (see Example 3).

```
* OS/3  PC  FILE TRANSFER UTILITY *

                TRANSMIT DATA TO HOST

Enter personal computer file specifications :

          PC drive:filename
          (A:step.com     )

  NOTE: File is in the form filename.extension.
```

Figure D-12. PC Screen Redisplayed

3. When the transfer is completed, the mode screen (see Figure D-13) is displayed with a now total record count of the records transferred.

```
*  O S / 3   P C   F I L E   T R A N S F E R   U T I L I T Y *

                        Version 2R0

     1 : Receive Character Data File from Host
     2 : Transmit Character Data File to Host
     3 : Receive Hexify Data File from Host
     4 : Transmit Hexify Data File to Host
     5 : Terminate PCTRAN

     Select mode : (1)

  NOTE : Depress any function key to terminate transfer and restart.

  RECEIVE RECORD COUNT = 000223
```

Figure D-13. Mode Screen Redisplayed at End of Fourth Successful File Transfer

Example 5: Receive Character Data File from Host

1. This last example returns to the selection of mode 1 in Figure D-13 to show how PCTRAN transfers an OS/3 load module from the host to the PC. Press XMIT to get the next screen.

2. Figure D-14 shows the parameter (pctran00) specified as a load module and module type (l) in the host file (YLOD). Press XMIT to open the host file.

Note: *When a module is received from the host with a module type specified, you must use the same module type (l or o) when you transmit the module back to the host.*

```
*  O S / 3   P C   F I L E   T R A N S F E R   U T I L I T Y *

                    RECEIVE DATA FROM HOST

Enter OS/3 file specifications in the following format:
          module-name,filename,vsn,SAT=YES
                         OR
          FILE=filename,VSN=vsn,RCFM=VAR
(pctran00,$y$lod,res,l                                          )
(                                                               )
See general editor description of file specifications.
Sample Keywords: MODULE,FILE,VSN,TYPE,DEVICE,SAT

Note: The OS/3 file default parameter values are as follows:
    DEV=DISK            SAT=NO          RCFM=FIX
    EXTEND=YES          RCB=NO          RCSZ=256
    INC=1               INIT=NO         SIZE=2
```

Figure D-14. Host Screen Redisplayed

3. When the file has been opened, the PC screen (see Figure D-15) is displayed. Drive (B) and file name (pctran.lod) are selected on the PC to receive the host data. Press XMIT to begin the transfer.

Note: During the transfer, the OS/3 module (pctran00) is translated from object code in the host file to a PC ASCII data file in (pctran.lod). When the module is transferred back to the host, it is once again translated back to object code format with (l) specified as the module type.

```
*  O S / 3   P C   F I L E   T R A N S F E R   U T I L I T Y *

                    RECEIVE DATA FROM HOST

Enter personal computer file specifications :

          PC drive;filename
          (B:pctran-lod   )

  NOTE: File is in the form filename.extension.
```

Figure D-15. PC Screen Redisplayed

4. When this transfer is successfully completed, the mode screen (See Figure D-16) reappears showing the total record count received at the host computer.

```
* OS/3 PC FILE TRANSFER UTILITY*

                Version 2R0

   1 : Receive Character Data File from Host
   2 : Transmit Character Data File to Host
   3 : Receive Hexify Data File from Host
   4 : Transmit Hexify Data File to Host
   5 : Terminate PCTRAN

   Select mode : (5)

NOTE : Depress any function key to terminate transfer and restart.

RECEIVE RECORD COUNT = 000223
```

Figure D-16. Mode Screen Redisplayed at End of Fifth Successful File Transfer

5. After completing the several file transfers described in the preceding examples, select option 5 to terminate PCTRAN. A final screen (see Figure D-17) is displayed when PCTRAN is terminated normally.

Notes:

1. *If PCTRAN terminates abnormally, the PC termination screen shown in Figure D-17 is not displayed. Interactive services displays an error.*

2. *When PCTRAN is initiated from a menu, the PC termination screen is not displayed before at termination before returning to menu services.*

```
* OS/3 PC FILE TRANSFER UTILITY*

      HAS TERMINATED NORMALLY
```

Figure D-17. PCTRAN Termination Screen

D.5. Batch Mode

STEP provides the ability to retrieve commands from an MS-DOS file and transmit them to the host computer. The commands are sent in a sequential manner in response to information from the host system.

The PCTRAN user can specify an MS-DOS drive and command file name using the CMD parameter. The default file name is the CMDFILE contained on the current MS-DOS drive and directory.

D.5.1. Using STEP CMD File for PCTRAN Batch Mode

Using EDLIN or an MS-DOS editor, construct a PCTRAN command sequence similar to the following example. In this example, the STEP default CMD file name CMDFILE is used. Refer to the *Terminal Emulator User Guide* (UP-11886) for information on command file processing.

Statement	Function
1. `X,M,,28`	Wait for "(" character in PCTRAN mode screen. Set PC menu mode.
2. `>^K2^K>`	Send 1 to host and transmit.
3. `>^KFILE=xxxx,VSN=yyyy^K^K>`	Send OS/3 file parameters (*xxxx, yyyy*) and transmit.
4. `>^Ka^zzzz^K>`	Send PC drive A the PC file name (*zzzz*)
5. `>^K5^K>` or `>^K2^K>`	Terminate (5) or next file (2), and transmit
6. `X,C,,00`	Accept any data string for next batch file execution, and clear PC menu mode.

Execution for this example is as follows: press the Ctrl and S keys simultaneously to start the CMDFILE. Then key in RV PCTRAN in system mode and transmit.

Note: *Enter ^K by pressing the Ctrl and K keys simultaneously.*

D.5.2. Using STEP CMD File for Automatic Terminal Logon

Execution of the log on command file can be initiated immediately upon loading STEP. Select Y for the "Enable file transfer" and "Start CMDFILE file" options on the file transfer options screen of the STEP configuration.

Note: *Enter ^K by pressing the Ctrl and K keys simultaneously.*

Remote Terminal (UNISCOPE)

Sign on and log on to an OS/3 remote terminal can be accomplished by using the following sample command file:

Statement	Function
1. `X,C,,1e`	Wait for SOE from host
2. `>$$SON user-id` *or* `>$$open user-id`	OS/3 $$SON user-id Telcon $$OPEN user-id
3. `>  OS/3 LOGON IN PROGRESS  >`	Wait for logon screen and transmit
4. `X,M,,3E`	Wait for > in OS/3 logon screen, set PC menu mode
5. `>^KUSERID^K^K^K^KNO^KNO^K^K`	Send logon screen information
6. `X,C,,00`	Accept any data string for next batch execution, clear PC menu mode

Note: *There are no imbedded spaces in line 5.*

Remote Workstation

The following sample command file can be used to log on to an OS/3 remote workstation:

Statement	Function
1. `X,C,,00`	Accept any data string
2. `R DUMMY,,,"LOGON"`	Wait for DEPRESS TRANSMIT FOR LOGON message
3. `>  OS/3 LOGON IN PROGRESS  >`	Wait for logon screen and transmit
4. `X,M,,3E`	Wait for > in OS/3 logon screen, set PC menu mode
5. `>^KUSERID^K^K^K^KNO^KNO^K^K`	Send logon screen information
6. `x,c,,00`	Accept any data string for next batch file execution, clear PC menu mode

Notes:

1. *The word LOGON in line 2 must be in uppercase letters and must be enclosed in double quotation marks as shown.*

2. *If the CMDFILE is not executed immediately upon loading STEP, line 2 must not be used.*

D.6. Error Messages

If OS/3 or the PCTRAN program detects an error condition, an error message is displayed. Error messages are listed below.

Message Number **Meaning/Action**

FT000 INVALID SPECIFICATION FOR FILE

A syntax error has been detected in the host or PC specification. Correct the error and reenter specification.

FT001 PC DISK FULL, TRANSFER NOT COMPLETED

Another diskette/disk should be used or unnecessary files should be erased from the diskette/disk to provide additional file space.

FT002 PC DISK NOT READY

Put diskette/disk in ready state.

FT003 PC DISK IS WRITE-PROTECTED

Make sure the correct diskette/disk has been specified. If so, write enable the diskette/disk.

FT004 PC I/O ERROR

A diskette/disk I/O error has occurred. Correct the PC hardware problem.

FT009 PC ERROR-ACK NOT TRANSMITTED TO HOST

The PC has not transmitted an ACK (acknowledge) status to the host while receiving data. The data transfer is questionable.

FT010 PC BLOCK SIZE EXCEEDS MAXIMUM

The PC has transmitted a block of data that exceeds the PC control page block size. The data transfer is questionable.

Message Number	Meaning/Action
FT011	RECORD SIZE EXCEEDS MAXIMUM
	The record size specified by the host RCSZ parameter is larger than 7680 characters. Reenter the host specification with valid record size.
FT012	RECORD SIZE EXCEEDS RCSZ, RECORD SPANNED
	A data record larger than RCSZ has been transmitted to the host. The record has been spanned over multiple records on the host file at a record size equal to RCSZ.
FT013	PC HEXIFY DATA, MODULE TYPE INVALID
	Operator has specified a module type of O or L in the host file specification, but has selected hexify data mode transfer. OS/3 object O or load L modules must be transferred using character data mode.
FT014	SYSTEM ERROR MNN
	An OS/3 system error has occurred. Refer to the *System Messages Reference Manual* (UP-8076) for an explanation of this error.
FT015	INVALID TERMINAL INDEX VALUE
	An incorrect value has been specified for the IND=*n* parameter. Manually set the terminal index field of the control page to the correct screen number , or terminate PCTRAN and reenter the parameter with the correct screen number.
FT016	INVALID MODIFIER
	An incorrect value has been specified for the CMD parameter. Manually set the CMDFILE field of the control page, or terminate PCTRAN and reenter the parameter.

D.7. User Guidelines

The following guidelines apply to PCTRAN:

- The file transfer utility (PCTRAN) supports character and PC hexify mode data transmission; these modes are selected by the user. When using character mode, packed decimal fields and binary control characters within the data cannot be transferred and are converted to spaces. The EBCDIC characters that represent these control characters are

  00 through 07
  09 through 1A
  1C through 22
  24 through 27
  2A
  2D through 2F
  32
  34 through 37
  3C through 3D
  3F

- When using hexify mode, the data is translated by the PC terminal emulation software and by PCTRAN during the file transfer. This option allows you to transfer PC object code and binary data to and from the OS/3 host system.

- If transferring OS/3 object, load, or screen format modules to the PC, the data is translated by PCTRAN and is maintained on the PC as character data. When the module is transmitted back to the host system, it is translated to OS/3 object code format. This facility is activated whenever the host type parameter specifies O, L, F, FC, or Menu.

- Data base files can require formatting to sequential format prior to using PCTRAN.

- The maximum record size that can be transferred is 8192 characters.

- If you initiate a PCTRAN data transfer and the transfer is not completed (the mode screen is not redisplayed with a record count), do the following:

  1. Retrieve the STEP control page (press and hold the ALT key and then press the 3 key) to see if an error message is displayed.

  2. Press any function key to display the mode screen menu.

  3. Take the necessary corrective action (see the list of error messages).

  4. Reinitiate the data transfer.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

_____

(Document Title)

_____ _____ _____

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

Comments:

From:

_____

(Name of User)

_____

(Business Address)

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

CUT

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

Unisys Corporation
E/MSG Product Information Development
PO Box 500 — E5-114
Blue Bell, PA 19422-9990

FOLD

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

_____

(Document Title)

_____ _____ _____

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

Comments:

From:

_____

(Name of User)

_____

(Business Address)

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

CUT

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

Unisys Corporation
E/MSG Product Information Development
PO Box 500 — E5-114
Blue Bell, PA 19422-9990

FOLD