

NOS
USAGE

PROPRIETARY NOTICE

The ideas and designs set forth in this document are the property of Control Data Corporation and are not to be disseminated, distributed, or otherwise conveyed to third persons without the express written permission of Control Data Corporation.

| REVISION RECORD | |
|-----------------------------|------------------------------|
| REVISION | DESCRIPTION |
| A (12-20-85) | Manual Release. |
| B (08-04-87) | Manual updated to NOS 2.5.1. |
| C (12-10-87) | Manual updated to NOS 2.5.2. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| Publication No. FH3025-1 | |

NOS USAGE.

or use Comment Sheet in the back of this manual.

©COPYRIGHT CONTROL DATA CORPORATION
All Rights Reserved

NOS USAGE

GENERAL COURSE DESCRIPTION

Course Title: Network Operating System (NOS) Usage

Course Number: FH3025

Course Length: 5 days

Description:

This course introduces the student to the Network Operating System (NOS). After completion of this course, the student will be able to use permanent and local files, create and execute jobs using the NOS Job Control Language, create and execute procedure files, create user libraries and program libraries (PL), and be able to describe the basic functions of the loader. The student will also know where to obtain further information on all of the topics covered in the class.

Prerequisites: NOS CYBER 170 Introduction or equivalent

Student Materials:

- NOS Usage Workshop Student Handout
- NOS Version 2 Reference Set, Volumes 2-3
- Update Reference Manual
- CYBER Loader User Guide

NOS USAGE COURSE OUTLINE

- I. NOS Introduction and Overview**
 - A. CYBER Components**
 - B. Software/Hardware Interface**
 - C. Control Point Concept**
 - D. Job Flow**
- II. Permanent File Concepts and Commands**
 - A. Indirect Access Permanent Files**
 - B. Direct Access Permanent Files**
 - C. Permanent File Commands**
- III. FSE (Full Screen Editor)**
- IV. Local File Commands**
 - A. Status Files**
 - B. Returning Files**
 - C. File Positioning**
 - D. File Copying**
 - E. File Content Reports**
- V. Job Control Language**
- VI. Procedure Files**
 - A. Passive Procedures**
 - B. User Prologue Procedures (UPROC)**
 - C. Menu-driven Procedures**
 - D. Interactive Procedures**
 - E. procedure File Execution Control Commands**
 - F. Job Flow Control Language (CCL) Directives**

NOS USAGE COURSE OUTLINE (CONT)

VII. Library Generation and Maintenance

- A. Local Library File Generation**
- B. Local Library File Maintenance**
- C. Declaring Global Libraries**
- D. Statusing and Inspecting Library Files**

VIII. Source Program Library Maintenance

- A. Creating Source Program Library Files**
- B. Accessing Program Library Files**
- C. Modifying Program Libraries**
- D. Regenerating Source Records**

IX. Magnetic Tapes

- A. Creating Label/Unlabeled tapes**
- B. Requesting tapes by Volume Serial Number**
- C. Accessing Multiple Allocatable Resources**

X. Loader

- A. Load Maps**
- B. Relocable Load Sequences**
- C. Absolute Binary Files**
- D. Loader Commands**

NOS USAGE COURSE CHART

| HOUR | DAY 1 | DAY 2 | DAY 3 | DAY 4 | DAY 5 |
|------|-------------------------------|----------------------------|------------------------------------|---------------------------------------|-------------------------------------|
| 1 | INTRODUCTION | LOCAL FILE COMMANDS | JOB FLOW CONTROL COMMANDS | MENU LAB | UPDATE |
| | NOS OVERVIEW | | | INTERACTIVE PROCEDURES | UPDATE LAB |
| 2 | PERMANENT FILE CONCEPTS | | | | LOCAL FILES LAB |
| 3 | | | | | |
| 4 | LUNCH | LUNCH | LUNCH | LUNCH | LUNCH |
| 5 | PERMANENT FILE LAB | JOB CONTROL COMMANDS | PASSIVE PROCEDURES LAB | INTERACTIVE PROCEDURES LAB | LOADER |
| 6 | FSE | | MENU-DRIVEN PROCEDURES | LOCAL/GLOBAL LIBRARY GENERATION | TAPE OR LOADER LAB (OPTIONAL) |
| 7 | FSE LAB | JOB CONTROL LAB | HELP LAB | LIBRARY LAB | REVIEW |

LESSON ONE

INTRODUCTION TO THE

NETWORK OPERATING SYSTEM

LESSON ONE

INTRODUCTION TO THE NETWORK OPERATING SYSTEM

Lesson Preview:

This lesson is intended to provide an overview of the Network Operating System (NOS) and the software/hardware interface.

Objectives:

After completing this lesson, the student will be able to:

- Name the basic components of a Control Data CYBER series computer.
- Have a basic understanding of the software/hardware interface.
- Log in to NOS.
- Describe the five stages of job flow through the system.

Projects:

- Exercise 1.1 (Study Questions)
- Exercise 1.2 (Definitions Review)
- Exercise 1.3 (Login to NOS)

References:

- Network Operating System Version 2 Reference Manual, Volume 2

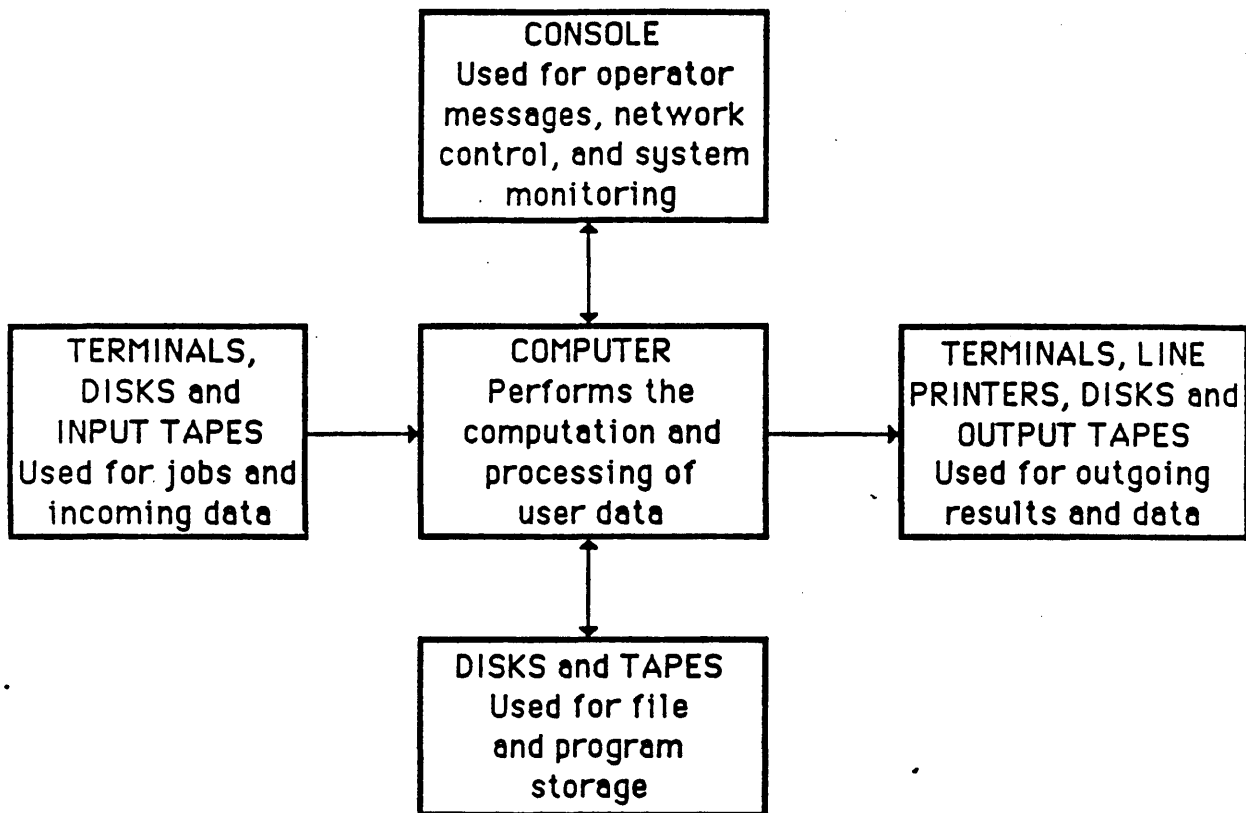
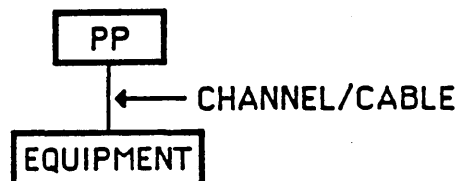


Figure 1-1. Overview of a Computer System

SYSTEM HARDWARE DEFINITIONS

| | |
|-------------------------------|--|
| CONSOLE | Displays operator messages, accepts operator commands, and controls network operations |
| CENTRAL PROCESSING UNIT (CPU) | Does arithmetic calculations and has access to central memory and extended memory. User programs execute in the CPU. |
| CENTRAL MEMORY (CM) | Used by user programs to contain executable code and transfer instructions to and from the CPU. Each CM word can contain up to four instructions or 60 bits of data. |
| EXTENDED MEMORY | Provides additional storage, fast data transfer rate, and is an option to central memory for use in data storage. |
| Input/Output Unit (IOU) | Contains multiple independent, generalized I/O computers called Peripheral Processors (PP). |
| Peripheral Processor (PP) | Can access channels, extended memory, and can read and write central memory and it's own memory. |
| CHANNELS | Provides path for PPs to move data to and from peripheral equipment. |



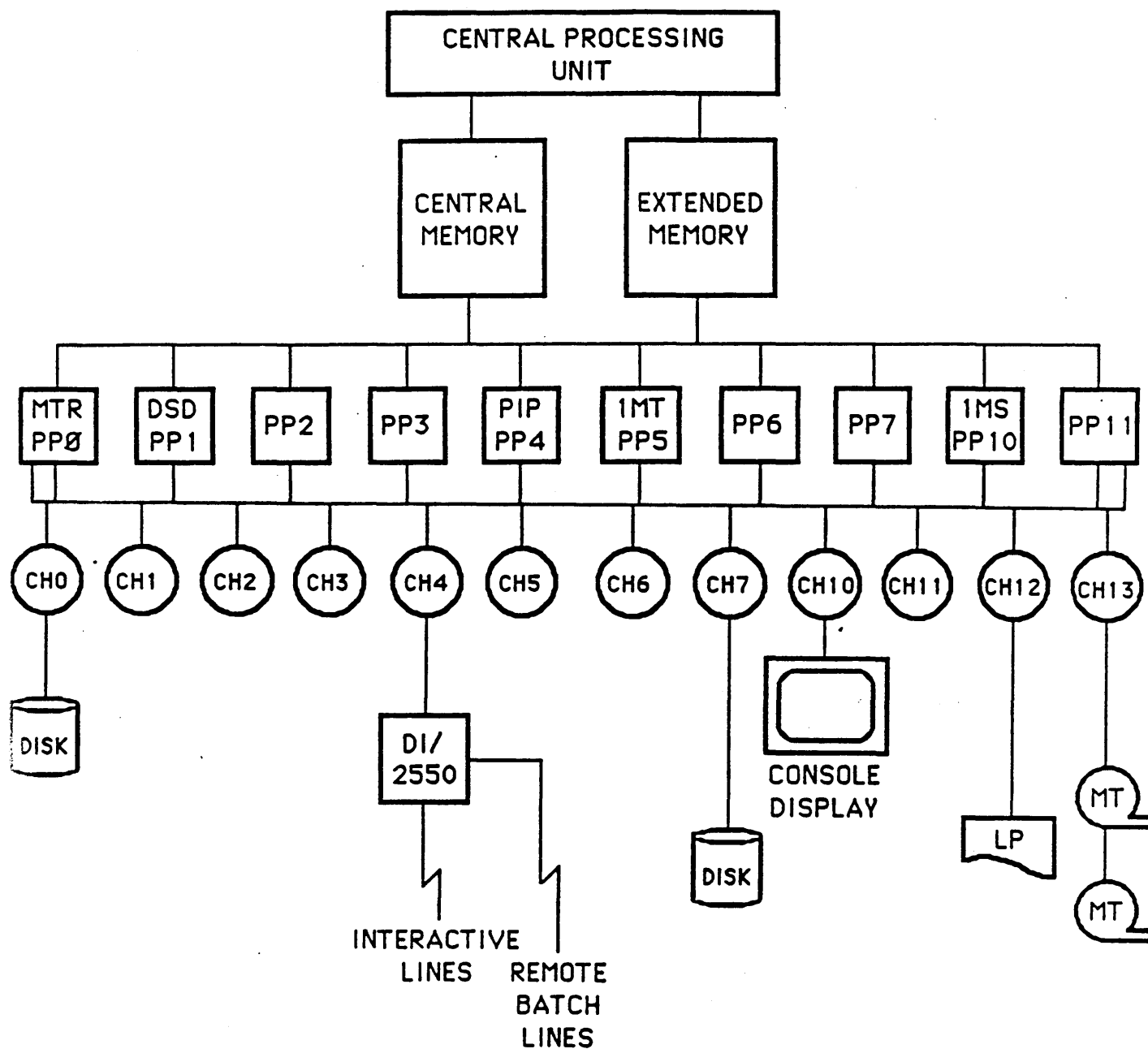


Figure 1-2. Example Hardware Configuration

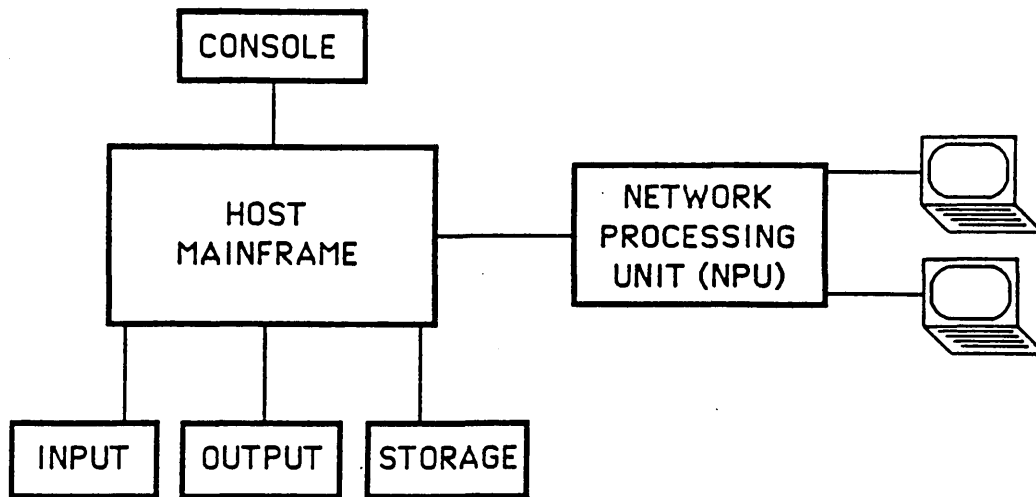


Figure 1-3. Example Single Host Arrangement

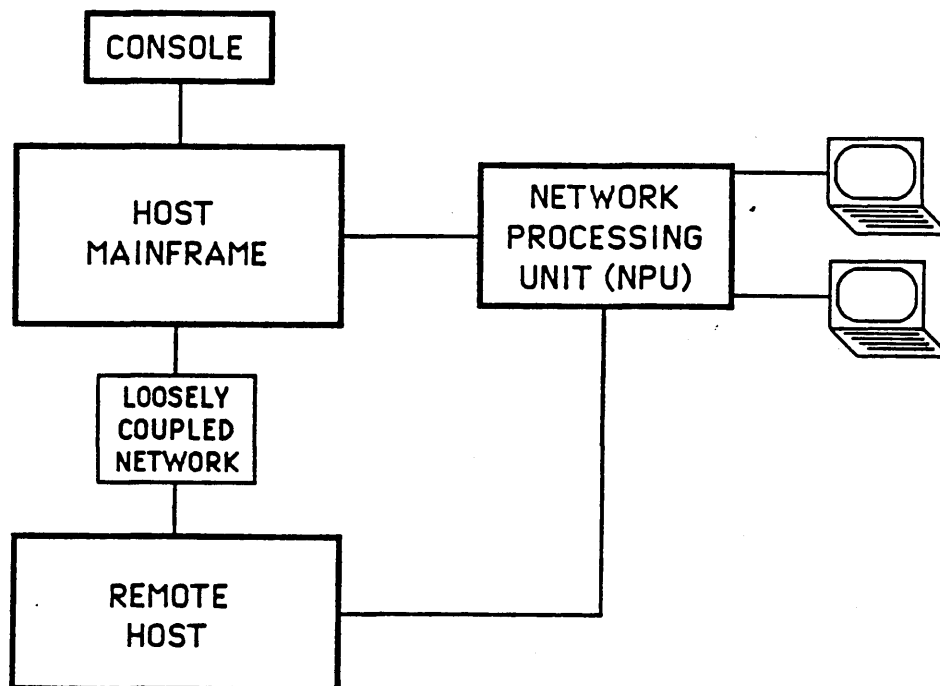


Figure 1-4. Example Multi-host Arrangement

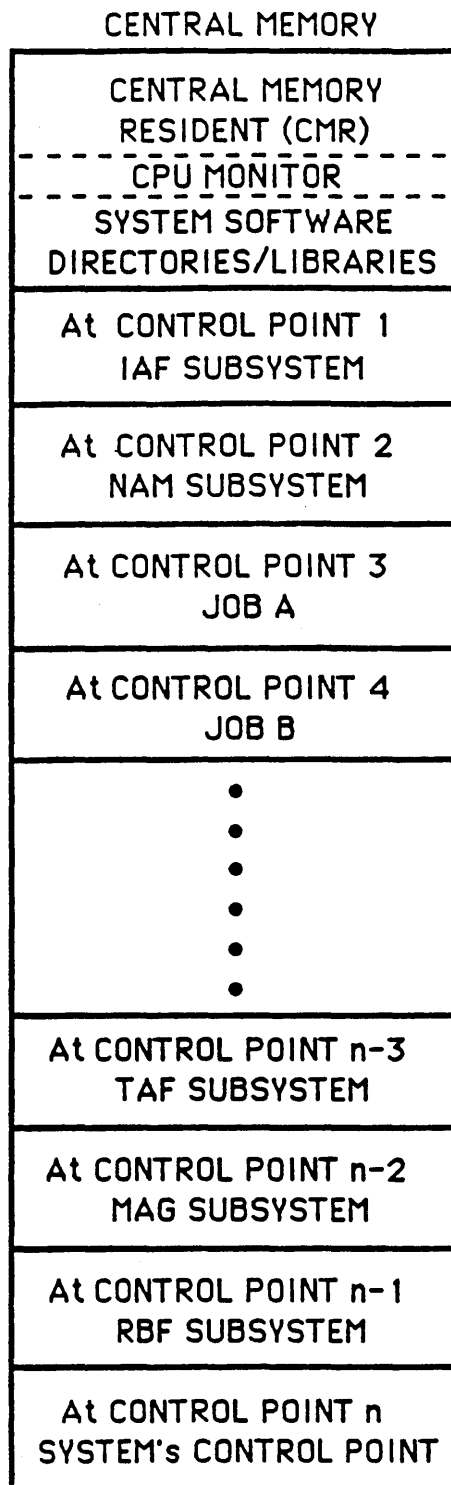


Figure 1-5. Example Memory Allocation

SOFTWARE DEFINITIONS

| | |
|-------------------------------|--|
| Central Memory Resident (CMR) | CMR resides in the low end of central memory permanently. The CPU monitor resides here, contains system tables, values, exchange packages, pointers, libraries, and performs executive and controlling functions during execution of user's job. |
| System Tables | An area in the beginning of Central Memory Resident (CMR) which keeps track of various operating system activities like Channel and equipment assignments, job control information, user exchange package and control point information and PP status. |
| Exchange Package | Defines CPU register values for a CPU program. Kept in CMR when job is not using the CPU. |
| Control Point | A job in memory is said to be at a control point. Some control points are reserved for system jobs. |
| Negative Field Length | Additional information about a user's job (e.g., local file names). |
| Reference Address (RA) | Address of first word of a users' area in central memory (CM). CPU uses this to keep track of where the user's job is stored. |
| Field Length (FL) | A contiguous amount of central memory occupied by an executing job at a control point. Starts at RA and ends at the last word occupied by the job. CPU hardware prevents users from reading outside of Field Length. |
| Job Communication Area | Contains information about user's CPU programs, including any outstanding system request, program parameters and hardware error exit information. |
| CPU Programs | Some reside in central memory (CM) at all times (CPU monitor). Some reside on disk and are loaded in CM and executed in CPU when needed. |
| Subsystems | Optional parts of the system. |

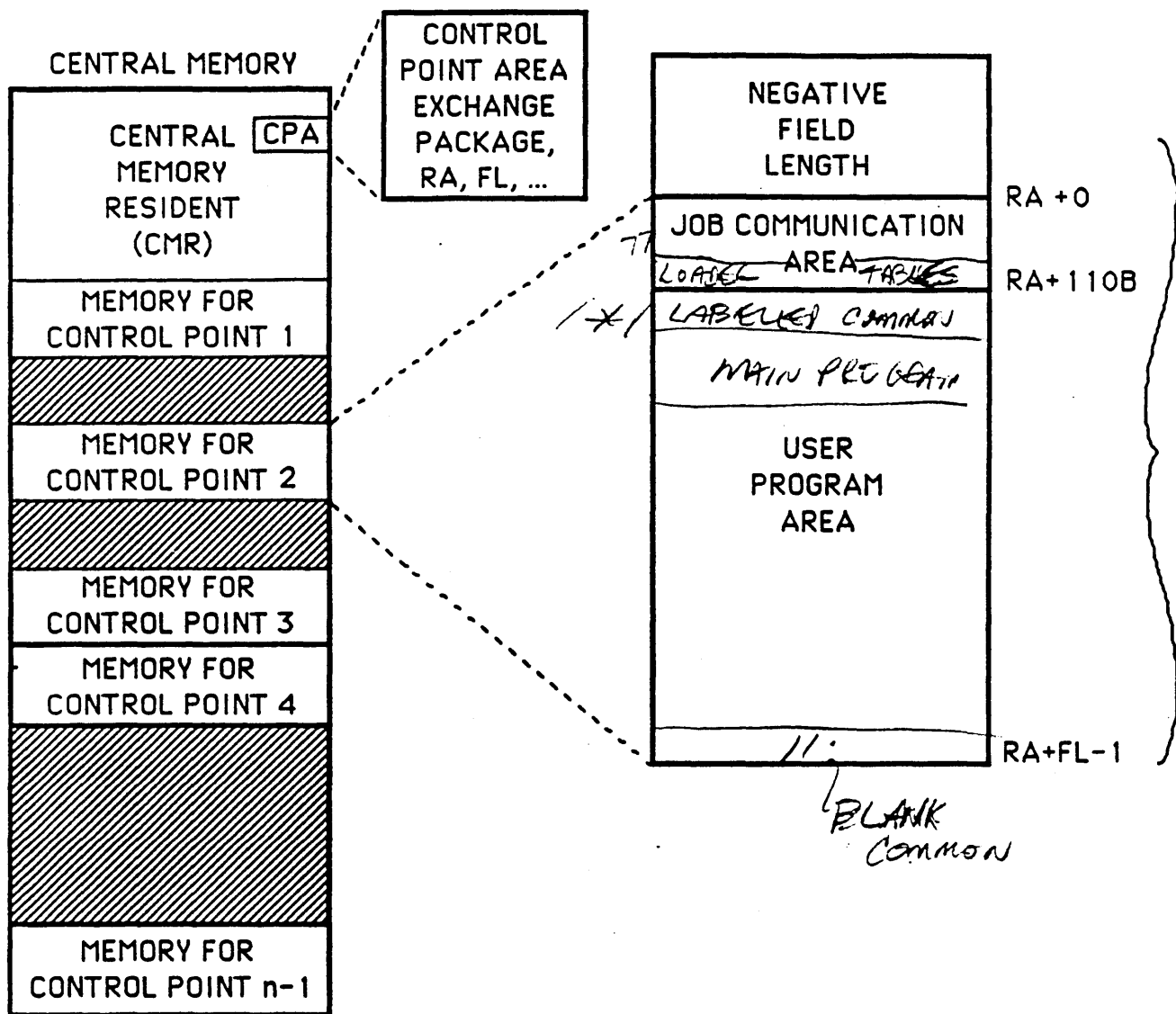
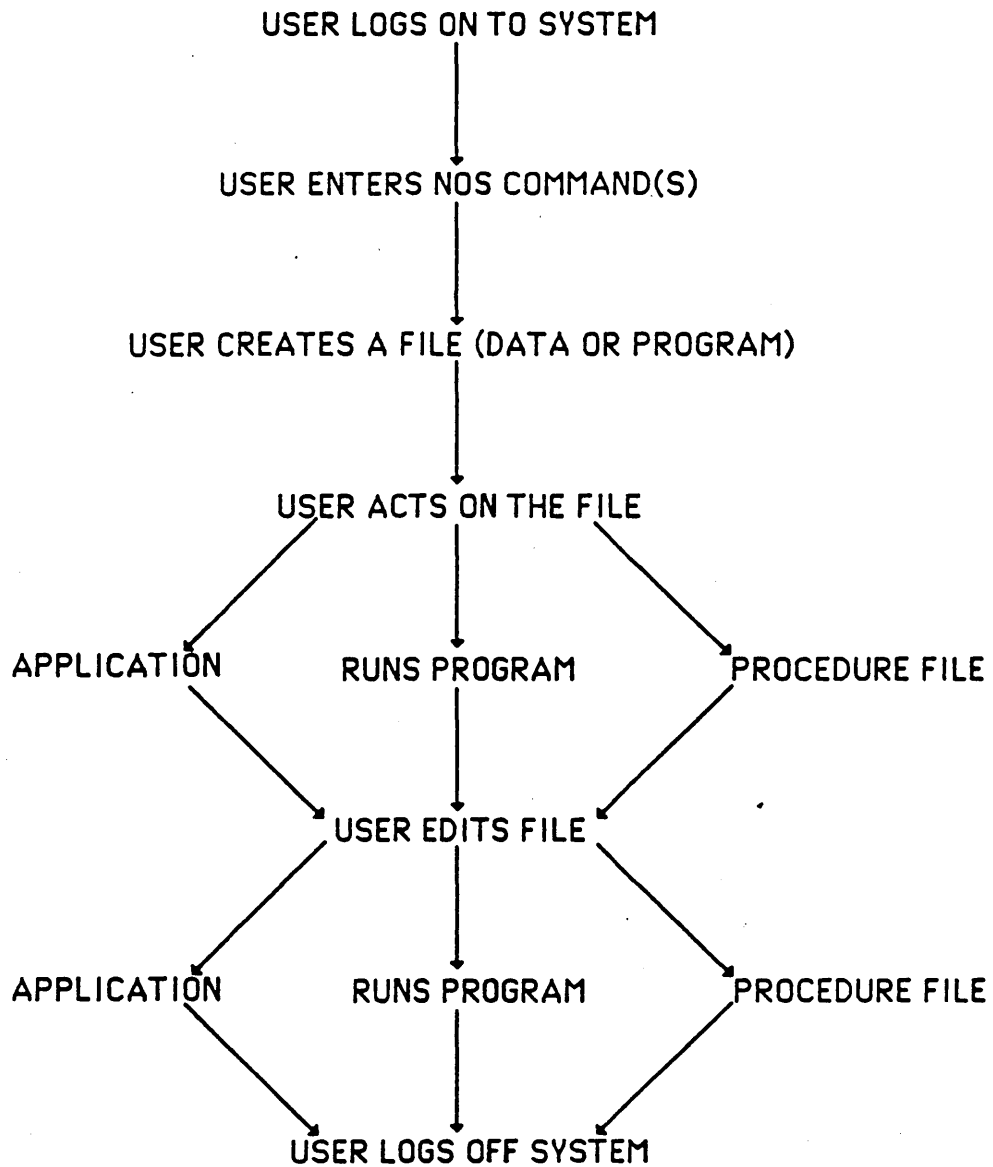


Figure 1-6. Control Point Concept

INTERACTIVE JOB FLOW



2550 LOG-IN PROCEDURE

1. Dial number.
2. When connected:
 - Enter Security Character (Optional)
 - Enter two carriage returns, then replace phone on cradle.
3. System responds FAMILY:
ADV, USERNAME, PASS, IAF
 - Enter family name
4. System responds USERNAME:
 - Enter username
5. System responds PASSWORD:
 - Enter password
6. System responds PERSONAL IDENTIFIER: (Optional)
 - Enter your identifier
7. System responds APPLICATION: (Optional)
 - Enter IAF
8. System responds CHARGE: (Optional)
 - Enter charge, charge number, project number
9. System responds with a "/" prompt. NOS is ready for NOS commands such as LIMITS which displays your username privileges.

NOTES:

1. Enter BATCH if system responds with "READY".
2. System will respond "IDLE" if no command is in process.
3. Enter cntl-T to abort a command.

CDCNET LOG-IN PROCEDURE

INTERACTIVE LOGIN PROCEDURE

1. Dial number.
2. When connected, enter two carriage returns, then replace phone on cradle.
3. The system will respond with a display similar to the following:

1200 bps ASCII, parity: even

Copyright Control Data Corporation 1985, 1986

DI System Name is 08002530009F, TDI_9F

Terminal Name is 43000, \$CONSOLE_30009F_4300000000

You may enter CDCNET commands.

Enter CREATE_CONNECTION _____
or CREC _____

CREC, PDS

4. System responds:

CREC ARH907

Connection \$A created.

5. Continue with step 3 (and following) as specified under 2550 LOG-IN PROCEDURE.

INTERACTIVE SESSION TERMINATION

- With phone disconnect
 - LOGOUT or
 - BYE or
 - GOODBYE
- Without phone disconnect
 - HELLO or
 - LOGIN

SWITCHING NETWORK APPLICATIONS

APPSW, application.

- Example:

/APPSW, TAF

Switch to Transaction Facility Application
without terminating the current session.

- Terminal Verification.

/APPSW,TVF

Switch to Facility.

JOB SEQUENCE NAME (JSN)

- Every job and queued file is assigned a unique four character name defined by the system. The JSNs are the primary job identifiers for the system and user.

REMOTE BATCH JOB FLOW

- There are five general phases of a Remote Batch Job's passage through the system.
- Phases of Remote Batch job processing:
 1. Entry into the system's Input queue.
 2. Assignment of resources by the system.
 3. Job execution without terminal involvement.
 4. Termination of the job.
 5. Queuing of the output for retrieval by the user.
- Activity Logs:
 - Job Dayfile - contains a report of commands entered and significant informative and error messages.
 - System Dayfile - contains a system-wide report of significant activities.
 - Account file - contains system-wide billing and statistical data.

EXERCISE 1.1

STUDY QUESTIONS

1. Name two of the subsystem programs that NOS controls simultaneously, in addition to its regular functions as an operating system.

IAF =

NAM =

2. Name the report maintained by NOS that details command processing and accounting information about a specific job.

3. Place the letter identifying each of the following devices in the blank next to its function.

a. Peripheral Processor C Contains jobs being executed.

b. Central Processing Unit A Transfers information for input/output equipment

c. Central Memory B Primary arithmetic device

4. Place the letter identifying each of the following items in the blank next to its function.

a. Field Length _____ A data image in central memory that enables jobs to setup registers within the central processing unit.

b. Control Point Area

c. Exchange Package _____ Part of Central Memory Resident that contains job name, field length, reference address, and other parameter values related to the user program.

_____ Part of central memory, allocated by NOS, where a job resides while in execution.

EXERCISE 1.1 (CONT)

5. Number the following events in the order in which they occur, if the computer is to execute a job and then print its output.

- _____ The output goes through a peripheral processor for printing on the terminals' screen.
- _____ A command is read from a terminal through a peripheral processor and onto disk storage.
- _____ A command output on disk is determined to belong to the terminal.
- _____ The command's job is assigned a control point number and central memory storage.
- _____ NOS determines the program needed for the command, central memory space, and other characteristics.
- _____ The program alternately shares use of the central processor with up to 34 other programs.

6. Number the following login parameters in the order in which they will be requested.

- _____ IAF
- _____ User name
- _____ Two carriage returns
- _____ Personal Identifier
- _____ Family name
- _____ Password
- _____ Charge number, project number

7. Name two ways in which a job may enter the system?

EXERCISE 1.2

DEFINITIONS REVIEW (Optional)

Define the following:

1. Central Processing Unit
2. Central Memory
3. Extended Memory
4. Peripheral Processor
5. Channel
6. Peripheral Equipment
7. Central Memory Resident
8. User Program's Load Address
9. Control Point Area
10. Job Communication Area
11. Reference Address
12. Field Length

EXERCISE 1.3

LOG-IN TO NOS

LOGIN VALUES:

| | | |
|---------------------|---|-------|
| Security Character | = | _____ |
| CDCNET Connection | = | _____ |
| NOS Family | = | _____ |
| NOS UserName | = | _____ |
| Username Password | = | _____ |
| Personal Identifier | = | _____ |
| NAM Application | = | _____ |
| Charge Number | = | _____ |
| Project Number | = | _____ |

Try the following command sequence:

```
/LIMITS.  
/TRMDEF,PG=Y,PL=25.  
/LIMITS.  
/Carriage/Return (RETURN or NEXT key)  
/APPSW,TVF  
/LOGOUT
```


LESSON TWO

**PERMANENT FILE CONCEPTS
AND COMMANDS**

LESSON TWO

PERMANENT FILE CONCEPTS AND COMMANDS

Lesson Preview:

In this lesson, you will be shown the difference between the two types of permanent files in the system, when it is appropriate to use either, how to create permanent files using various privacy and access mode parameters, how to access permanent files under one's own or another's username, and how to purge permanent files.

Objectives:

After completing this lesson, the student will be able to:

- Correctly enter commands using the appropriate command syntax
- Explain the concept of indirect access permanent files and use the applicable commands (SAVE, GET, REPLACE, APPEND)
- Explain the concept of direct access permanent files and use the appropriate commands (DEFINE, ATTACH)
- List, describe, and use other permanent file commands (CATLIST, PURGE)

Projects: Exercise 2.1 (Permanent File Commands)

References:

- NOS Reference Set Manual, Volume 2 and 3, Permanent File Sections

FILE CONCEPTS

- **Local File** - Any file assigned to a user is job while logged into a terminal
- **Permanent File** - A file on mass storage having a catalog entry
 - Direct
 - Indirect

What NOS calls a catalog entry would be called a directory entry on most other systems.

- To use a permanent file, you must make it local.
- The local and the permanent file names may be different.
- File names are 1-7 alphanumeric characters.
- File names should begin with a letter.
- All files in NOS are initially rewound.
- The local file designated on a PRIMARY command is always rewound before each command.

INDIRECT ACCESS - PERMANENT FILES

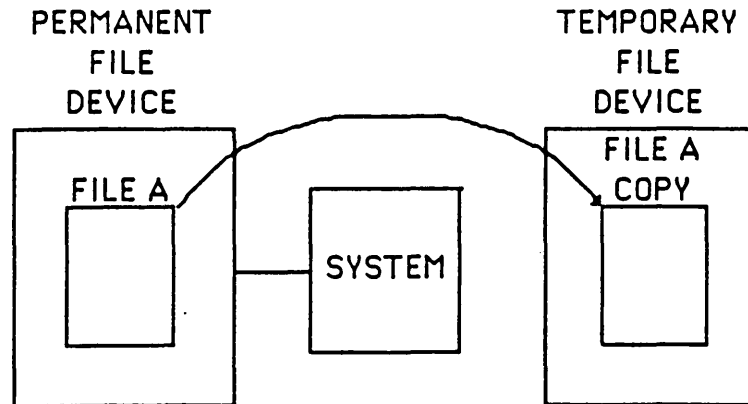


Figure 2.1 Method of Indirect Access

The only time that the system knows there is any link between file A and its copy is when the actual copy is performed. After the copy is complete, the system has no idea that there was any connection between the original file and its copy.

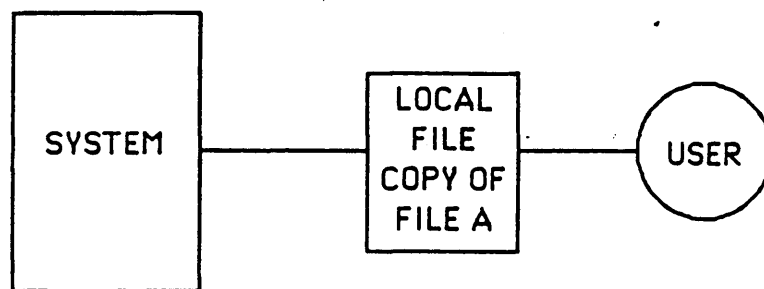


Figure 2.2. After Copy Made

The inherent advantage here is that regardless of what is done to the content of the local file, the permanent file remains unaltered and intact. One cannot accidentally damage the contents of the permanent file during local file operations.

DIRECT ACCESS - PERMANENT FILES

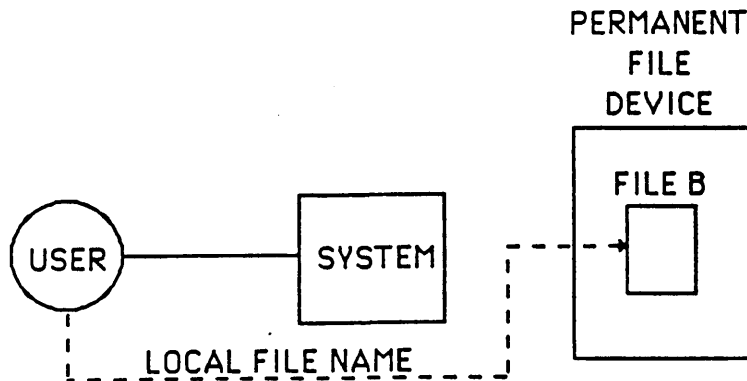


Figure 2-3. Method of Direct Access

No copy is made of the direct access permanent file. The permanent file is attached directly to the user's job and referenced by a local file name. At job termination, the pointer between the local and the direct access permanent file is removed, with the permanent file remaining in the same state it was at job termination time. While the file is attached to the user's job, any alteration made will be permanent, with no method of recovery of the original contents of the file. Thus one must use backup techniques to avoid having to rebuild important areas of the file.

PERMANENT FILE TYPE COMPARISONS

INDIRECT ACCESS

DIRECT ACCESS

- | | |
|---|---|
| <ol style="list-style-type: none"> 1. Work with a copy of the permanent file. 2. Created first and then made permanent by <u>SAVE</u>. 3. Accessed by <u>GET</u>. 4. Space allocated by 'sector' when not in use. 5. No 'write-interlock'. 6. File cannot be empty. 7. Local file is created in write-mode by default. 8. GET gets a local file in write-mode by default. | <ol style="list-style-type: none"> 1. Work with the file itself. 2. Both created and made permanent by <u>DEFINE</u>. 3. Accessed by <u>ATTACH</u>. 4. Space allocated by 'track' when not in use. 5. If one user has it attached in write mode, another user cannot attach it. 6. File can be empty. 7. Permanent file is created in write-mode by default. 8. ATTACH opens a local file pointing to the permanent file's space in read-mode by default. |
|---|---|

LPRU
64,000 words

W
Modify
Append
Read
E
Null

Default mode for Attach is Read
i.e. Attach in mode - Write

PURGE
PURGEALL (TY=IA, DA)

ACCESSING PERMANENT FILES

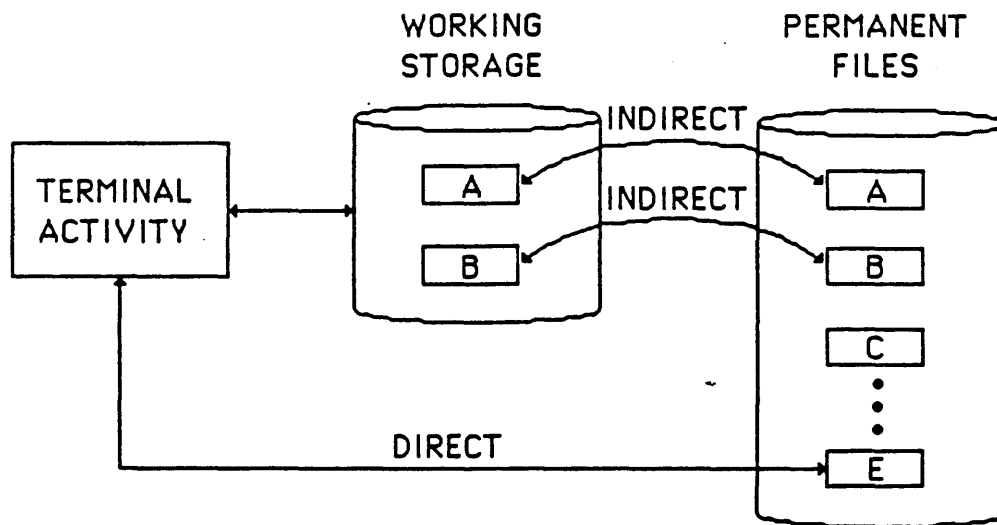


Figure 2-4. Permanent File Comparison

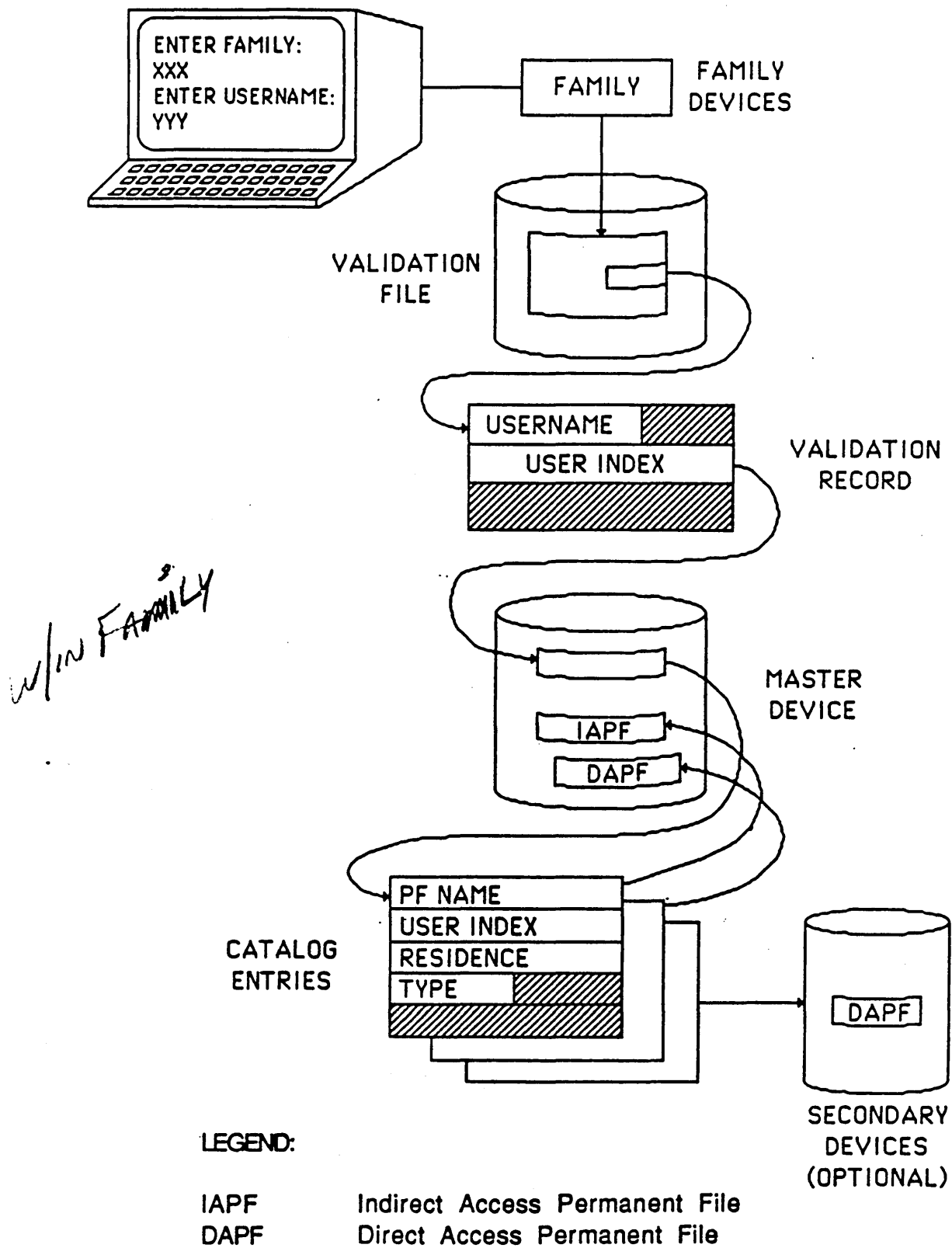


Figure 2-5. Permanent File Ownership and Storage Elements

PERMANENT FILE CATEGORY TYPES

- Private (CT=P; default)
 - Alternate users must be granted specific access permission via a PERMIT command
 - Originator may find out who accessed these files via CATLIST
- Semi-private (CT=SPRIV)
 - Alternate users can access file without being PERMITTED
 - PERMIT command can be used to set explicit permission levels
 - Originator may find out who accessed these files via CATLIST
- Public (CT=PU)
 - Alternate users can access file without being PERMITTED
 - PERMIT command cannot be used
 - Originator cannot find out who accessed these files
- Examples:

/CATLIST,FN=MYFILE,LO=FP.

for full permit list on file MYFILE (FN keyword is required)

/CHANGE,BIN1/AC=Y,CT=PU.

Public file BIN1 can be catlisted by anyone.

/CATLIST,UN=SMITH.

to catlist alternate user's permanent files

PERMANENT FILE SECURITY

(DIRECT AND INDIRECT ACCESS FILES)

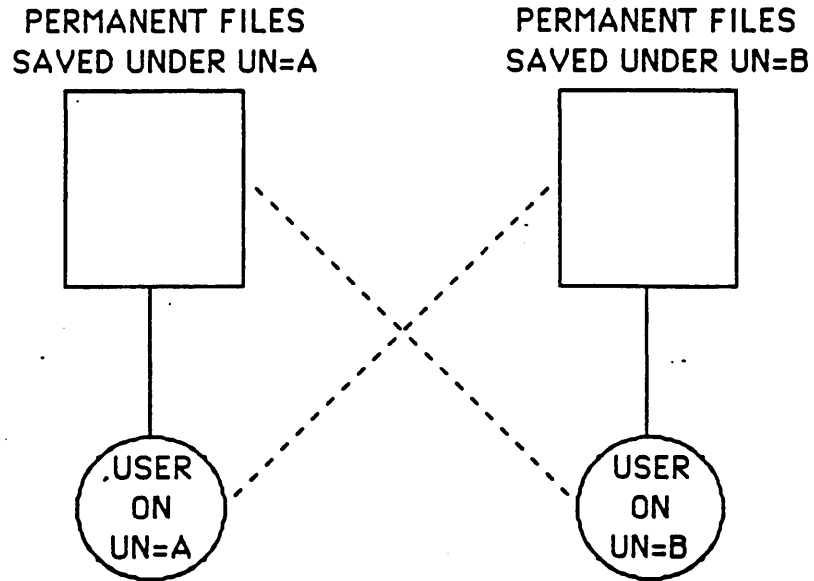


Figure 2-6. Permanent File Security

Permanent file protection only on dotted line accesses. Anyone logging in on a given user name may purge any file on that same user name regardless of access privileges defined for that file.

Therefore, never give anyone your username's password. Do not use it as a file password. Do not permit others to remote batch job files and procedure files with USER statements in them.

| SPRIV | PUBLIC |
|--------------------|-------------|
| WHO HOW WHEN | HOW MANY |

PERMISSION CATEGORY - PRIVATE

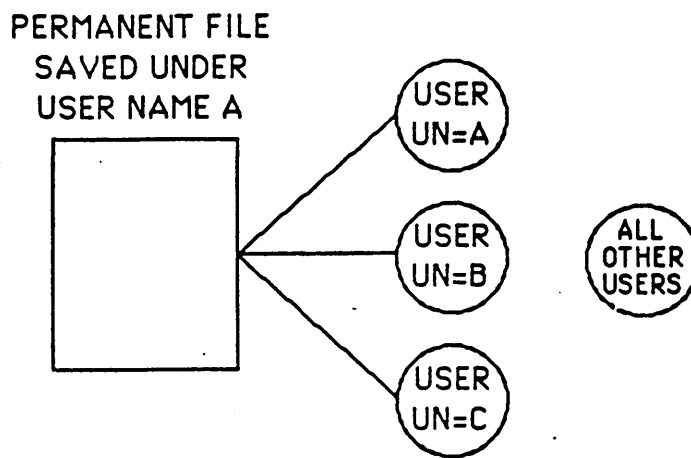


Figure 2-7. Accessing PRIVATE Files

User name A has access to the file, since the file was saved under their user name. User name A may execute, read, append or write on the file regardless of the access privileges defined when the file was saved. Any user who may perform the same operations on the file as User A is said to have ultimate privileges.

In this case, Users B and C have been given explicit permission to access the file by User A. In order to accomplish this, User A used either PERMIT command or control statement. Only users given explicit permission via the PERMIT command are allowed access to a private permanent file. Even though they have explicit permission, however, Users B and C must still know that the file is saved under User name A and the file password, if any, which A has placed on the file in order to actually access it.

Examples:

USER A: /PERMIT,pfn,B=W,C=R.
/CHANGE,pfn/PW,AC=Y.
?Secret

USER B: /GET,pfn/PW=SECRET. *UN=A*

USER C: /GET,pfn/PW=SECRET. *UN=A*

where user B,C get the file in write mode. Only user B allowed to replace the file in user A's catalog of permanent files.

PERMISSION CATEGORIES - SEMIPRIVATE AND PUBLIC

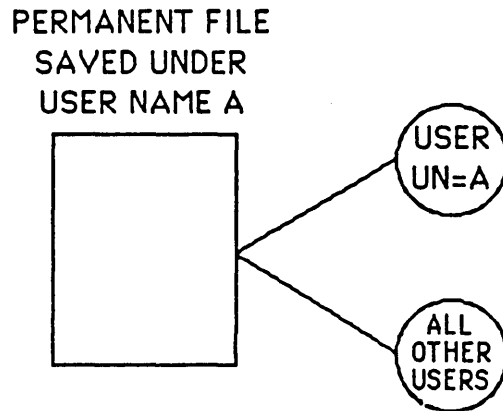


Figure 2-8. Accessing Semiprivate or Public Files

Again, User name A has ultimate privileges since the file was created under that user name. All others may access the file, provided they know the relevant protection information, i.e., the user name that the file was created under and the file password, if any. If the file is categorized as semiprivate, full bookkeeping as to user names who have accessed the file, is recorded by NOS.

USER A: /CATLIST.

USER B: /CATLIST,UN=A.

SYNTAX FOR COMMAND STATEMENTS

- Special characters: \$ \ *
- Command
- Separator , (+ " - /
- Blanks ignored
- Parameter - Keyword or keyword=value
- Terminator .)

Example:

/SAVE(FILEL=PERM/CT=PUBLIC)

Permanent file PERM will be saved as a permanent version of the local file FILEL. The slash indicates the end of the local=permanent file name list. PUBLIC is the value for the keyword CT.

Use HELPME, command to learn formats
of NOS commands.

PERMANENT FILE COMMAND SUMMARY

| <u>COMMAND</u> | <u>DIRECT</u> | <u>INDIRECT</u> | |
|----------------|---------------|-----------------|------------------------|
| APPEND | | X | <u>Add data at end</u> |
| ATTACH | X | | Access |
| CATLIST | X | X | Directory |
| CHANGE | X | X | Rename |
| DEFINE | X | | Create |
| GET | | X | Access |
| PACKNAM | X | X | Auxiliary packs |
| PERMIT | X | X | |
| PURGALL | X | X | Delete all |
| PURGE | X | X | Delete |
| REPLACE | | X | Alter data |
| SAVE | | X | Create |
| OLD | | X | Declare primary file |
| RECLAIM | X | X | Dump/Load |

Figure 2-9. Permanent File Commands

COMMONLY-USED COMMON PARAMETERS

CT - File Permit Category

P, PR, or PRIVATE
S or SPRIV
PU or PUBLIC

*Roll out
& insert*

NA - No abort option; error handling

- * Temporary error such as direct access file busy or requested auxiliary device busy - job processing is suspended until error is corrected.
- * Permanent error such as non-existent file or device - job processing continues with next command.

PN - Pack name

1-7 character pack name of auxiliary device; used in conjunction with "R" parameter which specifies device type.

PW - Password

1-7 character file password.

**NOTE: NEVER USE YOUR USERNAME PASSWORD
AS A PERMANENT FILE PASSWORD.**

AC - Y or N

Allow CATLIST (Y/N).

SS - Subsystem

Interactive subsystem to be associated with a file; default is null.

BASIC

BATCH - Recommended

EXECUTE

FORTTRAN - *AUSI 77*

FTNTS - *KN AUSI 66*

NULL

COMMONLY-USED COMMON PARAMETERS (CONT)

UN - Username

Retrieve a permanent file from another user's catalog. To do this, you must be validated by file creator (PERMIT command or CT parameter).

WB - Wait-if-busy

Wait for busy files or mounting of disk packs; will abort on other errors; cannot use with "NA" option.

Example: /ATTACH,FILE/M=W,WB.

XD - Expiration date

Expiration date for file password or file permit; format: YYMMDD
Must be authorized to use this parameter (see LIMITS).

XT - Expiration time

Life of file password or file permit in days

4095 or * - Infinite expiration date

Ø - Expires immediately

Default is site-defined

Must be authorized to use this parameter (see LIMITS).

Example: /PERMIT,DEMO,SALES=E/XT=10.

Allows username SALES to execute file DEMO for 10 days.

INDIRECT ACCESS

FILE COMMANDS

SAVE COMMAND

- Makes a copy of the specified local file on mass storage as an indirect access permanent file.
- Permanent file cannot already exist.
- Can save more than one file with a single SAVE command.
- Local file cannot be empty (zero length).
- FORMAT:

SAVE, lfn₁=pfn₁, ..., lfn_n=pfn_n/params

Where:

lfn Name of local file to be saved

pfn Permanent file name if different than local file name

params Optional parameters

Common Parameters

CT File permit category
NA No abort option
PN Alternate pack name
PW Defines the file password

Command Specific Parameters

AC = Y or N (default unless changed by installation), indicates whether alternate users can obtain information about this file using the CATLIST command.

M = APPEND, A
 EXECUTE, E
 MODIFY, M
 NULL, N
 READ, R
 WRITE, W

Indicates the mode in which alternate users can access the file.

SAVE COMMAND EXAMPLES

Save 2 files

Examples:

/SAVE,JUSTONE.

/SAVE,THESE,FILES.

/SAVE,LOCAL=PERM/CT=PUBLIC,PW=MYPASS,M=READ,NA.

/SAVE,THIS=THAT,HERE=THERE,NOW=THEN.

/ENQUIRE,O=FILE6.

/SAVE,FILE6/PW.

?SECRET

Save ENQUIRE output.

SAVE prompts for password value since PW format was used in place of PW=SECRET format.

REPLACE COMMAND

- Replace or create an indirect access permanent file on mass storage using a local file.
- Can replace more than one file with a single REPLACE command.
- Local file cannot be empty (zero length).
- FORMAT:

REPLACE, lfn₁=pfn₁, ..., lfn_n=pfn_n/params

Where:

lfn . Name of local file to be saved

pfn Permanent file name if different than local file name

params Optional parameters

NA No abort option

PN Alternate pack name

PW Specifies an alternate user's file password when replacing a file in an alternate user's catalog.

UN Alternate username

- Examples:

/REPLACE, WINTER.

/REPLACE, WINTER=SUMMER, SPRING=FALL.

/REPLACE, WINTER/UN=SEASONS, PW=WEATHER.

File WINTER will be replaced in alternate user's catalog if SEASONS gave this user WRITE mode permission and the permanent file ~~SEASONS~~ exists there.

WINTER

GET COMMAND

- Gets copy of an indirect access permanent file and makes it local.
- Can specify more than one file to retrieve at once.
- FORMAT:

GET, lfn₁=pfn₁, ..., lfn_n=pfn_n/params

Where:

lfn Name of local file to be opened

pfn Permanent file name if different than local file name

params Optional parameters

| | |
|----|---------------------|
| NA | No abort option |
| PN | Alternate pack name |
| PW | File password |
| UN | Alternate username |
| WB | Wait-if-Busy option |

- Example:

/GET,TWEETY,BUGS,WILEY.

/GET,BUGS=BUNNY/PW=CARROT,UN=DISNEY,PN=SAM,NA.

Permanent file from pack named SAM and owned by DISNEY will be copied to local file BUGS. GET will wait for pack SAM to be mounted (NA).

/ENQUIRE,F

/CATLIST

/GET, LFILE6=FILE6

/ENQUIRE,F

Check for FILE6 in catalog.

Note local file name and type.

APPEND COMMAND

- Add specified local file(s) to the End-of-Information of an indirect access permanent file.
- Get the permanent file to see the effect of APPEND.
- FORMAT:

APPEND,pfn,lfn₁,...lfn_n/params

Where:

pfn Permanent file name of indirect access file where appended files will be written.

lfn Local file name(s) of file(s) to be appended

params Optional parameters

NA No abort option

PN Alternate pack name

PW File password

UN Alternate username (requires WRITE mode permission)

WB Wait-if-Busy option

- Examples:

/APPEND,PERM,ONE,TWO,THREE.

/APPEND,PERM,LOCAL1,LOCAL2/UN=OTHERU.

Note the order of file names: permanent file name is first to allow a list of local file names.

/GET,PERM

/APPEND,PERM,PERM

/LENGTH,PERM

/GET,PERM

/LENGTH,PERM

Local file is not larger.

Permanent file is larger.

DIRECT ACCESS

FILE COMMANDS

DEFINE COMMAND

- Creates an empty direct access permanent file
- Usually can be larger than indirect access permanent files
- Can also define an existing local file as a direct access file if the local file was assigned to a mass storage device on which you can have a direct access permanent file
- Can create several direct access files on a single define command
- FORMAT:

DEFINE, lfn₁=pfn₁,...,lfn_n=pfn_n/params

Where:

lfn Name by which the permanent file space will be referenced as a local file

pfn Permanent file name if different than local file name

params Optional Parameters

Common Parameters

CT File permit category

NA No abort option

PN Alternate pack name

PW File password

WB Wait-if-Busy option (e.g., pack not yet mounted)

DEFINE COMMAND (CONT.)

PARAMS (cont.)

Command Specific Parameters

- AC Allow catlist, same as indirect access permanent files
- M EXECUTE,E
MODIFY,M
NULL,N
READ,R
WRITE,W
Indicates the mode in which alternate users can access the file.
- S SIZE in disk sectors

- Examples:

/DEFINE,APPLE,ORANGE/M=W,AC=Y,CT=SPRIV.

Remember M=W is required to allow others to write on the file.

/DEFINE,LEMON=LIME/CT=SPRIV,M=NULL,AC=Y,PW=CITRUS.

File LIME is semiprivate but no one can use it unless explicitly permitted with the PERMIT command.

DA FA
1
1

/GET,FILE6

/DEFINE,FILE8/CT=PU,AC=Y

/LENGTH,FILE8

See empty status

/COPYEI,FILE6,FILE8

/COPYEI,FILE8

Forgot to rewind

/REWIND,FILE8

/COPYEI,FILE8

Contents now seen

/CATLIST

See permanent file names

/ENQUIRE,F

See file names, positions, and types

ATTACH COMMAND

- Assigns a direct access permanent file to a job
- Can assign more than one file with a single ATTACH command
- FORMAT:

ATTACH, lfn₁=pfn₁,...,lfn_n=pfn_n/params

Where:

lfn Name of local file

pfn Permanent file name if different than local file name

params Optional Parameters

Common Parameters

NA No abort option
PN Alternate pack name
PW File password
UN Alternate username
WB Wait-if-Busy option

Command Specific Parameters

M EXECUTE,E
 MODIFY,M
 Rx - Read and share with one user having M=x
 READ,R - Share mode prevented
 WRITE,W - Share mode prevented
 Indicates the mode in which user, including the owner,
 is accessing the file.

Default

lfn/CT=P, M=R

ATTACH COMMAND EXAMPLES

- Examples:

/ATTACH,HULK,HOGAN/M=W,WB.

Attach will wait if file is in use.

/ATTACH,WEASEL=HEENAN/PW=WRESLER,UN=AWA,NA,PN=O.

Assuming a PACKNAM (MYPACK) was issued, PN=0 returns the user to Family permanent files.

/ATTACH,NOTES/M=RA,UN=LIBRARY.

User can read NOTES while another user appends more notes to NOTES. NOTES, we say, is attached in "share mode" by this user.

/CLEAR — *release/return all files*
/ENQUIRE,F
/ATTACH,FILE8A=FILE8
/ENQUIRE,F

Note "PM*" file type for FILE8A denoting FILE8A points to direct access permanent file and is opened in read (or execute only) mode.

/WRITER(OPEN)
/DEFINE,OPEN/NA
/CATLIST,FN=OPEN

Will only work if OPEN was opened on a device which this user can have direct access files.

PERMANENT FILE
COMMANDS APPLICABLE
TO BOTH INDIRECT AND
DIRECT ACCESS FILES

CATLIST COMMAND

- Lists information about your permanent files
- Can list information about other's files if you are validated for any file access and Allow - Catlist (AC=Y) is granted by owner
- FORMAT:

CATLIST,params.

Where:

| <u>params</u> | <u>Optional parameters</u> |
|---------------|--|
| FN= pfn | List information for this permanent file |
| LO= F | Full list options |
| FP | File permissions defined for the permanent file specified on the FN parameter |
| P | Users permitted to access the private or semi-private file specified by the FN parameter |
| L= lfn | Local file to receive report (OUTPUT default) |
| NA | No abort option |
| PN= pn | Alternate packname |
| UN= un | Alternate username |
| WB | Wait-if-Busy option (e.g., Pack not mounted) |

CATLIST COMMAND EXAMPLES

Examples:

/CATLIST. Gives names of all of your permanent files.

/CATLIST,LO=F. Gives full list options on all your permanent files. Owner sees password values.

/CATLIST,LO=FP,FN=APPLIC.

1. For PRIVATE and SEMI-PRIVATE files only.
2. Specifies who has been granted access with a PERMIT; specified by an *.
3. Gives list of alternate users accessing files. Includes number of accesses, date and time of last access.
4. Filename is required.
5. You must be the file's owner.

/CATLIST,LO=F,FN=XYZ.

Password values are included in the listing since these are your files. Gives full list options on only your file 'XYZ'.

/CATLIST,LO=F,UN=SMS25.

Gives full list options on all files in SMS25's catalog that YOU MAY ACCESS and that have AC=Y set. Will NOT list passwords associated with the files.

/CATLIST,FN=FILE6,LO=F.

CHANGE COMMAND

- Change permanent file characteristics
- FORMAT:

CHANGE, nfn₁=ofn₁,...,nfn_n=ofn_n/params

Where:

nfn New permanent file name

ofn Old permanent file name

params Optional Parameters

Common Parameters

CT File permit category

NA No abort option

PN Alternate pack name

PW File password (PW=0 to remove)

WB Wait-if-Busy option

XD Expiration date of password (yymmdd)

XT Expiration time of password in days (0-4095,*)

Command Specific Parameters

AC = Y or N, Allow CATLIST.

M = APPEND,A

EXECUTE,E

MODIFY,M

NULL,N

READ,R

WRITE,W

Indicates the mode in which alternate users can access the file.

CHANGE COMMAND EXAMPLES

- Examples:

/CHANGE, NOW=BEFORE, NEW=PREV/AC=Y.

NOTE: Format is newname=old name.

/CHANGE, NEWFILE=OLDFILE/PW=NEWPASS, CT=PRIVATE, M=READ,
XD=860615.

Note that the format of XD date is YYMMDD.

/CHANGE, APPLICS, APPLICB/PW.
? 0

PW without value causes a read from INPUT (terminal). A value of zero will remove any existing file password security.

/CATLIST, FN=FILE1, LO=F
/CHANGE, NEWNAME=FILE1/PW=NEWPASS
/CHANGE, NEWNAME/CT=SPRIV, M=NULL
/CATLIST, LO=F, FN=FILE1/NA.ERROR
/CATLIST, LO=F, FN=NEWNAME
/PERMIT, NEWNAME, ASE1230=R/XT=*.
/CATLIST, FN=NEWNAME, LO=FP.

PACKNAM COMMAND

- All subsequent permanent file commands in the job are directed to the specified auxiliary pack unless PN=0 is specified on the command.
- Use of auxiliary packs allows file sharing across families.
- FORMAT:

PACKNAM,PN=packname,R=n.
PACKNAM,packname,R=n.
PACKNAM.

Where:

PN 1-7 character identifier of auxiliary pack

R Device type of auxiliary pack

- Examples:

/PACKNAM,PN=AUXPACK. Installation-defined device type is the default.

/GET,DOCLIST. DOCLIST on AUXPACK is used.

/PACKNAM. Clears the effect of the previous PACKNAM command; returns to family system devices.

/GET,DOCLIST. DOCLIST on family device is used.

/REPLACE,DOCLIST/PN=AUXPACK.
Local file DOCLIST is saved as indirect file DOCLIST on the auxiliary pack AUXPACK.

PERMIT COMMAND

- Allows another user to access private files in your permanent file catalog
- Can also change the mode in which another user can access a semiprivate file
- **FORMAT:**

PERMIT,pfn,username1=m1,...,username_n=m_n/params

Where:

| | |
|----------|--|
| pfn | Permanent File Name (one name only) |
| username | Username of user who is being permitted to use the file (Note that format does not include UN=) |
| m | Mode in which alternate user may access the file (Note that format does not include M=) |
| params | Optional parameters |
| PN | Packname |
| NA | No abort option |
| WB | Wait-if-Busy option |
| XD | Expiration date |
| XT | Expiration time |

PERMIT COMMAND EXAMPLES

- Examples:

YIP
YIP
/PERMIT,TEST1,TOSDEH=WRITE,TOSTMF=NULL/XD=890215,NA.

Permission for user TOSTMF has been revoked.

/PERMIT,TEST2,TOSJEB=READ/PN=ACTPACK,WB.

Will wait for pack to be mounted.

/PERMIT,FILE8,ASE1230=W/NA.

/PERMIT,FILE6,ASE1230=E/NA.

/CATLIST,FN=FILE6,LO=P.

To see permitted users.

/CATLIST,FN=FILE8,LO=FP.

To see full permits.

PURGE COMMAND

- Removes files from the permanent file device
- Indirect file access permanent file space is released immediately
- Direct access permanent file space is released immediately if not attached by any user. Else DAPF space is released when the attach count goes to zero.
- FORMAT:

PURGE,pfn₁,...,pfn_n/params.

Where:

pfn Permanent file name to be removed

params Optional parameters

NA No abort option

PN Alternate pack name

PW File password

UN Alternate username

WB Wait-if-Busy options (e.g. Pack not mounted)

PURGE COMMAND EXAMPLES

- Examples:

/PURGE,JUSTONE/NA.

Continue to next command unconditionally (NA).

/PURGE,TWO,THREE,FOUR/PN=ALTPACK,WB.

/PURGE,FIVE/UN=OTHUSER,PW=LIVER.

Will work if user has WRITE mode permission.

/CATLIST Shows file FILE6,FILE8

/ATTACH,FILE8/M=W

/GET,FILE6

/PURGE,FILE6,FILE8

/ENQUIRE,F

/CATLIST

/DEFINE,FILE8. Note missing files. Will this DEFINE work?

PURGALL COMMAND

- Purges all permanent files in your catalog that satisfy certain criteria
- The TY parameter is required if no other candidate criteria is specified.
- FORMAT:

PURGALL,AD=ad,AF,CD=cd,CT=ct,DN=dn,MD=md,NA,PN=packname,
R=r,TM=tm,TY=ty,WB.

Where:

Common Parameters

CT File permit category
NA No abort option
PN Alternate packname
R Device type of alternate pack
WB Wait-if-Busy option

Command Specific Parameters

AD Access Date; purge all files accessed before this date


AF Purge all files after the date specified on AD, MD, or CD parameter

CD Creation Date; purge all files created before this date

DN Device Number assigned to a device at system initialization (intended for usage by site personnel)

MD Modification Date; purge all files modified before this date

TM Time modified; purge all files modified before this time

 TY File type: I or Indir, D or Direct, A or All

PURGALL COMMAND EXAMPLES

- Examples:

/PURGALL, TY = ALL Purge all files.

/PURGALL, TY=D, MD=850601, TM=120000.

Purge all direct access files modified before
12:00 noon on June 1, 1985.

/PURGALL, CD=850630, AF, PN=altpack, WB.

Purge all files created after (AF specified)
June 30, 1985 on auxiliary pack ALTPACK.
Wait until the pack is available or until all
files are no longer busy to perform the purge.

RECLAIM COMMAND

- Allows users to manage backups of their permanent files on a dump file.
- The dump file can reside on a labeled tape or be a direct access file itself.
- RECLAIM activity history is saved on a direct access database named RECLDB by default. The VSNs of dump tapes are maintained in the database.
- FORMAT:

RECLAIM, DB=pfn,l=lfm,PW=pw,UN=un,NV,Z.

Where:

| | |
|----------------|--|
| pfn | Name of RECLAIM dump file (default RECLDB). |
| l=lfm | Name of local file containing RECLAIM directives (default is INPUT unless Z is specified). |
| Z | Directives appear as comments on RECLAIM command. |
| L=lfm | Report file name (default OUTPUT). |
| PW=pw UN=un | Used when data base is owned by another user and you are permitted in write mode (M=W). |
| NV | New volume (or permanent file) is to be used as the dump file. |

RECLAIM DIRECTIVES

| | |
|----------|---|
| COMPACT | Rewrites a dump file using only the active files in the database. |
| COPY | Creates local file copies of all dumped files meeting specified criteria. |
| DELETE | Disables all files in the database that meet the specified criteria. Space is not released. |
| DUMP | Dumps specified files to tape or mass storage. |
| END/QUIT | Terminates current RECLAIM session. |
| LIST | Lists RECLAIM database information about specified dumped files. |
| LOAD | Loads specified files into you permanent file catalog. |
| REMOVE | Deletes on tape Volume Serial Number (VSN) from the database. |
| RESET | Reactivates all files previously disabled by the DELETE directive. |
| SET | Redefines RECLAIM defaults for any directive option. |

RECLAIM COMMAND

- See RECLAIM command in NOS Reference Set, Volume 3 for RECLAIM directive options. Note that candidate selections in general, increase qualification restrictions. The EX option reverses candidate selection criteria.

RECLAIM EXAMPLES

- Examples:

/DEFINE,TAPE.

ET
/RECLAIM,NV,~~EQ~~=NO,Z./DUMP,DT=MS.

Dump all files to direct access dump file TAPE (DF default). Current contents of TAPE will be evicted.

/CLEAR.

/ATTACH,TAPE/M=W.

/RECLAIM,Z./LOAD,PF=NEWLIB.

Reloads latest backup of NEWLIB if it does not yet exist.

/RECLAIM,Z./LOAD,RP=Y,PF=NEWLIB.

Same, but an existing NEWLIB will be replaced.

/RECLAIM,Z./LIST,UN=O,TY=A.

Backup file listing for all files (default) and all users whose files are dumped on TAPE.

LOCAL FILE

COMMANDS

LOCAL FILE COMMANDS

- The following local file commands are being introduced here to assist in this lesson's permanent file exercises.

- CLEAR
- CATALOG/ITEMIZE
- ENQUIRE, F - 1151 local files
- SETFS, file status

CLEAR COMMAND

- Releases all local files except those specified.
- Files will no longer be local.
- FORMAT:

CLEAR. Clears all local files

CLEAR,*,lfn₁,...lfn_n. Clears all files except those specified.

- Examples:

/CLEAR.

Alternate commands are "RETURN,*." and "UNLOAD,*."

/CLEAR,*,EXCEP1,EXCEP2.

SETFS COMMAND

- Set No-Auto-Drop (NAD) status for one or more local files so that CLEAR does not release them.
- FORMAT:

SETFS(lfn₁,...lfn_n/FS=fs)

Where

fs NAD for No-Auto-Drop or AD for Auto-Drop.

- Example:

/SETFS(OUTPUT,INPUT/FS=NAD)

OUTPUT and INPUT would have to be explicitly returned or unloaded.

CATALOG COMMAND

- Lists information about each record in a local file.
- FORMAT:

CATALOG,lfn,params.

Where:

lfn Name of local file to be cataloged

params Optional parameters

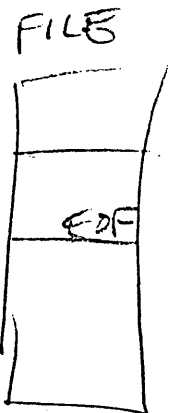
| | |
|--------|--|
| L=list | Local file to contain output from the catalog |
| N=n | Number of files to catalog |
| R | Rewinds lfn before and after catalog |
| U | Lists all records within a user library |
| T | Lists contents of deadstart decks on the file (for system analysts). |

- Examples:

/CATALOG,SEARS.

/CATALOG,PENNEYS,L=ORDER,R,U,N.

Rewind PENNEYS and list information including the contents of records in any user library (ULIB), of all files (EOFs) in the file. Remember the anagram 'RUN' for this useful format.



lfn.
lfn, N
lfn, N=4.

ENQUIRE COMMAND

- Provides information about your jobs in the system.
- FORMAT:

ENQUIRE
ENQUIRE,options.
ENQUIRE,OP=options,FN=lf1,O=lf2.
ENQUIRE,JSN=jsn,O=lf2
ENQUIRE,UJN=ujn,O=lf2.

Where OP =

options

- A Gives listings of B,D,F,J,L,R,U options (default)
- B Indication and priority information
Username
Family
Subsystem
CPU priority
- D Resources demanded (via RESOURC command) and resources assigned
- F Status of files assigned to your job
Filenames
File lengths in PRUs
File type
Status of files, e.g., EOR, EOI, BOI
- J Control register contents
Error flag field
Pending commands
- L Loader information such as Map option and global library file list

ENQUIRE COMMAND (CONT)

options (cont)

- R Resources used
 CPU time
 MS (Mass Storage) activity
 MT (Magnetic Tape) activity
 PF (Permanent File) activity
- S Accumulated SRUs (System Resource Units), derived from CP
 (Central Processor) time, I/O (Input/Output) activity, memory
 usage
- T Accumulated CPU time
- U Resources available for use
 Commands (no limit)
 Mass Storage
- UJN User Job Name; 1-line report of each job with specified UJN
 such as:
 - Job Sequence Number (JSN)
 - Job status - rolled out, executing, queue
- JSN Detailed report of job with specified JSN.
 Depending upon job, could list:
 - Job dayfile (up to 10 lines)
 - Succeeding commands
 /ENQUIRE,JSN. Produces a 1-line report on each job
 originating from or destined for your
 user name.
- FN = file Status of local file specified
- O = ofile File on which to write output from ENQUIRE; OUTPUT is
 default file

EXERCISE 2.1

PERMANENT FILE COMMANDS

1. _____ Create an Indirect Access permanent file with a password using the SAVE command.
2. _____ Create an Indirect Access permanent file using the REPLACE command.
3. _____ Create a Direct Access permanent file using the DEFINE command.
4. _____ Issue an ENQUIRE, F.

This command will list all local files. The files created in #1, 2, and 3 will appear in the output.

5. _____ Issue a CLEAR and an ENQUIRE, F.

The files created in #1, 2, and 3 will no longer be local files, nor will any other files which may have been local to your terminal.

6. _____ Issue a CATLIST, LO=F.

All of your permanent files and their attributes will be listed at your terminal.

7. _____ Issue the commands necessary to make several of your permanent files local to your terminal (GET and ATTACH), and issue an ENQUIRE, F.

The filenames used on the GET and ATTACH commands will appear as local files.

EXERCISE 2.1 (CONT)

8. _____ Issue the PERMIT command to allow several other usernames with access to your permanent files using different permission for each username (M=READ, M=WRITE, M=EXECUTE, etc.)
- _____ Issue a CATLIST, LO=FP, FN=fn.
- Permissions granted using the PERMIT command will be displayed.
9. _____ GET an indirect access file and CATALOG the local file copy, noting the file length in PRUs. Make several other files local and APPEND them to the indirect access file previously CATALOGed.
- _____ GET and CATALOG (CATALOG,lfn,R,N.) after the APPEND. Note the change in length (PRUs).
10. _____ CHANGE some of your permanent files' characteristics, e.g., mode (READ, WRITE), password. Issue a CATLIST, LO=F.
- The file characteristics just changed will be output to the terminal.
11. _____ PURGE some of the files you no longer need and issue a CATLIST.
- The files will no longer appear in your catalog listing.
12. _____ Use RECLAIM to make a backup of your permanent file on a direct access RECLAIM database. Use RECLAIM to list the contents of this database.

LESSON THREE

FULL SCREEN EDITORS

LESSON THREE

FULL SCREEN EDITOR (FSE)

Lesson Preview:

This lesson introduces the student the NOS Full Screen Editor.

Objectives:

After completing of this lesson, the student will be able to:

- Create , modify, and save a file using FSE
- Modify a file using the appropriate commands and save the changes (make the changes permanent)
- Request on-line HELP for individual FSE directives

Projects:

- Exercise 3.1 - FSE

References:

- NOS Full Screen Editor User's Guide

GETTING STARTED

1. Log on

2. Set up terminal in screen mode, e.g.,

/SCREEN,721T. For a 721 terminal with type ahead allowed.

3. Set up terminal in ASCII or NORMAL (upper case only) mode, e.g.,

/CSET,ASCII. or
/CSET,NORMAL.

4. Initiate FSE by entering

/FSE,lfn. to create or to edit a local file or

/FSE,lfn,G. to make the direct or indirect permanent file lfn a local
file

5. QUIT (or Q) is the FSE end directive.

6. Enter FSE without parameters to resume FSE where you left off.

NOTE: If SCREEN mode is not initiated, FSE
will respond with its name and a "??" prompt.
This is the line mode of FSE.

FSE DIRECTIVES - GENERAL

1. Enter directives:



In screen mode

??

In line mode

2. Abbreviate with first or first three characters
3. Spaces or commas required between numbers, but not between words.

Examples:

LONG VERSION

COMPACT VERSION

COPY 20 TO 50

C20T50

MOVE 20 50 TO 100

M20.50T100

LOCATE ALL WORD/XVAR/

LAW/XVAR/

4. Usually parameters may be specified in any order.

Examples: LAW/XVAR/
LA/XVAR/W
L/XVAR/AW

5. You may use ; to string directives together on a single line.

Example: L/This/; DeleteW; DW; VIEW

6. Directives usually default to Pointer (cursor) at last line changed unless overridden by directive parameters.

NOS FSE COMMAND EXPANDED

- Order-independent parameters

FN = lfn

OP = character set and/or get file

| | |
|---------|---|
| DISPLAY | Upper Case Only; use for NOS commands and |
| NORMAL | most compiler input. |
| ASCII | Example: OP = GA |
| ASCII8 | |
| GET | NOTE: Does GET/ATTACH in Write mode. |

I = Input directive file. Default is INPUT

O = Output listing file. Default is OUTPUT

IP = Enables user to change default procedure library from "FSEPROC".
So, on subsequent -PROC.NAME(FILE), if you do not specify file,
you get "FSEPROC".

. directive(s) NOTE: Some sites have added FSE command
procedures which have disabled this ability
to supply default directives on the FSE
command.

- Examples:

```
/FSE,FN=NEWFILE,OP=GA.SP2  
/FSE,NEWFILE,I=FSEDIRS.SL  
/FSE,NEWFILE,IP=721PROC.SPO
```

SCREEN MODE FSE KEYS

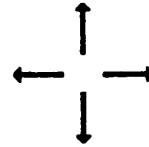
1. **HOME** Puts cursor at the directives line (Home line)

2. To exit FSE:

- A. **QUIT** Edited files remain local. Exception: Direct Access files in which changes are permanent

- B. **HOME** QR Makes everything permanent.

3. Arrow keys for cursor positioning:



NOTE: On most terminals, these keys are repeating.

4. Most corrections can be made by typing over the data in the work area of the screen and using function keys as marked on the keys themselves or on prompt line at the bottom of the screen (terminal dependent).

Enter **HOME** SET PROMPT 2 to see all key definitions.

5. Inserting characters:

A B **C** D
 ↑ Position to C and press **INSRT** key

A B C D
 ↑ Cursor positioned here

Note: On some terminals the INSERT key acts as a toggle key (insert mode ON/OFF).

6. Deleting characters:

A B C D
 ↑ Position C. and press DELETE

A C D : Result

7. Insert lines: (shift)



Delete lines: (shift)



Note that the top description on prompt lines refer to shifted function key actions.

8. Page forward and backward:



9. Moving lines (copy and delete):

A. Place cursor on first line to be moved and

MARK

B. Place cursor on last line and

MARK

C. Place cursor on the line which you want lines inserted before
and

MOVE

10. Copying lines:

Same as "MOVING", except

COPY

instead of

MOVE

11. Moving/Copying characters:

Same as moving and copying lines, except



MRKCHR

instead of

MARK

12.

UNDO

Reverses all changes since last
function key press

CR

or

UNMARK

Cancels MARK directive without reversing changes

13. On line help:

A. FSE creates local file "FSEHELP"

B.

HELP

Splits screen with "FSEHELP" on bottom

C.

EDIT

To revert


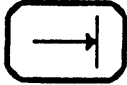
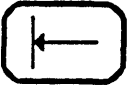






14. Create "MULTI-RECORD/FILE" files:

A. (EOR) in column number 1

B. (EOF) in column number 1















C. Simply delete line to delete EOR or EOF. Use CATALOG command to see impact of (EOR) and (EOF).


OTHER TERMINAL DEPENDENT "KEYS"

1.  Backspaces a single space and erases previous character
2.  &  Default tabs: 1,7,72.
See SET directive to change the default settings.
3.   Clears screen.
 Repaints screen.
4.  Puts current line at screen top.
 Puts current line at screen bottom.
5.  Stops in-progress search for replacement on 721 terminals.

Use SET PROMPT 2 and GET STATUS to see your current terminal key definitions.

OTHER 721 "PROGRAMMABLE" FUNCTION KEYS

1.  Inserts 22 blank lines in front of cursor.
2.   Deletes all blank lines until first non-blank line.
3.  Put cursor at first line in a file.
4.   Put cursor at last line in a file.
5.  Prompts for text to locate.
  Go to next occurrence of "LOCATE" text.
6.  Default screen width.
 
On other 80 column terminals, use the SET VIEW OFFSET directive.
Example: SVO 80.
7.  Moves line cursor is at to screen middle.
8.  Move cursor to end of current line.

9. **SPLIT** Splits line. The character which the cursor is on goes to beginning of next line.
10. **JOIN** Takes line below cursor and adds it to line cursor is at.
11. **ONECOPY** Same as  **COPY** except marks are cancelled after the copy.
12. **PARA**
- A. Reformats lines you marked with **MARK** to conform with margins set via SET WORD FILE directive.
- B. Like the .FILL directive.
- C. If no lines marked, reformats entire paragraph.
13. **CENTER**
- A. Centers line between first and last columns as set by SET WORD FILL.
- B. Like the .CENTER directive.
14. Function key 16 is currently undefined. Feel free to define Function Keys to meet your needs (see SET KEY directive).

FSE ALTER DIRECTIVE

ALTER : Enables user to change lines

1. Without parameters:

Can enter "MODIFY CHARACTER" and text to change "CURRENT LINE." Like "XEDIT" MODIFY.

2. Format: (parameters)

END direction string range QUIET

3. END : Adds string at EOL.

Ex: A E/ABC/ Adds ABC at End of Line.

4. Direction : A "COMMON PARAMETER"

Specifies direction and number times directive is to be executed

Ex: A P 2/#/ Alters (deletes) previous two occurrences.

5. String : Specifies text or modification character to change line(s).
See example above. "COMMON PARAMETER"

6. Range : Another "COMMON PARAMETER" specifies lines to be altered.

Ex: A A/^ #/ Inserts one blank before each line.

7. QUIET : Do not display results of ALTER

Ex: AAQ/^ #/ Same as above, but no veto option.

FSE BACK AND COPY DIRECTIVES

BACK: Returns user to a file you already edited during the current editing session.

FSE remembers last seven files edited. Pressing the BACK function key or entering BACK directive can return user to all seven. See also FSE's FSE directive.

COPY: Copy lines from one location to another, either within a file or between two files.

FORMAT,PARAMETERS : Range (F1) IO line (F2) QUIET

Ex: C 10 50 TO 20 (Y) (FILEA)

Copies lines 10 through 50 from current file to file FILEA at line 20.

FSE DELETE DIRECTIVE

DELETE: Deletes one or more lines.

1. Without parameters : The current line is deleted.
2. Format, parameters:

Range BLANK WORD IN tab QUIET

3. Range:

Ex: D 30 33
D :I

Delete current line and insert text with prompting.

4. BLANK: Deletes all blank lines from "current" to first non-blank line.
5. WORD: Deletes words, blanks or individual special characters within a line without deleting the whole line.
6. IN tab : Deletes a specific "TAB" field.

Ex: D IN 1

and TAB FIELD

COLUMNS (default)

| | |
|---|--------|
| 1 | 1-4 |
| 2 | 5-19 |
| 3 | 20-39 |
| 4 | 40-EOL |

Then: Columns 1-4 deleted.

Set Tab 1, 10, 20, 30, 40

1-9, 10-19, 20-29, 30-39, 40-EOL

①

②

③

④

⑤

FSE's FSE DIRECTIVE

ESE: Specifies a different file to edit.

1. Parameter format:

file charset GET READ SPLIT

2. File : Filename required

3. Charset : DISPLAY is default

4. GET:

Ex; EG ALPHA

5. READ: Uses local or attached direct access files. No GET or ATTACH is done.

6. SPLIT :

A. SET specifies how many lines in lower half of screen.

B. Can have only two parts on a split and the new file will always be in the lower half.

FSE GET AND HELP DIRECTIVE

GET: Either displays status of the files you are editing and activate function key definitions or lists the column numbers (with an on-screen ruler).

PARAMETER FORMAT : STATUS ALIGN

One or the other parameter is required.


Ex: GS to see how function keys can be defined using the SET KEY directive.

GA to place a column ruler on the "current" line. Use the SET SCREEN directive to remove ruler.

HELP: Displays the FSE "HELP" file.

1. Without parameters : Cursor positioned at first line of "HELP" file. Screen is split.
2. PARAMETER FORMAT : Directive
3. Directive : Specifies a legal FSE directive.

Ex: HREPLACE

4. Press  or ~~BACK~~ to exit.

*Set Screen
to remove*

FSE INSERT DIRECTIVE

INSERT : Inserts text

1. Without parameters:

A. SCREEN:

<--- Insert what?

The user inserts one line only in response to the above prompt.
Insertion after "current line."

B. LINE :

```
?? |  
32? A B C D  
33? X Y Z  
34? <cr>  
??
```

2. PARAMETER FORMAT:

line# PREVIOUS string BLANK WORD

3. Line:

Ex: 16 \ABC/

Inserts text \ABC after the sixth line.

Note: No matter what user does with "line", user can insert one line only in screen mode, except for BLANK and WORD.

4. PREVIOUS : User wants insertion before rather than after.

5. BLANK: In screen mode, inserts blanks after current line. Screen all blanks except for first and last line.

6. WORD: Inserts 3Ø blank characters at current cursor position PREVIOUS WORD gives 3Ø blanks before current character.

LOCATE FSE DIRECTIVE

LOCATE : Locates a specified character string.

1. Without parameters : The last character string specified is used. Otherwise user is prompted for one:

2. PARAMETER FORMAT:

WORD direction string range IN tab UPPER QUIET

3. WORD : Looks for a "word" as opposed to a "string." "Word" = a string contained in alpha-numeric (non) characters or blanks.

In line mode, positions to line; in screen mode, positions to screen.

4. Direction : A "COMMON PARAMETER". Indicates number of lines, not number of strings, located.

Ex: L N 2/ABC/

5. String:

Ex: L/The/./night./

6. Range:

Ex: L NEXT 2/The/FIRST

7. IN tab :

Ex: L A/Abc/IN

and TAB columns = 1, 5, 20 and 40.

Then: Locate all occurrences of 'Abc' in column numbers 20-39.

8. UPPER :

Ex: L U/ThE/

Will find: THE, THe, The, ThE, tHE, tHe, thE, the

FSE MOVE DIRECTIVE

MOVE: Moves text from one place to another within a file or between two files.

1. Without parameters : Nothing happens
2. PARAMETER FORMAT:

range (file1) IO line (file2) QUIET

Ex:

M 20 50 T LAST(FILEY)

Adds lines 20 through 50 from current file to end of file FILEY.
FILEY will be created if not yet local.

Ex:

SET X
MMTOX

Moves marked text to the line number previously saved in X.

FSE PRINT/POSITION DIRECTIVE

PRINT : Either prints (displays) a range of lines and moves pointer (LINE MODE) or just positions cursor (SCREEN MODE).

1. Without parameters : Current line printed or cursor goes to line beginning

2. PARAMETER FORMAT :

range QUIET

Ex:

SCREEN EDITING:

PL
PE
P1
PN3
PR3
P1Ø 15

LINE EDITING:

PN3
PR3
P1Ø 15

Ex:

| | |
|----|---|
| SL | Leave SCREEN mode |
| PA | Print all lines (after enabling printer) |
| SS | Return to SCREEN mode (after disabling printer) |

FSE QUIT DIRECTIVE

QUIT : Exits FSE or an FSE procedure

1. Always check that files were made permanent (shown in FSE's exit status messages).

2. Parameter format :

REPLACE QUIET UNDO PROC/nos command

3. REPLACE : Cannot be used with UNDO.

4. PROC: Used only in FSE procs. Instructs FSE to stop processing current procedure without exiting FSE. Enter FSE, FSEPROC to see your default FSE procedure file from username LIBRARY and example FSE procedures. The first one, named STARTUP, is run each time you start up FSE. Change and replace it if you wish your own version under your own username.

5. /nos command:

Ex: QR/FSE A handy way to save what you are doing and return immediately to FSE.

6. UNDO: Cancels an entire FSE session.

Ex: QUNDO You cannot abbreviate UNDO here!

FSE REPLACE DIRECTIVE

REPLACE: Replaces one text string with another.

1. Without parameters : Replace last strings specified. Or, you are prompted for intended text.

2. PARAMETER FORMAT :

WORD direction str1 str2 range IN tab UPPER QUIET

3. WORD : Word = str1 contained within non-alphanumerics or blanks

Ex : R W/Topic/subject/

4. Direction : A "common parameter". Using a number greater than "one" with NEXT, REPEAT and PREVIOUS, specifies number of lines, not number of occurrences.

Ex: R N /abc/xyz/ or R N 1/abc/xyz/
R N2/abc/xyz/

Note: "Direction" is a part of "range's" definition.

5. IN tab : Replaces text appearing only in certain tab fields.

6. UPPER : Capitalization is ignored.

FSE SET DIRECTIVE

SET: Sets various FSE and file parameters.

1. Without parameters : Not allowed.
2. PARAMETER FORMAT: keyword
3. ANNOUNCE/string/ : Enables user to display "string", usually from within an FSE procedure.
4. CHAR character : Sets "soft tab" character. GET STATUS displays it. \=def.
5. EILENAM file:
6. HEADER value :

YES=

NO= - Abbreviates Header line

7. INCREMENT number : Line increment
8. KEY number SHIFT string LABEL string : Redefines a programmable function key to execute the directives specified via "string."

Ex: S K1 S / / L /
 Legal Label to be put
 FSE on the key.
 directives




Ex:

SK6S\QRO\FSE\L\QREP/

9. LINE: Useful for printing file ("??" appears as prompt)
10. SCREEN: Useful to repaint screen and **must** be first command after FSE screen mode recovery to resume where you left off.

FSE SET KEYWORDS (CONT)

11. MARK range WORD: Sets 1 or 2 temporary markers to be used with another directive.

Ex: SM = 
SMW =  

Ex: SM 5 10
CMT 50

12. NUMBER value: Instructs FSE to treat file as sequenced or unsequenced.

A. AUTO: Treat as if unsequenced. In line mode, numbers are general, but not part of file.

B. BASIC: Treat as if sequential and sequential lines.

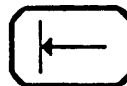
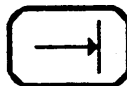
C. EORTAN: Like BASIC, except extra space not generated. Not recommended.

D. NONE:

13. PROMPT value: SP2 recommended.

14. TAB col1, .., col20: When in screen mode, hard tab in column number 1 is also used, allowing use of

15. UNDO value: YES or NO



FSE SET KEYWORDS (CONT)

16. VIEW value : Enables user to change screen format.

A. COLUMN num : Specifies number of columns displayed.

B. EEDIT num : Specifies right column to be edited.

C. LINE num : Specifies number of lines displayed.

D. OFFSET num : Changes leftmost column displayed. Default is SV01.
SV081 is useful on 80 column terminals.

E. SPLIT num :

F. WARN num :

17. WORD value :

A. CHAR char : Defines character attribute as alphanumeric or
punctuator. It reverses current setting. See Reference
Manual for commands this affects.

B. FILL margin1, margin2, margin3 : Sets margins other than defaults (1,
65,5) for use with CENTER and FILL.

18. X line WORD

Y line WORD

Z line WORD

Sets X, Y, or Z value to current or specified line (or word)
value. Like  "MARK" except permanent.

Ex:

a. Put cursor on character.

b. SET X W

c. Put cursor on second character.

d. SET Y W

e. Put cursor where you want string to go.

f. COPY X Y

FSE TEACH/UNDO DIRECTIVES

TEACH: Provides a practice file upon which you may try some FSE operations. If user uses CDC721, a tutorial is also provided.

EDIT

to return.

Ex: FSE, NOTHING./TEACH

NOTE: To set up teach files for terminals other than the 721, establish a file named Vterminal (terminal same as SCREEN) under User name LIBRARY.

UNDO: Removes all "marks" and reverses all changes to previous <cr> or function key.

FSE VIEW DIRECTIVE

VIEW : Allows viewing a group of lines. ESPECIALLY USEFUL IN LINE MODE.

1. Without parameters:

LINE : Displays current, 4 preceeding, and 4 succeeding lines

SCREEN : Centers current line vertically on screen.

2. PARAMETER FORMAT

line direction SCREEN HOME

3. Line : "Common parameter." specifies which line(s) you want to view.

4. Direction : "Common parameter". Part of "line." Specifies the direction you want to view. NEXT and PREVIOUS only!

5. SCREEN: With screen editing only. Enables user to view next or previous screen.

Ex: VNS

Same as  key.

FSE DOT DIRECTIVE

. (dot) : Performs word processing functions.

1. Format : .parameter

2. CENTER: Centers current line horizontally between preset margins.

Ex: SET WORD FILL 5 50 10
CENTER

This centers current line within columns 5 and 50.

3. FILL :

Ex: S W F 5 50 10
FILL or **PARA**

Adjusts entire paragraph to margins 5 and 50, plus indents first line to 10.

4. DELETE: Deletes current character and closes up line. Will not take characters from next line.

5. END : Moves cursor to EOL. Same as **ENDLIN**

6. INSERT : Inserts 1 blank before cursor.

7. INSERT/string : Inserts "string" before cursor.

8. JJOIN : Takes line below cursor and adds it to current line. Same as

Ex: J

JOIN

9. SPLIT : Splits line. The character cursor is at goes to beginning of next line. Same as

SPLIT

Ex: S

10. POS number : Moves cursor to column number of current line.

Ex: .P7 Moves cursor to column 7.

EXERCISE 3.1

FULL SCREEN EDITOR

1. Enter FSE using any file name and type "teach" from the home position. This will put you into a lesson that will guide you through an FSE session using the various keys and commands.

It should take approximately 45 minutes to an hour to complete. If you finish early, continue on.

2. Define a function key, such as shift-F6, to be the directive QRQ/FSE. Use GET STATUS to see current Function Key definitions. Caution - format is SET KEY n SHIFTED etc.
3. Define a Function Key to use SET VIEW OFFSET to switch between column 1 and column 81.
4. Save these definitions and the SP2 directive in the FSE STARTUP procedure in your own FSEPROC file. Restart FSE to try it out. Remember to save the customized file FSEPROC under your username.
5. Try out several FSEPROC procedures; such as CUT, PASTE, UPPER, and LOWER.

LESSON FOUR

LOCAL FILE COMMANDS

LESSON FOUR

LOCAL FILE COMMANDS

Lesson Preview:

This lesson acquaints the student with the NOS local file commands and demonstrates their use.

Objectives:

After completing this lesson, the student will be able to:

- List information about a local file before and after specific events.
- Release space assigned to local files.
- Use the file positioning statements, REWIND, SKIPF, SKIPFB, SKIPR, BKSP, and SKIPEI to get to a desired position in a file.
- Use the file copy utilities, COPY and COPY (BR, BF, CR, CF, EI, and SBF) to copy portions or all of one file to another.
- Use the VERIFY command and the verify parameter on the COPY command to certify the results of copying.

Projects: Exercise 4.1 (Local File Commands)

References:

- NOS Version 1 Reference Manual, Volumes 2 and 3, Local File Command.

LOCAL FILE POSITIONING

This lesson concerns file structure, positions, and content of local files.

File content is unimportant when using some commands, but is important with others. These differences will be examined in this lesson.

A file, as defined in Lesson 2, is a collection of information and can be manipulated by the local file commands. Files are more efficiently processed on disk because of the higher transfer rate of disk over tape. Where possible, tape files should be considered to be archival and not be physically positioned, but rather, a disk copy can be used for sorts and searches.

Beginning-of-information (BOI), end-of-record (EOR), end-of-file (EOF) and end-of-information (EOI) are the lowest data organizational level at which the commands and compiled programs work.

When is a file at BOI? Always, in the case of a empty file (only an entry in the job's File Name Table). After a REWIND . . . and in many cases, because of a rewind parameter of a command, such as, "CATALOG,EFFILE,R."

When is a file at EOI? The empty file is one example. Another case is when a file has specifically been positioned to EOI by means of COPYEI, SKIPEI, or as the result of some move on the file's current position pointer.

Any other position on a file requires programmer knowledge of the file structure. It is crucial therefore, to understand the effect of each command on the file's position.

LOCAL FILE COMMANDS

The ENQUIRE,F command lists names and attributes (length, position, etc.) of local files.

The CLEAR, UNLOAD, and RETURN commands allow clearing of file space.

The CATALOG,SCOPY and TDUMP commands, give a report about what is in a file and how it is organized. This is useful when a detailed analysis of the file is required and no other documentation about the file exists.

The various, skip, and backspace (BKSP) commands can be used once the file structure and content are known, to position the file exactly to where a read or a write are to be performed.

The copy commands, duplicates logical records and files or entire physical files, depending on the particular command used and the parameters selected. File position is affected by the copies.

The VERIFY command provides a means to guarantee that information was duplicated accurately.

You have already been introduced to several local file commands (CATALOG, CLEAR, and ENQUIRE) in Lesson 2 - Permanent File Concepts and Commands and to the local/permanent file editor (FSE) in Lesson 3 - Full Screen Editor.

MASS STORAGE FILE STRUCTURE

When NOS stores a file, it automatically converts the file to a structure that will conform to the physical characteristics of the storage medium. The logical file and record marks are converted to physical BOI, EOR, EOF, and EOI indicators. The basis of all physical file structure is the Physical Record Unit (PRU), the minimum amount of data that can be read or written in a single read/write access.

On a disk, tracks exist for recording information. A sector is the part of a track containing enough space to record 64 words or one PRU.

PRU size - 64 CM words (up to 640 characters of data)

BOI - PRU 1 or random address 1 of the disk

EOR - PRU of less than 64 words with a link to the next PRU

EOF - Zero-length PRU with special link to next PRU

EOI - Zero-length PRU with no forward link

Zero-length PRU - No data in the sector

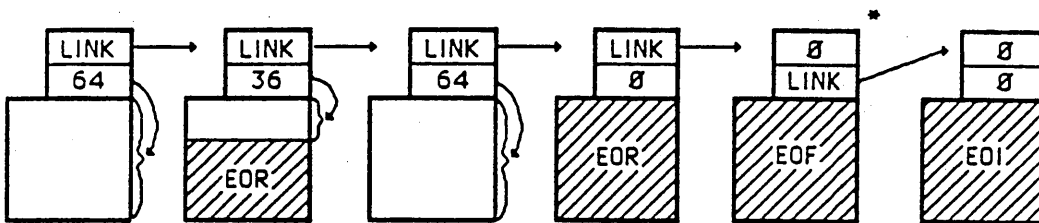
Short PRU - Less than 64 words of data in the sector

Full PRU - 64 words of data in the sector

Logical Record - Zero or more full PRU's terminated by a Short or Zero-length PRU.

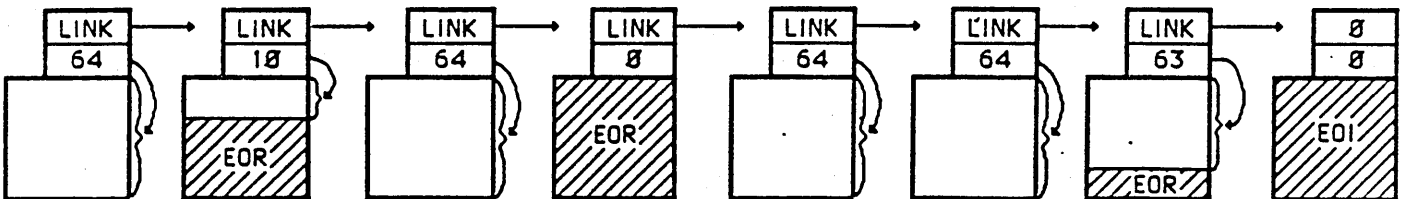
EXAMPLES OF MASS STORAGE FILE STRUCTURE

FILE 1 - TWO RECORD FILE

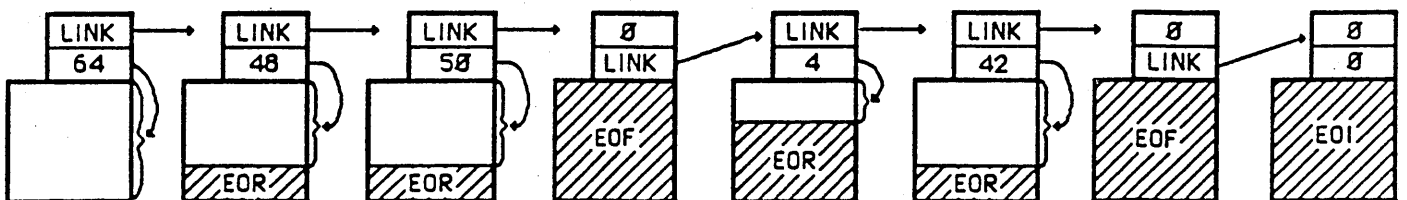


* Note that an EOF is not required as the last PRU of a file.

FILE 2 - THREE RECORD FILE



FILE 3 - MULTI-FILE FILE



RETURN COMMAND

- Release local files assigned to a job.
- Returns tape unit to system if RESOURCE limits are involved.

FORMAT:

RETURN,lfn1,lfn2,...,lfn_n.

RETURN,*,lfn1,lfn2.

RETURN,*.

Examples:

/RETURN,TO,THE,FUTURE.

/ENQUIRE,F

/RETURN,*,BUT,THESE.

/ENQUIRE,F

Files with No-Auto-Drop (NAD) and files BUT and THESE remain. Files INPUT and OUTPUT remain only because IAF reassigned them.

UNLOAD COMMAND

- Same as RETURN for local files.
- Unloads tape but tape unit resource remains with job.

FORMAT:

UNLOAD,lfn1,lfn2,...,lfn_n.

UNLOAD,*,lfn1,lfn2.

UNLOAD,*.

Examples:

/UNLOAD,THESE,FILES.

/UNLOAD,*,BUT,NOT,THESE.

TDUMP COMMAND

- Lists a local file in octal and/or alphanumeric format

FORMAT:

TDUMP,params.

Where params =

I = Ifn1 Local file to be listed
L = Ifn2 Local file to contain output (default=OUTPUT)
A Alphanumeric dump only
R = rcount Number of records in the file to be listed
F = fcount Number of files in a multi-file to be listed
N = lines Number of lines to be listed
NR No rewind; do not rewind Ifn1 before the TDUMP

Examples:

/TDUMP,I=TAPE1.
alphanumeric.

Dumps in both octal and

/TDUMP,I=BINARY,L=LIST,R=2,NR.

/GET,BINS,TEXT
/TDUMP,I=BINS,L=LIST2.
/TDUMP,I=TEXT,A.

REWIND COMMAND

- Rewinds files (local files, including tape files)
- Positions files at Beginning-Of-Information (BOI)

FORMAT:

REWIND,file1,file2,file3,...,file_n.

REWIND,*.

REWIND*,file1,file2,..., file_n.

Examples:

| | |
|----------------------|--------------------------------------|
| /ENQUIRE,F | To see file positions. |
| /REWIND,DOG,CAT,PAT. | Rewind these three files. |
| /ENQUIRE,F | |
| /REWIND*,NOTME,ORME. | Rewind all files but NOTME and ORME. |
| /ENQUIRE,F | |
| /REWIND,*. | Rewind all files. |
| /ENQUIRE,F | |

SKIP COMMANDS

- **SKIPEI, lfn.**
 - Positions file at End-Of-Information (EOI).
- **SKIPF, lfn, n.**
 - Skip forward n number of files (EOFs).
- **SKIPFB, lfn, n.**
 - Skip backwards n number of files (EOFs).
- **SKIPR, lfn, n.**
 - Skip forward n number of logical records (EORs). End of files (EOFs) are treated as EORs.
- **SKIP, label.**
 - Not a file positioning command.

- **Examples:**

| | |
|----------------|-----------------------------------|
| /SKIPEI, FILE2 | |
| /COPY, F2 | No output |
| /SKIPFB, FILE2 | |
| /COPY, FILE2 | Copy last file (or EOF) |
| /REWIND, FILE2 | |
| /SKIPR, FILE2 | |
| /COPY, FILE2 | Copy all by first record (or EOF) |
| /REWIND, * | |

BKSP COMMAND

- Backspace specified number of logical records (EORs) on a local file.
- Will be stopped by Beginning or Information.

FORMAT:

BKSP,lfn,n.

Example:

/BKSP,ZIGGY,3.

If ZIGGY has three records and an EOF was positioned at EOI, ZIGGY will be positioned between first and second record since the EOF is counted as one record.

COPY COMMAND

- Copies data from one local file to another (files not rewound).
- Will stop at double consecutive EOF.
- See also COPYEI command.

FORMAT:

COPY,lfn1,lfn2,v.

Examples:

/COPY,CAT,RAT,V. Copies CAT to RAT with verify.
/REWIND,*.
/COPY,MASTER,OLDMSTR.
/REWIND,*.
/VERIFY,MASTER,OLDMSTR. Will not compare successfully if MASTER had
two consecutive imbedded EOFs.

SCOPY COMMAND

- Useful to see EOR/EOF structure along with data in a text/data file.

Example:

/ATTACH,HELPLIB/UN=LIBRARY.
/SCOPY,HELPLIB. Note that the interactive LIST command
actually is performed by SCOPY.

FCOPY COMMAND

- Generalized file copy, with conversion, routine.
- See NOS Reference Set, Volume 3, for parameters and examples of converting a NOS IAF ASCII file to 8-bit ASCII for printing on Printer Support Utility (PSU) line printers.

COPYBF/COPYCF COMMAND

- Copies file(s) of a multifile local file to another local file.
- On tapes, copies in coded (COPYCF) or Binary (COPYBF) mode.

FORMAT:

/COPYBF,lfn1,lfn2,n.

Where n=number of files to copy

Example:

/COPYBF,BIGFILE,SMALLER,6. Copies 6 files (or to EOI).

COPYBR COMMAND

- Copies specified number of records from one local file to another

FORMAT:

COPYBR,lfn1,lfn2,n.

Example:

/REWIND,BIGREC.

/COPYBR,BIGREC,MISCREC,10.

Note that EOFs are treated as EORs.

COPYCR COMMAND

- Has first and last column options.

FORMAT:

COPYCR,lfn1,lfn2,n,fc,lc.

Example:

/DAYFILE,DAYF.

/REWIND,DAYF.

/COPYCR,DAYF,COMMAND,99,11,80.

/FSE,COMMAND.

An easy way to build NOS command files for Remote Batch jobs and procedure files.

COPYEI COMMAND

- Copies one file to another until End-Of-Information (EOI) is reached.
- Recommended for random files (last record is type OPLD in CATALOG or ITEMIZE listing).

FORMAT:

COPYEI,lfn1,lfn2.

Examples:

| | |
|--------------------------|--------------------------------|
| /COPYEI,ME,YOU. | Copies file ME to file YOU. |
| /WRITEF(TEXT,2). | Writes two EOFs. |
| /WRITER(TEST). | Write one EOR. |
| /REWIND,*. | |
| /COPY(TEST,NOEOR). | Stops at two consecutive EOFs. |
| /REWIND,*. | |
| /COPYBF(TEST,LARGER,99). | |
| /REWIND,*. | |
| /COPYEI(TEST,TESTEI). | |
| /REWIND,*. | |
| /ENQUIRE,F. | And compare sizes. |

COPYSBF COMMAND

- Copy shifted binary file.
- Printer carriage "tape" control characters are added to lines of the file.
- A top-of-form will be inserted before each logical record (EOR).
- Used to prepare local text files to be routed to printer or printed at terminal after the EFFECT,OFF command has been issued.
- Very important if first character of line image otherwise would contain data. For example, one (1) will cause top-of-form; zero (0) will cause double spacing.

FORMAT:

COPYSBF,lfn1,lfn2,n.

Example:

/COPYSBF,TEXT,PRINT,2. Copies two files from TEXT to PRINT.
/ROUTE,PRINT,TID=C,DC=PR.

/EFFECT(OFF).
/COPYSBF,LISTING. Allows COPYSBF to control paging/spacing.

PACK COMMAND

- Removes all end of record (EOR) and end of file (EOF) marks from the specified local file.
- Copies file as one record to another file.
- Caution; do not PACK procedure and binary files.

FORMAT:

PACK,file1,file2,x.

Where:

file1 = File to be packed

file2 = File on which to write packed data

x = Any alphanumeric character string, 1-7 characters, which indicated file1 should not be rewound before pack

Example:

/LENGTH,ITUP.

/PACK,ITUP.

/LENGTH,ITUP.

/PACK,TEST,ONEREC.

Packs ITUP onto itself.

The three record file from COPYEI example is reduced to one record on file ONEREC.

/ENQUIRE,F.

WRITE COMMANDS

WRITEF,lfn,n.

- Writes n number of filemarks (EOFs) on a local file.

WRITER,lfn,n.

- Writes n number of empty records (EORs) on a local file.

WRITEN,lfn1,lfn2.

- Removes sequence numbers from lines in a sequenced text file.

Example:

```
/BASIC
/NEW,TEST
/TEXT
00100 enter lines
00200 of text
00300 -BREAK-
/WRITEN,TEST,TESTOUT
/LIST
/LIST,F=TESTOUT
/BATCH
```

VERIFY COMMAND

- Compares data on one local file to another.
- See NOS Reference Set, Volume 3, for compatible formats on tape files.

FORMAT:

VERIFY,lfn1,lfn2,params.

Example:

/GET/PEAS=FROZEN,CARROTS=FROZEN. Assume files on FROZEN contain different data.

/VERIFY,PEAS,CARROTS,N,R. Verifies all records of files PEAS and CARROTS. Files rewound before and after.

/SKIPF,PEAS.

/VERIFY,PEAS,CARROTS. Verifies records in "current" file only.
Will this verify successfully?

/SKIPEI,PEAS.

/SKIPEI,CARROTS.

/VERIFY,PEAS,CARROTS,N. Will this verify successfully? What did we forget to do here?

EXERCISE 4.1

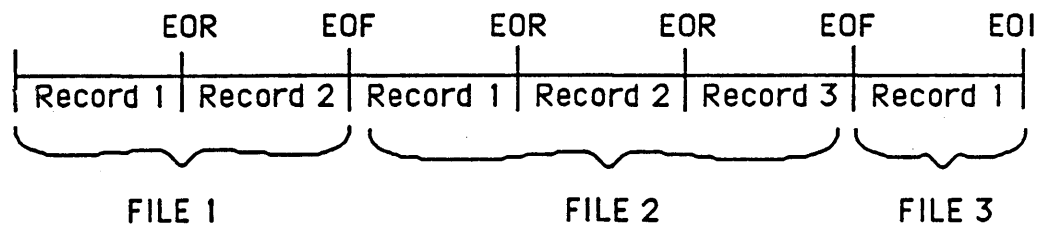
LOCAL FILE COMMANDS

1. Get FILE _____, under username _____. This file contains a FORTRAN source program. CATALOG the file to see the file structure.
2. Compile the program,

FTN5,I=____,B=BINS,L=LIST.

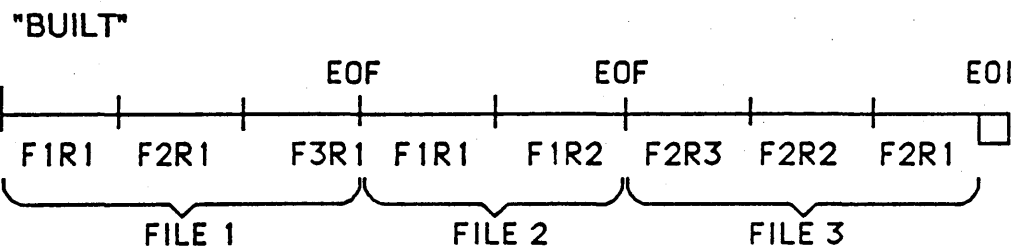
CATALOG and TDUMP on the binaries created (BINS). Is there any difference between the catalog of the source (done above) and the binaries (in BINS)? If so, what is/are the difference(s)?

3. Use FSE to create a file "GIVEN" which has the following file structure:



Use CATALOG and SCOPY to see this structure.

4. Use the copy and positioning utilities to create a file "BUILT" with the following structures:



Where FmRn is record n of file m in the file "GIVEN" of exercise step 3 above.

Hint: Disassemble file and build new one.

EXERCISE 4.1 (CONT)

5. Verify the records in the source file of step three against corresponding records in File 3 of step four's "BUILT" file. Is the result what you expected?
6. Combine "BUILT" file into one logical record.
7. Unload all local files except the files of step three through six. Check that the FORTRAN source and binary files were returned.
8. Use the COPYCR example to inspect, edit, and save your command sequence of this exercise as the file "JOBDECK."

LESSON FIVE

JOB CONTROL LANGUAGE

LESSON FIVE

JOB CONTROL LANGUAGE

Lesson Preview:

In this lesson, the student will learn how to structure a job by the use of JOB Control statements. This includes the identification of a job and the specification of needed resources by means of the following statements:

| | | |
|---------|---------|--------|
| CHARGE | JOBCARD | QGET |
| COMMENT | LIMITS | SETJOB |
| DAYFILE | NOTE | SUBMIT |
| DROP | PASSWOR | USER |
| ENTER | REDO | |
| HELP | RENAME | |
| HELPME | ROUTE | |

Objectives:

After completing this lesson, the student will be able to:

- Use the JOBCARD, USER, and CHARGE statements to gain access to the system.
- Use the LIMITS statement to receive a report of allowable resources for an account number.
- Use the PASSWOR statement to change the password on the USER statement.
- Use the DAYFILE, COMMENT, and * statements.
- Use the interactive help procedure to obtain information on the NOS commands.
- Define the NOS queues, send jobs to the queues, and manipulate queued files.

Projects: Exercise 5.1 (Job Control Commands)

References:

- NOS Reference Set, Volume 2.
- NOS Reference Set, Volume 3, Section 7.

COMMAND SYNTAX

1. Separators:

+ - " / , (

2. Terminators:

.)

3. Embedded blanks are ignored.

4. Filenames are 1-7 alphanumeric characters.

5. Filenames SHOULD begin with a letter.

6. NOS expects all commands to be NORMAL (6-bit) upper-case characters.

EXAMPLE:

| | |
|------------------|---|
| /FSE,JOBDECK,NG. | is same as |
| /GET,JOBDECK/NA. | |
| /CSET,NORMAL | |
| /FSE,JOBDECK. | an editing session involving NOS command files. |

COMMENT COMMAND

- Places comment in system dayfile and job dayfile.

FORMAT:

COMMENT.comments entered here

*comments entered here

Examples:

/COMMENT. THERE ARE COMMENTS WITH ONE FORMAT

/*THIS IS ANOTHER WAY TO WRITE COMMENTS

DAYFILE COMMAND

- Dayfile is a chronological activity log created during job execution which contains job history and accounting information.
- A terminal session's dayfile is discarded at LOGOUT.
- A remote batch job's dayfile is appended to the job's OUTPUT file for printing.
- System writes job's dayfile to OUTPUT or file specified on statement.

FORMAT:

DAYFILE.

DAYFILE,L=lf,FR=string,OP=ip,PD=pd,PL=pl,I=infile.

- or -

DAYFILE,lf,string,op,pd,pl,infile.

Where:

lf Local file on which to write dayfile

string String to search for in dayfile

op Search option

T Time field

M Message field (default)

I Incremental dayfile listing (default)

F Full dayfile listing

pd Print density

pl Page length

infile File containing previous dayfile; default is current dayfile

DAYFILE EXAMPLES

Examples:

| | |
|---------------------------------|--|
| /DAYFILE. | Incremental dayfile listed (from last dayfile). |
| /DAYFILE,OP=F,L=LIST. | Full dayfile written to file LIST. |
| /DAYFILE,L=DFILE,FR=ERROR,OP=M. | File DFILE will begin with last message beginning with "ERROR" or full dayfile if not found. |
| /*DAYFILE START | |
| /DAYFILE,FR=* | List dayfile beginning with the last message beginning with an asterisk. |

ENTER COMMAND

- Ability to enter series of commands on one line.
- Basically allows a one line procedure command list.
- The character after the ENTER commands terminator will be the command separator (any separator which is not used in the commands themselves.)

FORMAT:

ENTER./command1/command2/.../commandn

Example:

```
/ENTER./GET,PRINT./COPYSBF,PRINT,PRINTME./ROUTE,PRINTME,DC=PR.  
/FTN PROC  
/ENTER./NOEXIT./LGO./DMP./DMD(0,3000)  
/GET,COMPILE,SOURCE.  
/FSE,SOURCE
```

Under FSE:

```
SK6S\QRQ/ENTER./COMPILE./FSE.\L\QRPROC\
```

Shift-F6.

EXPLAIN/HELP COMMANDS

- Generates a menu of online help features
- Prompts for a selection

FORMAT:

HELP or EXPLAIN

Example:

| | |
|-----------------|---|
| /SCREEN,PCCONN. | Personal Computer using the CDC Connect Software. |
| /HELP. | |
| /EXPLAIN | |

HELPME COMMAND

- Gives brief description of command
- Lists command format and all valid parameters
- HELPME is an interactive procedure which prompts for parameters.

FORMAT:

HELPME,COMMAND

--> Press <CR> any time during dialogue to get more information about command parameters

Example:

```
/LINE  
/HELPME,COPY  
/SCREEN,72230T
```

- Format of command with all parameters
- Brief description of command

Copy from this file? <lf1> <cr> or cntl-T to quit

Copy to this file? <lf2> <cr>

<cr> for all other parameters will default.

```
/HELPME,SCREEN
```

REDO COMMAND

- Allows modification and reentry of a previously entered command.
- Only modify parts of the command; don't have to reenter entire line.
- Editing conventions are the same as FSE's ALTER directive (See NOS REDO in Reference Set, Volume 3).
- Useful to see what system saw when "INVALID COMMAND" is issued. Just type R.

FORMAT:

REDO,string/GO

Where:

| | |
|--------|---|
| string | Initial characters of command to be modified and reentered; up to 10 characters. Use \$ delimiters if non-alphanumeric characters are required. |
| GO | Reenter command without modifications |

Examples:

| | |
|-------------------------------|--|
| /REDO,DAYFILE/GO. | Resubmits last command beginning with DAY. |
| /CATLIST. | |
| /R/GO | Resubmits last command (CATLIST). |
| /ATTACH,OPL=NOSOPL/UN=LIBRARX | |
| /R | Allows easy correction of LIBRARX to LIBRARY and resubmission. |

LIMITS COMMAND

- Lists user's validation limits.
- Shows user index - important if permanent files are reloaded by the installation.

FORMAT:

LIMITS.

LIMITS,L=lfm. (default is OUTPUT)

NOTE COMMAND

- Creates file containing, lines of data on same line as command.
- Format is like ENTER (and BLOCK) commands.
- See also BLOCK (letters) and Security Header (SEC HDR) if you have listing format needs.

FORMAT:

NOTE,Ifn,NR./LINE1/LINE2

Where:

Ifn Local file name (default is OUTPUT)

NR No rewind of Ifn before or after command execution

Example:

/NOTE,,NR./WHAT A GROOVY JOB!

/NOTE/ANY PRECEEDING OUTPUT IS EVICTED

PASSWOR COMMAND

- Allows user to change password.
- Changes batch or interactive password based on origin of job in which it appears.

FORMAT:

PASSWOR,oldpassword,newpassword.

Example:

/PASSWOR,WAS,ISNOW.

/PASSWOR.

System will read password values
from INPUT.

? oldvalue

? newvalue

RENAME COMMAND

- Changes name of local file.
- Note the nfn=ofn format.

FORMAT:

RENAME,nfn₁=ofn₁,...,nfn_n=ofn_n.

Example:

```
/GET,WHITE,YELLOW  
/RENAME,RED=WHITE,BLACK=YELLOW.  
/ENQUIRE,F
```

NOS QUEUE RELATED COMMANDS

- ROUTE
- JOB
- USER/CHARGE
- SUBMIT
- SETJOB
- ENQUIRE, UJN (Lesson 2)
- QGET
- DROP

ROUTE COMMAND

- Queues a local file to any queue.
- File will no longer be local.
- If "SECURITY VIOLATION" results, check dayfile and either JOBCARD or USER statement is in error. Continued security errors can result in your user name being invalidated automatically.

FORMAT:

ROUTE,lfn,DC=dc,REP=rep,UJN=ujn,TID=C,UN=un,DO=lid.

Where:

- | | |
|---------------|---|
| dc = IN | Must have correct JOB CARD or it remains local. |
| dc = WT | File lfn does not execute, instead it goes to terminal wait queue, must retrieve it with QGET. |
| dc = LP or PR | File lfn goes directly to be printed on a printer. |
| dc = NO | File lfn goes to input queue, no dayfile or OUTPUT file is generated unless overridden by a ROUTE or SETJOB statement in the job. |
| dc = TO | File lfn goes to Input queue, at end of job the OUTPUT queued file goes to terminal in wait queue, must retrieve it with QGET. |
| REP= rep | Repetitions of routed file (does not apply to files printed at IAF terminals). |
| UJN= ujn | User job name associated with routed file. |
| TID = C | Route OUTPUT file to BATCHIO (BIO) subsystem's printer at the central site (C). |
| UN = un | User to receive queued files. Default is current user. |
| DO = lid | Mainframe destination for queued output files. |

ROUTE EXAMPLES

Examples: Assume permanent file JOB contains:

MYJOB,T=100.
USER,un,pw,family.
ENQUIRE

/GET,JOB1=JOB,JOB2=JOB,JOB3=JOB.

/ROUTE,JOB1,DC=IN,UJN=TRY1.

JOB1 sent to be executed and output will return to user.

/QGET,UJN=TRY1.

Retrieves output of JOB1.

/ROUTE,TRY1,UN=MYBUDDY,DC=WT.

Either this user or user MYBUDDY can QGET (or DROP) TRY1.

/ROUTE,JOB2,DC=NO.

JOB2 runs and no output is returned. Perhaps data was saved in a file.

/ROUTE,JOB3,DC=WT.

JOB3 is queued directly as output in the terminal wait queue.

/FSE,LIST,A

/ROUTE,LIST,DC=PR,IC=ASCII,TID=C.

Print an upper/lower (6/12 bit) ASCII file on a 8-bit ASCII line printer. Appendix A in NOS Reference Set, Volume 3, contains NOS supported character sets.

JOB CARD

- First card of job deck
- Specifies user job name

FORMAT:

JOBCARD.

JOBCARD,px.

Where px = Job processing parameters (see Vol. 3, Section 7)

- 1-7 alphanumeric characters
- First character must be alphabetic
- Can specify job processing parameters, e.g., job priority

Example:

JOB1,T=100. Remember terminators!

JOB2,P5.

USER CARD

- Required card; must be after job card.
- Note that family name is specified last.

FORMAT:

USER,username,password,family.

Example:

USER,TOSUSER,KITTY,KMF.

CHARGE CARD

- Required if username is set up to use charge and project numbers and no default is defined for the user name.
- If required, NOS will prompt user with "CHARGE REQUIRED".
- Dynamic updating and enforcing of usage against charge and project limits is an option supported by NOS through a utility called PROFILE. Each charge number can be assigned a master username which is responsible for maintaining the charge and project number usage profiles.

FORMAT:

CHARGE,chargenumber,projectnumber.

Example:

CHARGE,CHARGE001,PROJECT005.

CHARGE,*.

Displays current charge and project numbers.

SUBMIT COMMAND

- Places a job file into the input queue as a separate job.
- Caution: Errors in file (e.g., different Batch passwords) could cause SECURITY errors.
- Allows reformatting of job file.
- Each record on the job file is a separate job.

FORMAT:

SUBMIT,lfn,q,NR,RB=un.

Where:

- lfn Local file name of file to be submitted to INPUT queue.
- q Disposition of job output after file lfn executes. Values can be BC, TO, or NO. Default is Local Batch at the central site (BC).
- NR No rewind before job submission.
- un A destination Remote Batch (or IAF) username.

SUBMIT EXAMPLES

Example:

```
/GET,SKEEZIX.  
/SUBMIT,SKEEZIX,TO.  
/ENQUIRE,UJN.
```

Where job file SKEEZIX file format is:

```
/JOB  
/NOSEQ  
MYJOB,T=100,P=4.  
/USER  
/CHARGE  
SETJOB (DC=TO)  
ENQUIRE,F.  
CATLIST.
```

NOTE: Slash here denotes SUBMIT directives.

Note: No values and
no period.

```
./EOR  
data record if required.  
/ENQUIRE,UJN.  
/QGET,UJN=SKEEZIX.
```

SETJOB COMMAND

- Change some of the current job's attributes.

FORMAT:

SETJOB,ujn,dc,op.

SETJOB,UJN=ujn,DC=dc,OP=op.

Where:

UJN Changes default UJN to newly specified ujn

DC Disposition for output upon job termination

TO Output goes to wait queue

NO No output

DF Default output processing

OP Job processing options for detached jobs

SU (default) Stays suspended until recovered or timeout

TJ System terminates job

Examples:

/SETJOB,JOBID1.

/SETJOB,JOBID2,DC=WT.

/SETJOB,JSN=AABX,OP=TJ.

/SETJOB(DC=TO).

QGET COMMAND

- Makes a queued file local to your terminal.

FORMAT:

QGET, jsn,q,ujn,lfn.

QGET,JSN=jsn,DC=q,UJN=ujn,FN=lfn.

Where:

JSN Job sequence name of queued file

UJN User jobname of queued file

DC Queue where desired file resides (user ENQUIRE,UJN to determine this)

WT (default)

PU

PL

PR

FN lfn Local filename to contain queued file

JSN Default if specified

UJN Default if specified

Example:

/ENQUIRE,UJN.

/QGET,AAAB.

/ENQUIRE,F.

/ENQUIRE,UJN.

/QGET,JSN=BAAD,UJN=SHEEP,DC=WT,FN=WOLF.

Note that the filename of the local file here will be AAAB.

DROP COMMAND

- Drop executing remote batch jobs.
- Drop queued remote batch INPUT/OUTPUT queued files.

FORMAT:

DROP,JSN=jsn,DC=q,UJN=ujn,OP=R.

- o r -

DROP,jsn,q,ujn,R.

Where:

jsn Job Sequence Number (JSN) of job to be dropped

q Disposition Code (DC) of job to be dropped (e.g., based on ENQUIRE, UJN)

ujn User Job Name (UJN) of job to be dropped

OP=R Drop specified executing jobs without Error EXIT processing

Examples:

/ENQUIRE,UJN.

/DROP,AAAB.

/ENQUIRE,UJN.

/DROP,,PR.

/ENQUIRE,UJN.

DROP,UJN=MYJOB,DC=WT.

/ENQUIRE,UJN.

Since JSN is specified, DC=ALL is assumed.

Drop all print for all JSN's.

Drop MYJOB out of terminal wait (for QGET) queue.

EXERCISE 5.1

JOB CONTROL COMMANDS

1. _____ Using FSE, create a job file which issues a dayfile to a local file (DAYFILE,L=FILE.) and uses both variations of the COMMENT command.
_____ Save the file.
2. _____ SUBMIT the job file created in #1 to the input queue and have output go to the wait queue (SUBMIT,lfn,TO.)
_____ Issue an ENQUIRE,F. The file you submitted will still appear as a local file.
3. _____ ROUTE the file created in #1 to the wait queue. (ROUTE,lfn,DC=wt)
_____ Issue the ENQUIRE,F command again. The file just routed to the wait queue is no longer a local file.
4. _____ Issue an ENQUIRE,UJN to display both the job sequence name (JSN) and the user job name (UJN) of your jobs in the queues.
_____ Issue a SETJOB command to change the user job name.
_____ ROUTE another job to the wait queue and issue an ENQUIRE,UJN. The last job routed will have the newly specified UJN.
5. _____ Issue a QGET command to make one of the files in the wait queue local to your terminal.
_____ Issue an ENQUIRE,F to see that the formerly queued file is now local.
_____ Issue an ENQUIRE, JSN to see that the file made local with the QGET command is no longer in the wait queue.

EXERCISE 5.1 (CONT)

6. _____ Drop all remaining jobs in the wait queue.
_____ Issue an ENQUIRE,JSN. Only one job will appear. It will have "executing" status - that is your terminal.
7. _____ Issue a DAYFILE command and have the output go to a local file. (DAYFILE,L=FILE1.)
_____ Enter: HELPME,RENAME. Enter the required parameters to rename the local file just created with the DAYFILE statement.
8. _____ Enter several commands on a single line to issue a DAYFILE, LIMITS,, and ENQUIRE (using the ENTER./ command).

LESSON SIX

FLOW CONTROL COMMANDS

AND

PROCEDURE FILES

LESSON SIX

FLOW CONTROL COMMANDS AND PROCEDURE FILES

Lesson Preview:

In this lesson, the student will learn how to structure a job by the use of flow control commands and how to write and use procedure files.

Objectives:

After completing this lesson, the student will be able to:

- State the rules of syntax governing formation of jobs using CYBER Control Language (CCL) now simply referred to as Flow Control Commands.
- Generate and save procedure files
- Call procedure files
- Generate and save a user prologue (UPROC)
- Alter execution of a procedure through flow control statements

Projects:

- Exercise 6.1 (Flow Control Commands/Procedure Files)
- Exercise 6.2 (Procedure HELP)
- Exercise 6.3 (Menu-driven Procedures)
- Exercise 6.4 (Interactive Procedures)

References:

- NOS Version 2 Reference Manual, Volume 3, Section 4

FLOW CONTROL COMMAND SYNTAX

- Comma or left parenthesis separates name and first parameter
- Commas separate consecutive parameters
- Period or right parenthesis terminates statement
- Right parenthesis ending an expression within a statement cannot also serve as the statement terminator
- Parenthesis can nest expressions within expressions; they do not imply multiplication
- Comments can follow statement terminator

A Flow Control command can be over 80 characters. It can extend over more than one line if each line to be continued contains no more than 80 characters and ends with a separator.

COMMAND SYNTAX EXAMPLES

1. Statement Expression Label
 Name String
 _____|_____|_____
 WHILE,R1.LT.R2,FINISH.

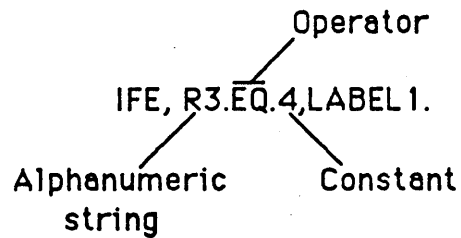
2. Statement Function Label
 Name String
 _____|_____|_____
 IFE,FILE(TEST1,LØ),LABEL1.

3. Statement Function
 Name "
 _____|_____|_____
 DISPLAY (FILE(TAPE1,BOI))

FLOW CONTROL COMMAND EXPRESSIONS

- Consist of operators and operands.
- Any character string that begins with a numeric cannot subsequently contain non-numeric characters, except for an optional B or D post-radix.
- Alphanumeric strings must begin with an alphabetic character.

Example:



EXPRESSION OPERATOR TYPES

- Used to relate operands in expressions.
- Three types of Operators
 - Arithmetic
 - Relational
 - Logical
- Three types of Operands
 - Numeric and string constants
 - Symbolic names
 - Functions

ARITHMETIC OPERATIONS

- The following are the arithmetic operators:

| OPERATOR | OPERATION |
|-----------|----------------|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ** | Exponentiation |
| Leading - | Negation |
| Leading + | Ignored |

Examples:

SET(R1=R2*2+1)

SET(R1=(R2+1)+(R3+1))

DISPLAY,5**3.

RELATIONAL OPERATORS

A relational operator produces a value of 1 if the relationship is true, and 0 if it is false. The following are the relational operators (either form may be used).

| OPERATOR | OPERATION |
|----------|--------------------------|
| = .EQ | Equal to |
| .NE | Not equal to |
| < .LT. | Less than |
| > .GT. | Greater than |
| .LE. | Less than or equal to |
| .GE | Greater than or equal to |

Examples:

```
DISPLAY,R1.  
IF,R1.NE.10,DO THIS.  
*R1 IS NOT EQUAL TO 10.  
ELSE, DO THIS.  
*R1 IS EQUAL TO 10.  
ENDIF, DO THIS.
```

LOGICAL OPERATORS

When an expression contains a logical operator, the system evaluates the full 60 bits of each operand and produces a 60 bit result. Each bit of the first operand is compared to the corresponding bit of the second operand. If the comparison is true, the corresponding bit in the result is set to 1. If the comparison is false, the corresponding bit in the result is set to 0.

| OPERATOR | OPERATION |
|----------|--------------|
| .EQV. | Equivalence |
| .OR. | Inclusive OR |
| .AND. | And |
| .XOR. | Exclusive OR |
| .NOT. | Complement |

Examples:

DISPLAY,1010B.XOR.1100B.

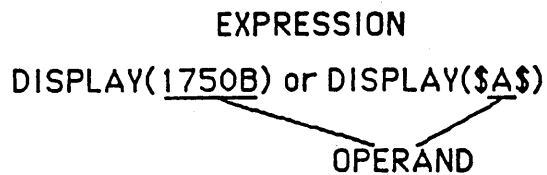
DISPLAY,1010B.EQV.1100B.

ORDER OF EVALUATION OF OPERATORS

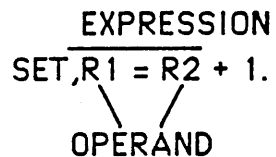
- The order in which operators in an expression are evaluated is:
 1. Exponentiation
 2. Multiplication, Division
 3. Addition, Subtraction, Negation
 4. Relations
 5. Complement
 6. AND
 7. Inclusive OR
 8. Exclusive OR, Equivalence
- Operators of equal order are evaluated from left to right.

EXPRESSION OPERAND TYPES

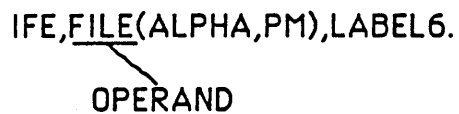
1. Operands can be numeric and literal string constants:



2. Operands can be NOS defined symbolic names:



3. Operands can be NOS defined functions:



CONSTANT OPERANDS

1. A constant is a string of 1 to 10 characters that NOS processes as an integer.
2. If first character is 0 to 9, all characters in the string must be numbers, except for a B or D optional post-radix.
3. If an operand must start with an alphabetic and is not a defined Job Control Symbolic Name or function it must be written as the special kind of constant known as literal.
4. LITERAL: A string of 1 to 10 characters delimited by "\$". For example, \$ A LITERAL\$.
5. Literals are interpreted as right-justified display code with binary zero fill, and processed as an integer.
6. If a literal is to be stored in Job Control register, it may be truncated. For example, SET(R1=\$AVERYLONG1\$).

SYMBOLIC NAME OPERANDS

- A string of characters predefined by NOS and having an assigned value.
- The value assigned is specified by the installation, NOS, or the user.
- With a few exceptions, all symbolic names have an initial value of Ø. -

Example:

IFE,EF=CPE,JUMP1.

IFE,OT=IAO,JUMP.

FUNCTION OPERANDS

- Used as expressions or operands within expressions in CCL statements
- There are 3 functions (prior to NOS 2.5.1):
 1. FILE(LFN,EXP) Determines if a file has a specified attribute.
 2. FILE(LFN,DT(MT)) DT determines device type on which a file resides. This example determines if LFN is assigned to 7-track magnetic tape.
 3. NUM(R1) Determines if string is numeric.
NUM(\$1978\$) If string contains special characters, it must be delimited by dollar signs and will always be evaluated as non-numeric.
- NOS 2.5.1 added a number of string functions to this set of file functions.

| | |
|-------------------------|--|
| STR(exp,first,last) | Produces a left-justified literal string. |
| STRB(numexp,first,last) | Produces a literal containing the octal value of "numexp". |
| STRD(numexp,first,last) | Produces a literal containing the decimal value of "numexp." |
- In STR/STRB/STRD, use positive values to count from left (use negative values to count from right).

| | |
|-------------|---|
| LEN(string) | Produces the character length of a literal (after \$ delimiters are removed). |
|-------------|---|

EXECUTION CONTROL COMMANDS

| COMMAND | DESCRIPTION |
|-------------|---|
| BEGIN | Initiates processing of a procedure |
| DISPLAY | Evaluates an expression and displays the result in the dayfile of the job |
| ELSE | Terminates skipping initiated by a false expression within an IF command or initiates skipping to a matching ENDIF command |
| ENDIF | Terminates skipping initiated by a matching IF, SKIP, or ELSE command |
| ENDW | Establishes the end of the loop |
| EXIT | Controls the command flow in the event of errors |
| IF (or IFE) | Conditionally skips one or more commands |
| NOEXIT | Disables EXIT error processing |
| ONEXIT | Enables EXIT error processing |
| REVERT | Returns processing from a procedure to the command record of procedure that called it |
| SET | Assigns values to user changeable NOS symbolic names |
| SKIP | Skips to the first matching ENDIF command |
| WHILE | Establishes the beginning of a loop. If the associated expression is true, the loop is processed; if it is false, the loop is not processed |

DISPLAY/SET

DISPLAY STATEMENT

- Evaluates an expression and sends result to dayfile
- Largest decimal value which can be displayed is 10 decimal (12 octal digits).
- Largest octal value which can be displayed is 20 octal (16 decimal) digits.

FORMAT:

DISPLAY (value or expression)

Example:

DISPLAY(DATE)

SET STATEMENT

- Assigns a value to a control register, error flag, dayfile-skipped-flag, or changes the current time-sharing subsystem.

FORMAT:

SET(SYM=EXP)

Examples:

SET(R1 = 7)
SET(R3 = R1 + R2 - 1)
SET(SS = BASIC)

EXIT/NOEXIT

EXIT STATEMENT

- Indicates position in the current command record where processing will resume if an error is encountered.
- Indicates where to terminate normal command processing if an error is not encountered.

FORMAT:

EXIT.

Example:

EXIT.
DAYFILE,OP=F.
*PROBLEMS OCCURRED.

NOEXIT STATEMENT

- Suppresses EXIT command processing.
- If an error occurs, processing continues with the next command instead of the command following the EXIT command.

FORMAT:

NOEXIT.

ONEXIT STATEMENT

- Reverses effect of a NOEXIT command.
- If an error occurs in processing the commands following ONEXIT, control transfers to the command following the next EXIT command.

FORMAT:

ONEXIT.

Example:

```
NOEXIT.  
GET,FILE.  
ROUTE,FILE,DC=PR.  
NOTE,,NR/FILE ROUTED  
ONEXIT.
```

```
NOEXIT.  
GET,MISSING.  
ONEXIT.
```

}

Is equivalent to GET,MISSING/NA.

IF/SKIP

IF STATEMENT

- Alternate form is IFE ("if expression").
- Conditionally initiates skipping of succeeding statements.
- Use to control flow, determine error, avoid errors, and recover from errors.
- If condition is true, next statement is processed.
- If condition false, skipping initiated until matching ELSE or ENDIF.

FORMATS:

1. IF,expression,label.
2. IF,expression.command.

SKIP STATEMENT

- Initiates unconditional skipping of succeeding control statements.
- Equivalent to an if statement with an expression which is never true.
- Only an ENDIF terminates skipping initiated by a SKIP.

FORMAT:

SKIP,label.

Example:

```
PURGE,JOHNNY.  
SKIP,CAT.  
EXIT.  
* PURGE DID NOT WORK  
ENDIF,CAT.
```

ELSE/ENDIF

ELSE STATEMENT

- Performs either
 - Terminate skipping initiated by false IF, or
 - Initiates skipping from ELSE to ENDIF. This happens for a true IF command.
- Neither SKIP nor ELSE terminates skipping initiated by another SKIP or ELSE.

FORMAT:

ELSE,label.

ENDIF STATEMENT

- Terminates skipping initiated by SKIP, IF, or ELSE.
- Label string on ENDIF must match.

FORMAT:

ENDIF,label.

Example:

```
IF,FILE(FTNBIN,..NOT.LO),COMPILE.  
GET,FTNIN/NA.  
FTN5,I=FTNIN,B=FTNBIN,L=0.  
RETURN,FTNIN.  
ENDIF,COMPILE.  
*SINGLE LINE IF.  
IF,FILE(FTNDATA,..NOT.LO).GET,FTNDATA/NA.  
IF,FILE(FTNDATA..NOT.LO).ATTACH,FTNDATA.
```

IF-ENDIF PROCESSING

PROCEDURE

```

      .
      .
      .
IF,condition,SAME.
      .
      .
      .
(commands)
      .
      .
      .
ENDIF,SAME.
      .
      .
      .
(commands)
  
```

PROCESSING IF CONDITION IS TRUE

```

      .
      .
      .
IF,condition,SAME.
      .
      .
      .
(commands)
      .
      .
      .
ENDIF,SAME.
      .
      .
      .
(commands)
  
```

Process
all
commands.

PROCESSING IF CONDITION IS FALSE

```

      .
      .
      .
IF,condition,SAME.
      .
      .
      .
(commands)
  
```

Skips to matching
ENDIF.

ENDIF,SAME. Continues
proces-
sing
after
ENDIF.

IF-ELSE-ENDIF PROCESSING

PROCEDURE

```

      .
      .
      .
IF,condition,MATCH.
      .
      .
(commands)
      .
      .
      .
ELSE,MATCH.
      .
      .
      .
(commands)
      .
      .
      .
ENDIF,MATCH.
      .
      .
(commands)

```

PROCESSING IF
CONDITION IS TRUE

```

      .
      .
      .
IF,condition,MATCH.
      .
      .
      .
(commands)
      .
      .
      .
ELSE,MATCH.↓

Skips to matching
ENDIF.

ENDIF,MATCH. | Continues
               | processing
               | after
(commands)    | ↓ ENDIF.

```

```
PROCESSING IF
CONDITION IS FALSE
```

```

      :
      :
      :
IF,condition,MATCH.↓

Skips to matching
ELSE.

ELSE,MATCH.
      :
      :
      :
(commands)  Continues
            processing
            after
            ELSE.

ENDIF,MATCH.
      :
      :
      :
(commands)↓

```

WHILE/ENDW STATEMENTS

- Bracket a group of control statements into a loop that can be repeatedly processed.
- Loop is repeated as long as expression in WHILE is true.

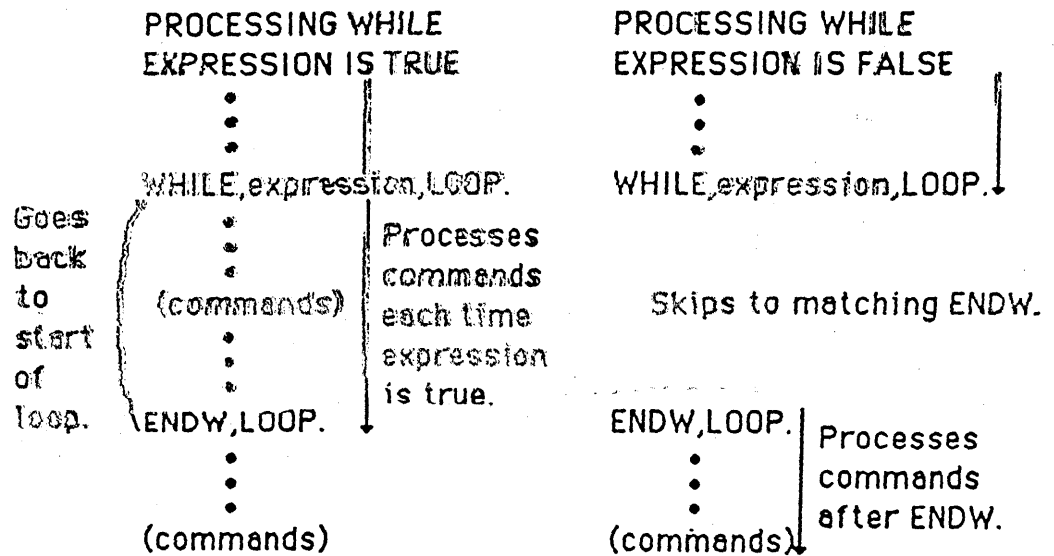
FORMAT:

```
WHILE,expression,label.  
ENDW,label.
```

Example:

```
*EMULATE REP=13 FOR AN IAF TERMINAL  
SET,R1=0.  
SET,R2=14.  
WHILE,R2.GT.R1,LOOP.  
SET,R2=R2-1.           Original plus 13 copies.  
REWIND,DATA.  
COPYSBF,DATA,LISTING.  
DISPLAY,R2.           for debugging  
ENDW,LOOP.  
ROUTE,LISTING,UN=MYBUDDY,DC=PR.
```


REPEATING COMMANDS



PROCEDURE FILES

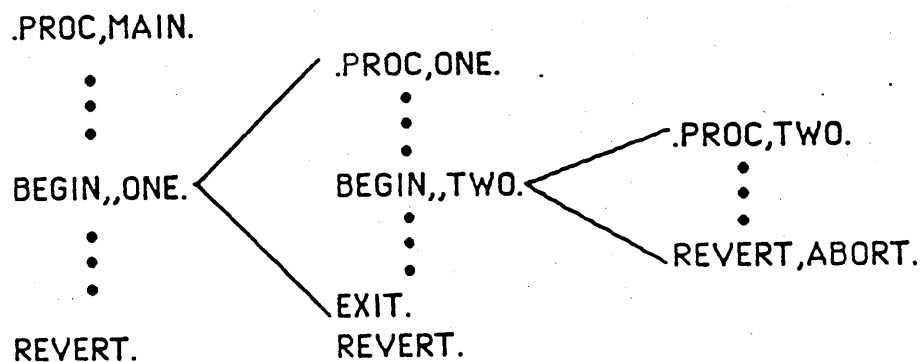
PROCEDURE

Definition:

"A user-defined set of instructions that is referenced by name. The instructions consist of procedure directives and system commands."

PROCEDURE FLOW

- Procedures can be nested.
- Control reverts to next command.
- Error exit is honored on REVERT,ABORT.



BEGIN STATEMENT

- Inserts a procedure into a control statement record or into another procedure
- Analogous to FORTRAN "CALL" statement

FORMATS:

BEGIN,PNAME,PFILE,P1,P2,...,Pn.

-PNAME,PFILE,P1,P2,...Pn. (interactive jobs only)

PFILE,P1,P2,...,Pn.

PNAME,P1,P2,...,Pn.

- See BEGIN statement in the NOS Reference Set, Volume 3, for PNAME and PFILE defaults under each of these formats.

Parameter formats:

KEYWORD

KEYWORD=

KEYWORD=VALUE

KEYWORD=SYMBOL+

KEYWORD=SYMBOL+B

KEYWORD=SYMBOL+D

BEGIN substitutes the value
of the symbolic name.

Given: file COMPILE containing:

.PROC,PASSIVE,IN,OUT,BINS.

*BEGIN PASSIVE,IN,OUT,BINS.

FTN5,I=IN,L=OUT,B=BINS.

REVERT,NOLIST.

EXIT.

DAYFILE(FR=*)

REVERT,ABORT.

Example:

/BEGIN,PASSIVE,COMPILE,SOURCE,LIST,LGO.

INTERACTIVE BEGIN STATEMENT

- Same as preceeding (passive) BEGIN formats except:

- Additional formats

BEGIN,PNAME,PFILE?

PNAME?

-PNAME,PFILE?

- Additional parameter formats:

KEYWORD?

?

MENU BEGIN STATEMENT

- Procedure call same as Interactive BEGIN.
- Additional parameter formats are

OMITTED

?

CHOICE?

CHOICE

REVERT STATEMENT

- Terminates procedure processing. Returns to calling procedure or job level.

FORMATS:

REVERT.
REVERT,ABORT.
REVERT,NOLIST.
REVERT,EX. NOS command statement...

- NOS always appends this sequence to a procedure at BEGIN time:

REVERT.CCL
EXIT.CCL
REVERT,ABORT.CCL

USER PROCEDURES

(PROLOGUES)

- **Definition:**

A procedure the system executes immediately after a user logs in. The user can define his/her own prologue to do such items as set terminal characteristics, execute commands, and call other procedures.

- **Usage:**

- System will cause prologue to execute
- Will execute each time user's name is validated in any mode (Interactive login or submitted batch jobs)
- If prologue fails, NOS will not issue the RECOVER command automatically.
- Prologue procedure name can be seen by using the LIMITS command.

SETTING UP A USER PROLOGUE PROCEDURE

- Create a procedure file with the desired commands.
- Pretest the procedure.
- Save the procedure as a permanent file.
- Inform the system of the prologue using command:

UPROC,FN=pfile

- or -

UPROC,pfile

Where:

pfile Permanent file containing the prologue

- View definition of prologue using LIMITS.
- If necessary, clear with UPROC.

PROCEDURE PARAMETERS

- Definition:

A value that follows the procedure name that alters the behavior of the procedure

- Usage

- File names
- Variable constants checked for flow control
- Parameters on system commands

BEGIN/.PROC COMPARISON

BEGIN Command

Procedure Header Directive

| | |
|---|--|
| pname, | .PROC,pname*I, |
| ↑-----↑ | |
| keywrd ₁ =value ₁ , | keywrd ₁ "description ₁ "=(checklist ₁), |
| ↑-----↑ | |
| keywrd ₂ =value ₂ , | keywrd ₂ "description ₂ "=(checklist ₁), |
| ↑-----↑ | |
| • | • |
| • | • |
| • | • |
| keywrd _n =value ₁ | keywrd _n "description _n "=(checklist ₂). |
| ↑-----↑ | |

PROCEDURE PARAMETER SUBSTITUTION

- Order dependent
 - Compares parameters on the BEGIN and PROC commands in the order they appear; then substitutes in procedure
 - Ignores all excess on the BEGIN command

Example:

```
.PROC,START,IN=IN,OUT=OUT,LFN=MINE. (procedure header)
```

```
BEGIN,START,,,YOURS.
```

The default is used for IN and OUT while YOURS is substituted for MINE

- Order Independent
 - Parameter comparison is based on keyword matching.
 - NOS switches to independent order when:
 - + BEGIN contains a KEYWORD= or KEYWORD=VALUE format
 - + Procedure header contains KEYWORD=DEFAULT1/DEFAULT2
 - + A slash (or reverse slash) separator is encountered in the user's BEGIN command (or in the PROC parameter list).
 - Matching always begins in order dependent mode
 - You cannot switch back from order independent mode to order dependent mode.

INTERACTIVE PROCEDURES

Definition:

A mode of processing by a procedure in which the user enters a response to parameter prompts which cause parameter substitution prior to procedure execution.

INTERACTIVE PROCEDURE STRUCTURE

Procedure header Contains either pname *I for Interactive or pname *M for Menu type.

Formatting section

Help section (optional)

Procedure Body

NOS Command section

Data section (optional, generates local file)

OPTIONAL INTERACTIVE PROCEDURE DIRECTIVES

- These DOT directives are interpreted at BEGIN time and not seen by NOS.
- DOT directives are listed inside the back cover of NOS Reference Set, Volume 3.

| DIRECTIVE TYPE | DIRECTIVE | FUNCTION |
|----------------|------------------|---|
| Formatting | .CORRECT,prompt. | Specifies the prompt to follow an incorrect parameter entry |
| | .ENTER,prompt. | Specifies the prompt to appear before a parameter entry is made |
| | .Fn,label. | Specifies a label for programmable function key Fn |
| | .NOCLR,message. | Inhibits the automatic clearing of the screen while the procedure executes |
| | .NOTE,message. | Specifies a message to appear on the screen while the procedure executes |
| | .PAGE,label. | Specifies the string to precede the page number on the screen for procedures that require continuation screens. |
| | .PROMPT,text. | Specifies the text for the general information on how to proceed. |

Example:

.PROMPT, Select by Number or enter ? for help.

PROCEDURE DIRECTIVES (CONT)

| | | |
|-----------|---------------|--|
| Branching | .IF,label. | Allows conditional expansion of the procedure body |
| | .ELSE,label. | Terminates skipping initiated by a matching. .IF initiates skipping until a matching .ENDIF is found |
| | .ENDIF,label. | Terminates skipping initiated by a matching .IF or .ELSE |
| | .EX.command. | Submits a single command to the system for immediate execution |
| File | .DATA,lfn. | Creates a local data file |
| | .EOF or .EOP | Used to insert an End-Of-File mark in a data file created by the procedure |
| | .EOR or .EOS | Use to insert an End-Of-Record mark in a data file created by the procedure |
| Comment | * comments | Identifies comments in the procedure. These will not be dayfiled since NOS will not see them. |

HELP DIRECTIVES

- Used to denote information about the procedure or its parameters.
- Accessed later at BEGIN time by typing A "?" or by pressing HELP key during prompting sequence.

FORMATS:

| | |
|-----------------------|--|
| .HELP. | Supplies general help information. |
| .HELP,,NOLIST. | Suppresses NOS supplied help information. |
| .HELP,keyword. | Supplies help on a specific procedure call Keyword (or menu item on menu procedures). |
| .HELP,keyword,NOLIST. | |

END HELP DIRECTIVE

- Occurs once.
- Terminate help section.

FORMAT:

.ENDHELP.

USING HELP DIRECTIVES

RESULT

| PROCEDURE CALL | INTERACTIVE PROCEDURE(*I) | MENU PROCEDURE |
|-------------------|---|---|
| pname | The system prompts for parameters. | The system displays the menu and prompts for a selection. |
| pname? | The system provides .HELP text about the the procedure itself and then prompts for parameters. | The system provides any about the procedure itself, displays the menu, and prompts for a selection. |
| pname,? | SAME AS pname?. | You get an error message followed by the menu and a prompt for a selection. |
| pname,keyword? | The system provides any .HELP text for the parameter associated with keyword and then prompts for parameters. | SAME AS pname,?. |
| pname,choice? | Not applicable. | The system provides any .HELP text for that menu selection, displays the menu, and prompts for a selection. |

PARAMETER SUBSTITUTION/CONCATENATION

- Applies to both command and data sections of procedure body.
 - # - Graphics pound sign
 - Inhibits substitution
 - _ - Underscore
 - Concatenation of two parts after parameter substitution

MENU-DRIVEN PROCEDURES

Definition:

A procedure constructed so that when called by the user, displays several selections as to what the procedure can do and acts based on user selection from the menu.

FORMAT:

```
.PROC,pname*M"title",OPTION=
(1"first selection"
,2"second selection"
,N"Nth selection")
.ck
```

Where: ck Is an optional one to ten character comment keyword.

Example:

```
.PROC,UNEM*M"FILE ROUTING OPTIONS",OPTION=
(1"PRINT A FILE"
,2"PUNCH A FILE"
,3 "PLOT A FILE")
.
*BEGIN UNEM,#OPTION=OPTION.
IF,OPTION.EQ.1.MYPRINT.
IF,OPTION.EQ.2.MYPUNCH.
IF,OPTION.EQ.3.MYPLOT.
REVERT,NOLIST.
EXIT.
DAYFILE,FR=*.
REVERT,ABORT.
```

OR REVERT,EX.BEGIN,UNEM_OPTION.

INTERACTIVE PROCEDURE PARAMETER - PROMPTING FORMAT

- Provides validation of values passed at BEGIN time.
- Allows default values.
- Allows multiple keywords.

`.PROC,pname*!"title",p1,p2,...,pn.ck.`

Where:

| | |
|--------------------|---|
| <code>pname</code> | One to seven character procedure name |
| <code>title</code> | Zero to forty character title to the procedure |
| <code>pi</code> | Optional keywords to a maximum of fifty. Format is keyword=(valid checklist patterns) |
| <code>.ck.</code> | One to ten character comment keyword (optional) |

PARAMETER CHECKLIST PATTERNS

| CHECKLIST PATTERN | PARAMETER REQUIREMENT |
|-------------------|--|
| * A | Can be any string of 1 to 40 characters |
| * F | Can be a file name |
| * K | Can be the keyword itself |
| * N or * D | Can be omitted from the procedure call |
| * Sn(set) | Can contain n or fewer elements from the specified set |
| string | Can be the specified string |

- Each checklist pattern has the following three formats and each format specifies a different substitution scheme

| | |
|---------------|---|
| PATTERN=VALUE | Substitutes value for the keyword. |
| PATTERN= | Substitutes an empty (null) string for the keyword. |
| PATTERN | Substitutes the key word itself (in effect, no substitution). |

INTERACTIVE PROCEDURE EXAMPLE

- Procedure which will compile a FORTRAN program
- Use the user-specified files for input and output or use the default names provided in the procedure header

→ .PROC,RUNF*I"Compile from Input File"
 ,IN"-Input file"=(*F,*N=INFILE)
 ,OUT"-Output file"=(*F,*N=OUTFILE)

 .
 FTN5,I=IN,L=OUT,GO.
→ *BEGIN RUNF,#IN=IN,#OUT=OUT.
 REVERT,NOLIST.
 EXIT.
 DAYFILE,FR=*.
 REVERT,ABORT.

- Since all keywords have default values defined (*N), this procedure will not prompt for any keyword values unless called with an invalid file name (e.g., greater than seven characters).

EXERCISE 6.1

FLOW CONTROL COMMANDS AND PROCEDURE FILES

- * Remember to test all procedures twice.
- 1. Type in some Flow Control statements with variations on the syntax. Type some statements in with errors. What message(s) appear?
- 2. Obtain values for the symbolic names that are set by the system.
- 3. Set registers to different values and display them to make sure the new values are there.
- 4. Write a simple procedure file(s) using all Flow Control Commands (e.g., IF, ENDIF, EXIT.) and type into the system. Execute the procedure(s).
- 5. Using FORTRAN program, _____ (see next page) under username _____, write a procedure file to get that file. Execute the binaries. Be sure to save this procedure file - you will be using it later.
- 6. Write a User Procedure to put your terminal into Screen mode and put a message to the screen when the procedure has finished execution.
- * Make use of exit error processing in all procedures.
- * Remember to save all files and procedures you create ! ! ! !

SAMPLE FORTRAN PROGRAM

```
PROGRAM DEHFTN5
CHARACTER LAST*10,FIRST*10
REAL RATE, GROSS, NET, HOURS, FTAX
REAL FRATE
FRATE = 0.30
PRINT*, 'ENTER LAST NAME, FIRST NAME - BOTH IN SINGLE QUOTES'
READ*, LAST, FIRST
PRINT*, 'ENTER PAYRATE, HOURS WORKED - NO QUOTES'
READ*, RATE, HOURS
GROSS = HOURS * RATE
FTAX = GROSS * FRATE
NET = GROSS - FTAX
PRINT*, 'INDIVIDUAL PAYROLL STATISTICS'
PRINT*, ' '
PRINT*, 'EMPLOYEE NAME: ', FIRST, LAST
PRINT*, 'PAYRATE: ', RATE
PRINT*, 'HOURS WORKED: ', HOURS
PRINT*, 'GROSS PAY: ', INT(GROSS*100+0.5)/100.00
PRINT*, 'FEDERAL TAXES: ', INT(FTAX*100+0.5)/100.00
PRINT*, 'NET PAY: ', INT(NET*100+0.5)/100.00
PRINT*, ' '
PRINT*, 'END OF INDIVIDUAL PAYROLL'
STOP
END
```

EXERCISE 6.2

PROCEDURE HELP

Create an interactive procedure file that has three parameters. Write a help section that contains help for the procedure and for each individual parameter.

Hint: You will be building on this procedure and will later use this help section in a interactive procedure that will have three parameters for the execution of an FTN5 program. The parameters will be "IN" for the input file, "OUT" for output file, and "BIN" for the file to contain the binaries.

Practice obtaining help on the procedure itself and on permissible parameter values.

EXERCISE 6.3

MENU-DRIVEN PROCEDURES

You've been so innovative so far, just create a menu-driven procedure to call procedures from those that you've worked on so far or from the HELPLIB file under username LIBRARY. It can be as simple or as complicated as you wish. Be sure to try out help options with MENU procedures.

- * It is a good practice to check that a new or revised procedure returns all unnecessary local files after it completes.

EXERCISE 6.4

INTERACTIVE PROCEDURES

Expand upon previous help procedure file. Have three parameters in the .PROC LINE: "IN" to prompt for the input file, "OUT" to prompt for the file to contain the output, and "BIN" to prompt for the file on which to place the binaries from the FTN5 compile. Have the statements necessary to both compile and execute the FTN5 source program (DEHFTN5).

Use different checklist patterns when creating this procedure so that you become familiar with how substitution occurs.

LESSON 7

INTRODUCTION TO LIBRARIES

LESSON 7

INTRODUCTION TO LIBRARY GENERATION

Lesson Preview:

This lesson is intended to provide an introduction to binary library creation.

Objectives:

After completing this lesson, the student will:

- Have a basic understanding of binary library creation.
- Be able to set up a simple job to create a user library file.
- Declare the user library file to be a global library file.

Projects: Exercise 7.1

References:

- NOS Version 2 Reference Manual, Volume 3, Section 15 (LIBGEN statement)

The applicable meaning of the term LIBRARY must be determined from its context.

USER NAME LIBRARY - Username LIBRARY is a reserved username defined at system installation. This username can contain both text and programs, which is easily accessible by authorized users from a centralized location. Files stored under username LIBRARY need not be libraries themselves.

PROGRAM LIBRARIES - A program library is a collection of compressed source deck images stored in MODIFY or UPDATE format. Program Libraries can be recognized by the OPL/UPL type records contained within them.

USER LIBRARIES - User libraries are files which are searched by CYBER loader to satisfy external references within a program it is loading. They contain compiled or assembled routines. User libraries can be recognized in a file by the ULIB type record which defines them.

GLOBAL LIBRARIES - A Global library is a set of user library formatted files.. This library is searched:

1. By NOS for a program name, or procedure name that matches the command name, or
2. By CYBER loader to satisfy external references within a program it is loading. Global, in this context, implies that the user library files do not have to be specified "local" to each CPU program load sequence.

In this lesson, "LIBRARY" will refer to USER or GLOBAL libraries only.

USER LIBRARIES

A USER LIBRARY is a file containing records that are accessed individually. Library records can be of several types, including:

| | |
|------|---|
| ULIB | User library directory (first record) |
| ABS | Multiple entry point overlay |
| CAP | CYBER loader capsule |
| OVL | Overlay |
| PROC | Procedure file |
| REL | Relocatable user program |
| TEXT | Source or data (column 1-7 of first line is nonblank) |
| OPLD | Old program library directory (last record) |

Record types can be determined by using CATALOG.

Two types of files are allowed:

- Sequential (Library file input); slower access
- Random (User Library format)

When a command is encountered by NOS, the search order is:

1. If command is prefixed by a \$, go to step 5.
2. Search the user's local files or Global Library Set for a matching file name.
3. Search Indirect access and Direct Access permanent files of the user if BEGIN.
4. Search Indirect access and Direct access permanent files of user name LIBRARY if BEGIN.
5. Search the central library directory for a matching program name.
6. If three character name and user is permitted to call PP programs as commands, search PP program names.

LIBRARY CREATION

LIBGEN COMMAND

- Creates user library of routines for use with NOS and the CYBER Loader.

FORMAT:

LIBGEN,params.

Where params are any of the following in any order:

| | |
|---------------------|--|
| F=lf _n 1 | Name of file containing records to be placed on user library file lf _n 2. Default file name is LGO. |
| P=lf _n 2 | Name of the file on which the user library is to be written. Default file name is ULIB. |
| N=lf _n 3 | Name of the user library being generated, this name becomes the name of the ULIB and OPLD records. Default is lf _n 3=lf _n 2. |
| NX=n | If n is nonzero, no cross references are given. Decks are not cross-linked in the ULIB directory (avoid duplicate entry points on loads). Default is NX=0. |

Example:

```
/LIBGEN,F=RECORDS,P=USERLIB,N=LIBNAME.
```

This statement will create a new user library called USERLIB from the records contained on file RECORDS. The library name in ULIB record will be LIBNAME.

```
/ULIB?
```

Calls a NOS procedure to aid in using LIBGEN.

```
/CATALOG,LIBNAME,R,U,N.
```

To see user library contents.

DECLARING A
GLOBAL LIBRARY

LIBRARY COMMAND

- Specifies a global library set for your current job or session.
- To execute a procedure or program from a global library file, you need only to specify a procedure name or entry point name.
- A ZZZZZLD local file will be opened to allow random access into the current global library set.

FORMAT:

LIBRARY. To clear the current set of library files.

LIBRARY, lfn₁,...,lfn_n/directive.

Where:

lfn Local filename or system library containing the user library. Up to 24 files can be referenced in your global library set.

directives

- | | |
|---|--|
| A | Add specified files to the current global library set |
| D | Deletes the specified files from the current global library set |
| R | Replace the global library set with the specified library set (default). |

LIBRARY EXAMPLES

Examples:

`/LIBRARY,LIB1,LIB2/A.`

Add files LIB1 and LIB2 to the current global library set.

`/LIBRARY.`

Clear the current global library set.

`/ENQUIRE,L`

See the current global library set.

)

MODIFYING A LIBRARY

LIBEDIT COMMAND

- Generates a file with records copied from one or more other files

FORMAT:

LIBEDIT,params.

Where params optional parameters:

P=lfm Edit old file user library lfm - default is OLD

N=lfm Write new user library file here, default is NEW

I = lfm Input directives here - default is INPUT, I=Ø means no directives

B=lfm User records from this file for insertions and replacements

U Call LIBGEN to create new user library file

L=lfm List output here - OUTPUT is default

LO=listopt Where listop can include the following:

C List directives

E List errors

M List modifications

N List records written to new file

F Full listing

Z Directives appear on command line with a separator scheme like ENTER, NOTE, and BLOCK commands.

LIBEDIT EXAMPLES

Example:

```
/LIBEDIT,P=USERLIB,I=Ø,N=NEWLIB,B=NEWRECS,L=LIST,U,LO=F.
```

Old user library, USERLIB, will be used as input to LIBEDIT. The new library created (U) will be placed on NEWLIB. No input directives are needed, but insertion records will be read from file, NEWRECS. LIST will contain a full listing from this run.

```
/ULIB?
```

Calls a NOS procedure to aid you in updating user library files.

LIBEDIT DIRECTIVES

ADD DIRECTIVE

- Inserts records before a Ø-length record

FORMAT:

*ADD LIB_n,gid₁,gid₂,...,gid_n.

Where:

| | | |
|-------------------|-----------|------------------------|
| gid has the form: | type/name | |
| | name | (default type) |
| | type/* | (* = all) |
| | * | (all of default types) |
| | gid-gid | (gid = any of above) |
| | 0 | (0 length record) |

DELETE DIRECTIVE

- Suppresses copying of the specified records from the old file to the new file

FORMAT:

*DELETE gid₁,gid₂,...,gid_n.

*D gid₁,gid₂,...,gid_n.

INSERT or AFTER DIRECTIVE

- Copy the specified records or groups or records from current replacement file after copying specified old file record onto the new file

FORMAT:

*INSERT rid, gid₁,gid₂,...,gid_n.

*I rid, gid₁,gid₂,...,gid_n.

*AFTER rid, gid₁,gid₂,...,gid_n.

*A rid, gid₁,gid₂,...,gid_n.

REPLACE DIRECTIVE

- Replaces old file records with replacement file records

FORMAT:

*REPLACE gid₁,gid₂,...,gid_n.

LIBEDIT DIRECTIVE EXAMPLE

Example:

Using LIBEDIT to create a library.

LIBEDIT doesn't create libraries. The U parameter calls LIBGEN which creates the library.

```
/LIBEDIT,P=Ø,N=TESTLI,B=PROCFIL,U.
```

ENTER DIRECTIVES -

? *BUILD,LIBR Names directory to be built.

? *B,*,PROC/*

? C/R only to end input.

RECORDS WRITTEN ON FILE TESTLI

| | RECORD | TYPE | FILE | DATE | COMMENT |
|----------|---------|------|---------|------|---------|
| INSERTED | CREATE | PROC | PROCFIL | | |
| INSERTED | CRESAVE | PROC | PROCFIL | | |
| INSERTED | CTFILE | PROC | PROCFIL | | |
| INSERTED | CZFILE | PROC | PROCFIL | | |
| INSERTED | IWFILE | PROC | PROCFIL | | |
| INSERTED | FDUMP | PROC | PROCFIL | | |
| INSERTED | DM | PROC | PROCFIL | | |
| INSERTED | SORTMRG | PROC | PROCFIL | | |

LIBRARY GENERATION COMPLETE.

```
/SAVE,TESTLI  
/LIBRARY,TESTLI  
/CRESAVE
```

CMF ALREADY PERMANENT.

GTR COMMAND

- Get the Record (GTR) copies of records from a library on a specified local file.

FORMAT:

GTR,Ifn1,Ifn2,d,NR,S,NA. directives

Where:

| | |
|------------|---|
| Ifn1 | File which is searched for requested records. |
| Ifn2 | File where requested records are written. |
| d | Write new random access directory (OPLD). |
| nr | No rewind option. |
| s | Search Ifn1 sequentially for requested records. |
| NA | No abort option. |
| directives | Specifies record or group of records to be retrieved. |

Example:

/CATALOG,MYLIB,R,U,N.
/GTR,MYLIB,LOCLIB.REL/BIN

Get the relocatable (REL) record BIN from a library called MYLIB and make a copy of it on a local file called LOCLIB.

/CATALOG,LOCLIB,R,U,N.
/ULIB?

Can be used in place of GTR to extract records from a user library file.

EXERCISE 7.1

1. Access the source program library containing MAIN, SUB1, and SUB2, extract and compile these programs and execute them twice.
2. (Required) Access the same library but only extract SUB1 and SUB2, compile them, and use the NOS ULIB procedure to create a user library of these two subroutines and make a permanent file of this user library.
3. Use the NOS ULIB procedure to create another user library which contains SUB1, SUB2 and several procedure files, and make it a permanent file and add it to your global library set.
4. CATALOG your user library file.
5. Review ENQUIRE,L output.
6. Where would be a good place to set up your global library set?

Using FSE, add the necessary command(s) there.

Logout and log back in. Did it work? How do you know?

7. (Optional) Using GTR, get a record from the library created in #2, change the record name and LIBEDIT, this record back into the library - catalog it.

LESSON 8

**SOURCE PROGRAM LIBRARY
MAINTENANCE**

LESSON 8

SOURCE PROGRAM LIBRARY MAINTENANCE

Lesson Preview:

The purpose of a source-program library and how it is created, used, and maintained is discussed in this lesson. Also, UPDATE, the utility librarian, is exercised in its various call parameters and source-file directives.

Objectives:

After completing this lesson, the student will be able to:

- List the main advantages of a source level library.
- Use the UPDATE utility, control parameters, and directives to create, access, modify, and regenerate a source program library.

Projects: Exercise 8.1

References:

- UPDATE Reference Manual (Pub. number 60449900)

PURPOSE AND OVERVIEW OF A SOURCE PROGRAM LIBRARY

In the section on permanent files, one of the projects involved is working with a file containing a source program. With a single program of this size, editing and storing the file poses no special problems. However, maintaining a massive program with many subroutines, eg., Product Set code, generates two major concerns.

The first one is the lack of efficient editing ability on a line-by-line basis, and the other is the large gaps (blank columns) in most free format coding languages. These get stored and take up a great deal of space.

The source program librarian, UPDATE, provides a means of overcoming these two limitations as its basic function as well as other extended functions.

The program library is essentially a group of randomly organized logical records, each of which contains compressed line images of a deck. Continuing the example of the simple FORTRAN program we have been using, the first card (today, a line in a text file), "PROGRAM MAIN (OUTPUT)", would be the first element of a deck, logically named, "MAIN".

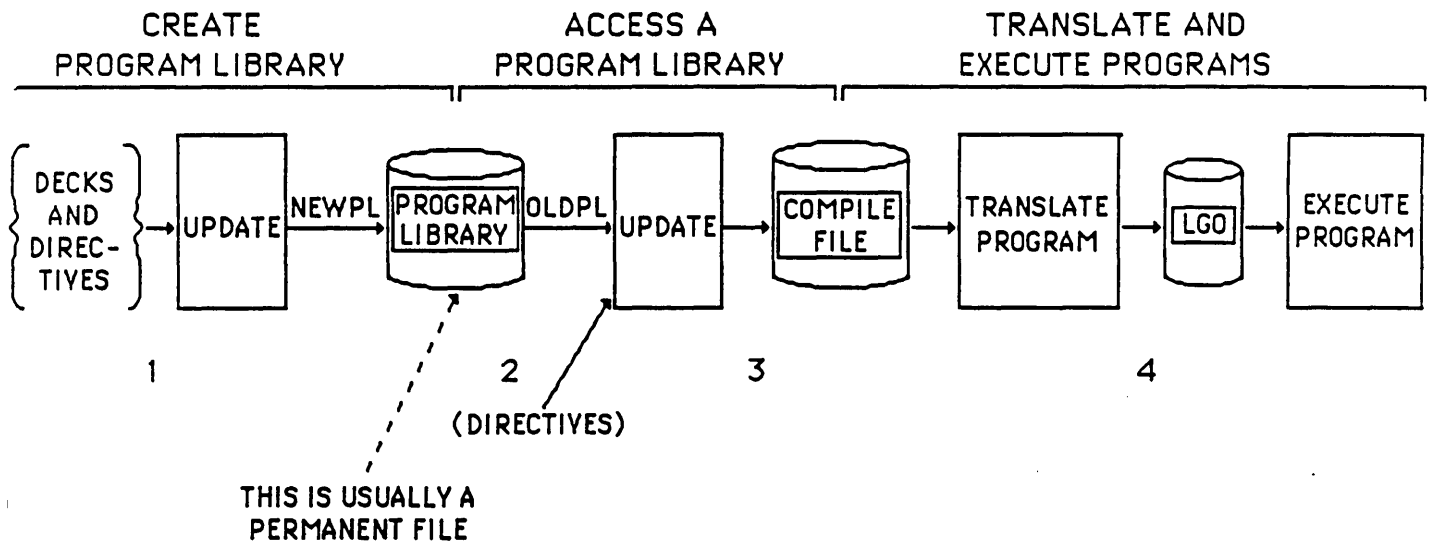
A program library must be created, used, and maintained. Figure 8.1 shows the main components and phases of this process. The important files are the program library file and the file on which extracted programs are placed for assembly or compilation.

The program library file when it is first created is named new program library (NEWPL). When it is used subsequently to extract programs from, it is referred to as an old program library (OLDPL).

The file of extracted routines is called the COMPILE file.

The program library can contain any symbolic data, and could more appropriately be referred to as a source program and data library.

PROGRAM LIBRARY - OVERVIEW, ACCESS AND EXECUTION



1. The PROGRAM LIBRARY is initiated at some point as line images.
2. The PROGRAM LIBRARY is saved as a permanent file, in random organization for processing efficiency.
3. When programs are required, directives extract decks and write them on a compile file.
4. The compile file is assembled or compiled into load modules which can be loaded and executed.

Figure 8-1.

CREATION OF A SOURCE PROGRAM LIBRARY

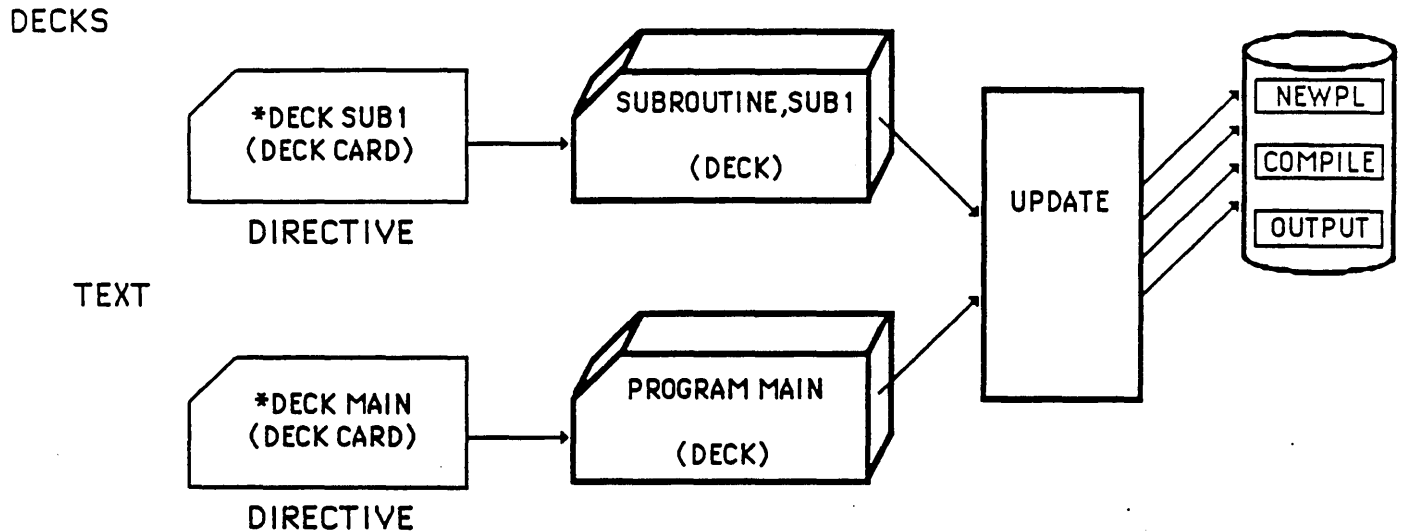


Figure 8-2. Create a Source Program Library

Figure 8.2 shows the elements which are involved in the creation of a source program library. No changes are required for the program decks MAIN and SUB1 except that the deck name is added as a card (today, a line) in front of each deck. In earlier examples, the FORTRAN programs were on a file as one logical record. This provides the convenience of making only one compiler call.

The decks in the current example are one logical record. This is generally true of genuine subroutines. Each deck is therefore kept and referenced as a random record and written with or without modification to the file from which the compiler can use them. This compile file can be formatted by directives to contain any grouping of logical records or files.

```

) .PROC,CREATPL      THIS PROC PLACES THE FTN PROGRAM MAIN
COMMENT.             AND SUBROUTINE SUB1 AS SEPARATE DECKS
COMMENT.             ONTO A PROGRAM LIBRARY FILE WHICH IS
COMMENT.             THEN CATALOGED AS A PERMANENT FILE.
DEFINE,PL            RESERVE PERMANENT FILE SPACE.
UPDATE,N=PL,
I=UPDIR.             THIS IS A NORMAL TYPE UPDATE.
SAVE (PL=MAINSUB).

```

```

.DATA,UPDIR.
*DECK MAIN
  PROGRAM MAIN
  _____
  _____
  _____
  _____
  _____
  END
*DECK SUB1
  SUBROUTINE SUB1
  _____
  _____
  _____
  _____
  _____
  END
(EOR)

```

These FORTRAN programs could be compiled and executed from the INPUT file by the addition of BKSP{INPUT} and FTN5{GO} since FORTRAN considers an * in column 1 to be a comment card.

The statistics and report from the UPDATE run are listed on the file OUTPUT.

UPDATE RUN TYPES

- Creation run (UPDATE detects automatically)
 - Input to update creation run
 - + Directives
 - + Text
 - Output from update creation run
 - + Compile file
 - + New program library (NPL)
 - + List file
 - + Source file
- Correction run
 - Input to update correction run
 - + Directives
 - + Old program library (OPL)
 - + Source
 - + Text
 - Output from update correction run
 - + Compile file
 - + New program library (NPL)
 - + List file
 - + Source file
- Copy run
 - Convert a program library from random or sequential access format to the opposite format.
 - Used when program libraries are transferred to/from tape.

UPDATE MODES

1. F - FULL UPDATE

All decks on old program library are written to new program library, compile file, and source file.

2. Q - QUICK UPDATE

Only decks mentioned on compile directives are written to the new program library, compile file, and source file.

3. NORMAL - SELECTIVE UPDATE

Only decks mentioned on compile directives or decks modified are written to the compile file.

UPDATE CONTROL STATEMENT

- Utility called by this statement
- Parameters specify options and files for the run

FORMAT:

UPDATE(params)

Where:

params optional parameters

C Compile file name (default COMPILE)

I Input stream file name (default INPUT)

N New program library file name (default NEWPL)

P Old program library file name (default OLDPL)

S Source file name (default SOURCE)

D 80 columns of data to be written to compile file

F Full UPDATE mode

Q Quick UPDATE mode

O List file name (default OUTPUT)

L List Options

UPDATE CONTROL STATEMENT EXAMPLES

Examples:

UPDATE,I=DIRTEXT,C=Ø,N=NEWPL,S=SOURCE,F.

This is a creation run. The directives and text are on file DIRTEXT, no compile file will be generated, the PL created will be named NEWPL, the file to which the source will be written will be named SOURCE. Full update mode is requested.

UPDATE,P=OLDPL,I=DIRTEXT,N=NEWPL,D,C=COMPILE,Q.

This is a correction run. The directives and text are on file DIRTEXT and NEWPL will contain the new program library. This is an update run using quick mode, so only decks mentioned on Compile directives are written on file COMPILE in 8Ø column format. No source file is generated.

UPDATE DIRECTIVES

- Identify decks
- Control input file
- Must begin with the master control character (* by default)
- Can specify comments

FORMAT:

*keyword params

Where:

keyword Update directive

params Parameters identifying decks, cards, lines, or files

CALL DIRECTIVE

- Causes a common deck (COMDECK) to be written onto the compile file

FORMAT:

***CALL comdeck**

COMDECK DIRECTIVE

- Establishes a common deck that can be called from other decks

FORMAT:

***COMDECK deck,NOPROP**

Where NOPROP determines whether any changes are propagated throughout the Program Library.

COMPILE DIRECTIVE

- Indicates which decks are written to the compile file

FORMAT:

***COMPILE deck₁,deck₂,...,deck_n**

DECK DIRECTIVE

- Establishes a deck on the program library
- Must be a unique name

FORMAT:

***DECK deckname**

IDENT DIRECTIVE

- Names set of corrections being made
- Lines added in the correction set are sequenced within the name specified

FORMAT:

*IDENT idname,B=num,K=ident,U=ident

Where

| | |
|-------|--|
| num | The beginning line sequence to be used. |
| ident | A correction identifier which must be known (K) or undefined (U) in order for this identifier to activate. |

DELETE DIRECTIVE

- Deactivates a line image or group of line images
- Optionally inserts text and directives

FORMAT:

*DELETE line

*DELETE line_n,line_m

INSERT DIRECTIVE

- Inserts text line images or compile file directives in the PL after the line specified
- Use BEFORE directive (same format) to insert before a line.

FORMAT:

***INSERT line**

YANK DIRECTIVE

- Temporary reversal of the impact of a correction set.

FORMAT:

***YANK ident**

PURGE DIRECTIVE

- Permanent removal of a correction set.

FORMAT:

***PURGE ident**

ACCESSING PROGRAMS FROM A PROGRAM LIBRARY

.PROC,ACESPL
COMMENT.
COMMENT.
GET(OLDPL=PL)
UPDATE(Q),I=UPDIR.
FTN5(I=COMPILE)
LGO.
.DATA,UPDIR.
*IDENT ACCESS
*COMPILE MAIN,SUB1
(EOR)

THIS PROC ATTACHES A PROGRAM LIBRARY FROM
PERMANENT FILES, EXTRACTS PROGRAMS
MAIN AND SUB1.

THIS IS A QUICK TYPE UPDATE.

With a quick UPDATE, the routines to be extracted from the source program library are specified on the *COMPILE directive card.

With only the two programs, Main and Sub1 on the library, an UPDATE(F),I=0, and no *COMPILE card will produce the same result: Program Main and Sub1 are in source image form on a file whose name is COMPILE. The full UPDATE places all decks from the program library onto the Compile file.

MODIFYING PROGRAMS ON A PROGRAM LIBRARY

```
.PROC,CORPGLB.      THIS PROC ACCESSES A PROGRAM LIBRARY,  
COMMENT.            MAKES CORRECTIONS TO PROGRAM MAIN AND  
COMMENT.            SUB1, AND PLACES THE CORRECTED DECKS ONTO  
COMMENT.            THE COMPILE FILE.  
GET(OLDPL=PL)  
UPDATE,I=UPDIR.  
FTN5,I=COMPILE.  
LGO.  
.DATA,UPDIR.  
*IDENT    COR1  
*INSERT   MAIN.3  
  
    CARD(S) TO BE INSERTED  
*DELETE   SUB1.4  
*BEFORE   MAIN.5  
    CARD(S) TO BE INSERTED  
(EOR)
```

Since each deck had a modification, both decks will be written to the Compile file in this, the normal UPDATE. In the case where it is desired to have decks written to the Compile file and which are not being modified, these decks will require specification with a *COMPILE directive.

There are two additional steps that could be performed by this UPDATE:

- The creation of a new program library which has the correction set added to it.
- The creation of a Source file which could be subsequently input to an original create-type UPDATE.

The advantage of the first addition, all new program library with corrections posted on them prevents the problem of having to deal with correction sets.

The Source file is not of any advantage in the uncluttered, early stages of a source program library, but is useful to simplify identifiers at later, more complex stages.

The parameters on the UPDATE card to accomplish both of these are:

```
UPDATE{N,S,I=UPDIR}
```

the rest of the job is the same, except that now it would be logical to save the NEWPL and purge the OLDPL.

RESCIND PROGRAM LIBRARY MODIFICATIONS

REGRESS TO EARLIER STATE

| | |
|-----------------|-----------------------------------|
| .PROC, RESCIND. | THIS PROC REMOVES THE EFFECT OF A |
| COMMENT. | CORRECTION SET BY USE OF A YANK |
| COMMENT. | DIRECTIVE. |
| GET(OLDPL=PL10) | |
| UPDATE,I=UPDIR. | A NORMAL UPDATE IS NEEDED. |
| FTN5,I,GO. | |
| .DATA,UPDIR. | |
| *IDENT | NEGATE |
| *YANK | COR1 |
| (EOR) | |

The OLDPL still has the changes defined by correction set identifier COR1.

If an S parameter were added to the UPDATE card, the resulting Source file could be put through an original create to obtain a program library that is an exact duplicate of the previous procedure (CREATPL) result.

If an N parameter named a NEWPL on the UPDATE statement, the new library would have the correction set COR1 flagged as inactive. A *PURGE directive, instead of the *YANK would make the rescinding irreversible.

ADD DECKS TO PROGRAM LIBRARY

```
.PROC,ADDECK      THIS PROC PLACES TWO NEW DECKS ON TO A
COMMENT.          NEW PROGRAM LIBRARY, IN ADDITION TO THE
COMMENT.          DECKS FOUND ON THE OLD PROGRAM LIBRARY.
GET(OLDPL=PL23)
UPDATE{N,C=O,I=UPDIR}
SAVE(NEWPL=PL45)
.DATA,UPDIR.
*IDENT  COR2
*INSERT MAIN.n
```

{Change(s) Main to make CALLs to Sub2 and Sub3}

```
*ADDFILE SUBS
.DATA,SUBS.
*COMDECK  SUB2
          SUBROUTINE SUB2
          _____
          _____
          _____
          _____
          _____
          END
*COMDECK  SUB3
          SUBROUTINE SUB3
          _____
          _____
          _____
          _____
          _____
          END
```

(EOR)

EXERCISE 8.1

Given on the next page:

1. A FORTRAN program deck with some statements in error and some missing statements.
2. Data that is input to the program.
3. The missing statements and corrections for the program.

PROBLEM: Set up a procedure file to do the following:

1. Call UPDATE to place the program on a permanent file in program library format as deck AVEPROG (place data in a separate deck named AVEDATA). and compile the program.
2. Call UPDATE a second time in the procedure to replace the error lines with the corrected ones and insert the missing lines. Use your procedure from Lesson 6 to compile the program to be sure you have made the changes correctly. Using the supplied data values as input to the corrected program, compile and execute it.
3. Delete the program library's permanent file from the system and make sure the delete actually took place.

```

PROGRAM AVE (DATA,OUTPUT)
DIMENSION X(10)
SUM=0
DO 10 I=1,10
1 1 SUM=SUM+X(I)
    AVG=SUM/10.
    PRINT 10,AVG
1 FORMAT (1H1,"THE ANSWER IS",F10.3)
STOP
END

```

1. Program in error

1. 2. 3.

4. 5.

2. Data

7.,8.,9.,10.

```

DIMENSION X(10)
SUM=0
READ *,X
10 SUM=SUM+X(I)
    AVG=SUM/10.
    PRINT 1,AVG

```

3. Corrections

LESSON 9

MAGNETIC TAPES

LESSON 9

NON-ALLOCATABLE RESOURCES - MAGNETIC TAPE

Lesson Preview:

In this lesson, you will learn how to request one or more magnetic tapes and auxiliary disk packs. The various types of tapes available under NOS are examined, and examples of actual requests for tapes and packs are given.

Objectives:

After completing this lesson, the student will be able to:

- Describe the NOS tape and auxiliary disk pack commands and their operation
- Create and access magnetic tapes
- Access files on auxiliary disk packs

Projects: Exercise 9.1 (Magnetic Tape Review)

References:

- NOS Version 2 Reference Manual, Volume 3, Section 12 and Appendix G

RESOURCE MANAGEMENT

Resources other than use of the central processing unit, central memory and mass storage by the job can also consist of magnetic tapes and auxiliary disk packs. Magnetic tapes are requested by means of an ASSIGN, REQUEST, LABEL, or VSN command statements. Auxiliary disk packs are requested by means of the PN parameter on a permanent file control statement.

The number of resources a user may access are stated in the user validation file. For any number of resources greater than one, a RESOURC control statement must be issued by the job, stating the qualities and maximum quantities required at any point in the job. This enables the system to utilize deadlock prevention routines which roll out jobs that have a potential of exceeding the number of available units.

There are techniques for requesting magnetic tapes either automatically or manually. We suggest that automatic assignment procedures be used whenever possible, since it enhances the overall efficiency of the system.

AUTOMATIC TAPE ASSIGNMENT

NOS processes tape requests as follows:

1. Whenever a tape is mounted, NOS checks for labels. If the tape was labeled, NOS keeps a record of the one to six character Volume Serial Number (VSN) read from the VOL 1 label and the equipment on which the tape is mounted.
2. When a request is made for tape assignment, NOS compares the VSN associated with the file with the VSNs read from mounted tapes.

If a match is found, NOS automatically assigns the tape to the requesting job, provided a deadlock would not occur. If the tape is not mounted, NOS rolls out the job until a tape with the required VSN is mounted.

For a mounted, unlabeled tape, the operator enters a VSN command specifying the required VSN. NOS is then able to automatically assign the tape.

3. If no VSN was supplied when the request is made, NOS directs the operator to assign an available unit.
4. For an ASSIGN card, the method of assignment depends on the nn parameter. If nn is a device type (MT or NT), the operator must assign an available unit. If nn is the EST ordinal of a tape unit, NOS automatically assigns the specified unit. Specifying an EST ordinal is not normally advised nor permitted.

RESOURC CONTROL STATEMENT AND DEADLOCK PREVENTION

The discussion of the RESOURC control statement in section 7 of the NOS Reference Manual, Volume 3 can be studied to understand resource control from a system viewpoint and how it is used by the programmer making a request for more than one non-allocatable device.

MAGNETIC TAPE RECORDING MODES

- The recording mode of a tape refers to the physical manner in which the tape is written. You may write information on magnetic tapes in either binary or coded mode.
 - Binary mode
 - + Binary mode operations copy files with no character set conversion
 - + Any tape format can be used
 - Coded mode
 - + Coded mode operations involve data conversion from NOS-supported character sets to external character sets
 - + Applicable largely to S and L tape formats

TAPE DATA FORMAT TYPES

- Use F=format on tape request commands to specify tape data format.
- Foreign (F=F)
- Internal (F=I) (NOS default)
- Long block stranger (F=L)
- Stranger (F=S)
- System Internal (F=SI) (NOS/BE default)

Of the five types of tape available, the F=I (Internal) format is the default tape data for NOS. The tape being written in this format is assumed to be read in the same format. The other tape formats are provided for compatibility with tapes created by other operating systems.

The tape formats are discussed in Section 2 of the NOS Reference Manual, Volume 3, and can be referred to for details of each format. The data formats differ in PRU (block) size and in File mark indicators.

LABELED TAPES AND VSN

The VSN is the main field in providing a labeled tape library capability to an installation. The VSN field is either recorded physically on a VOL1 header label or assigned logically by the operator to an unlabeled tape. In any case, the VSN is the physical number that tells the tape librarian which reel is required for mounting onto a tape unit.

The VSN field in the VOL1 header label does not have to be the same as the number for operator's visual identification. It is not unusual for a tape to contain dissimilarities between internal (VOL1) and external (visible) VSNs. There is no confusion, because the VSN command allows equating two or more internal/external numbers.

The use of a labeled (or unlabeled) tape is most easily done by use of the LABEL control statement. There are other, more elaborate methods of creating labels which involve using macros and setting fields in the File Environment Tables in a program.

The label formats of the ANSI standard labels are in Appendix G of the NOS Reference Manual, Volume 3.

- Label types
 - ANSI standard labels
 - + Required ANSI label information
 - VOL1 (contains VSN of this volume)
 - HDR1
 - EOF1
 - EOVS (contains VSN of next volume if known from VSN list at time EOVS is written)
 - Non-standard labels
- To list the labels on a tape, use the LISTLB command.

RESOURC COMMAND

- Allows use of more than one auxiliary pack or one tape concurrently.
- See also UNLOAD and RETURN in lesson 4.

FORMAT:

RESOURC,nt=u.

Where:

nt Resource type (see Vol. 3, Section 7)

u Number of units a job will use concurrently

Examples:

/RESOURC, GE=3.

/RESOURC, DW = 2.

ASSIGN COMMAND

- Names tape unit or device type (MT or NT).
- Assigns local file to specified unit.
- Normally used only from system console jobs.

FORMAT:

ASSIGN,NN,LFN,VSN=vs_n,D=den ,F=format,LB=lb,PO=p₁,p₂,...,p_n.

Example:

/ASSIGN,NT,TAPE1,VSN=EMS10,D=GE,F=I,LB=KL,PO=W.

A 6250 cpi (D=GE) 9 track tape (NT) will be mounted for the user. Its VSN is EMS10. It will be mounted with write ring in (PO=W) and its label (LB=KL) will be read (W not specified). The local file name for the tape will be TAPE1.

/ASSIGN,MS,OUTPUT

Output will not appear at terminal.

BLANK COMMAND

- Writes ANSI standard labels after load point of a tape.
- Normally done only by site personnel.

FORMAT:

BLANK,VSN=vsn,MT or NT,D=den,OWNER=user,FA=fa,OFA=ofa,VA=va,U.

Example:

/BLANK,VSN=Ø,NT,D=PE.

VSN COMMAND

- Associates a file name with one or more volumes of tape
- Alternative to listing VSN's on ASSIGN, REQUEST, or LABEL commands.

FORMAT:

VSN,lfn1=vsn1,lfn2=vsn2,...,lfnx=vsnx.

Example:

/VSN,LOCAL1=VSNA,LOCAL2=VSNB,LOCAL3=VSNC.

Tapes with external VSNs LOCAL1, LOCAL2, and LOCAL3 are later mounted and NOS accepts them even though their VOL1 label contain VSNA, VSNB, VSNC, respectively.

/VSN,DUMP=DAY=MON=TUE=WED=THU=FRI=SAT=SUN.

Any tape with a VSN of DAY through SUN will satisfy the request (later). The operator will only see the first VSN(DAY).

LABEL COMMAND

- Associates local file name with a magnetic tape.
- Advantage over ASSIGN/REQUEST is that label status errors are reported separately from data format errors.
- Can rewrite and verify tape labels.
- Tapes can be unlabeled (LB=KU).

FORMAT:

LABEL,Ifn,VSN=vs_n,MT or NT,D=den,F=format,LB=lb,FA=fa,W or R,
PO=p₁,p₂,...,p_n.

Example:

/LABEL,MYTAPE,VSN=Ø,NT,D=GE,F=SI,LB=KU,FA=A.

REQUEST COMMAND

- Associates a local file name with a magnetic tape
- Request appears at the console
- Performs the same function as LABEL except that label status is not checked until a later read/write is first issued.

FORMAT:

REQUEST,Ifn,VSN=vs_n,MT or NT,D=den,F=format,LB=lb,PO=p₁,p₂,...,p_n.

Example:

/REQUEST,TAPE1,NT,F=I,LB=KL,PO=R.

LISTLB COMMAND

- Lists labels of an ANSI-labeled tape file.
- Especially useful for multifiled labeled tapes.

FORMAT:

LISTLB,Ifn,LO=ltype,L=out.

Example:

/LISTLB,TAPELBL,LO=A,L=LIST.

Also see tapes chapter in Volume 2 of NOS Reference Set for good examples of multifile labeled tape creation and usage.

TCOPY COMMAND

- A useful command to convert from one tape data format to another.

FORMAT:

See NOS Reference Set, Volume 3.

TDUMP COMMAND

- A useful command to see data format of a tape file.

Example:

/TDUMP,I=TAPE1.

EXERCISE 9.1

MAGNETIC TAPE REVIEW

1. (true/false) All users have access to magnetic tapes.
2. (true/false) A RESOURC control card is required if more than one tape is to be used simultaneously by one job.
3. (true/false) A tape must be Blank labeled to write a VOL1 label on it.
4. (true/false) A redundant request for a tape (one which is already assigned to the job) is ignored.
5. (true/false) Though it is valid to create multiple sets on both labeled and unlabeled tapes, only labeled tapes can be positioned automatically to a given file.
6. (true/false) A LABEL statement for an already mounted tape will provide automatic assignment if the VSN is specified for the file.
7. (true/false) Any tape user can issue an ASSIGN statement.
8. (true/false) Any tape user can write unlabeled tapes.
9. (true/false) Any tape user can write labeled tapes.
10. (true/false) Blank labels are ANSI labels with most fields reset to default values.

LESSON 10

LOADER

LESSON 10

LOADER

Lesson Preview:

In this lesson, the student will learn how programs get loaded into core, how external references are satisfied, how libraries are used, how absolutes can be created, and how field length is controlled.

Objectives:

After completing this lesson, the student will be able to:

- Describe the basic function of the loader
- Describe the loader commands
- Describe the types of loads that can be done

Projects: Review Loader Map: NOS Reference Set, Volume 2.

References:

- CYBER Loader User Guide (supplied)
- CYBER Loader Reference Manual (60429800)

DEFINITION

- A loader is a program which:
 1. Brings together various blocks of code (called relocatable binaries or common blocks).
 2. Organizes them according to some priority.
 3. Connects the blocks together by patching in addresses to external references.
- The results of this process is one continuous block of code called absolute binary (or core image).

GLOBAL vs LOCAL LIBRARIES

GLOBAL library sets are defined by a LIBRARY card. The set is limited to two users and two system libraries, one user and 13 system or no user and 24 system libraries. No limit exists for a local library set. It is defined by LDSET (LIB=).

The GLOBAL library set remains in effect until redefined by another LIBRARY card. To empty the Global Library Set, use LIBRARY. LIBRARY may not appear in a load sequence.

Example:

| | |
|---------------------|-----------------------------------|
| JOB. | |
| USER (,) | |
| ATTACH,LIB1. | |
| ATTACH,LIB2. | |
| LIBRARY (LIB2) | The Global Library = LIB 2 |
| LIBRARY (LIB1,LIB2) | The Global Library = LIB 1, LIB 2 |
| LIBRARY. | The Global Library is empty. |

The Local Library Set remains in effect only until the end of the Load Sequence and affects only those routines loaded after the definition of the library. LDSET(LIB=) commands within a load sequence add to the LOCAL library set; they do not redefine it.

Example:

```
.PROC,LOCAL
ATTACH, LIB1.
ATTACH, LIB2.
GET(LGO)
LDSET(LIB=LIB1)
LOAD(LGO)
LOAD(BIN)
EXECUTE
LOAD(LGO)
LDSET(LIB=LIB1)
LOAD (BINA)

LDSET(LIB=LIB2)
LOAD(BINB)

NOGO.
```

Local Library = LIB1

Ends first load sequence

No Local Library Set; LIB1 will not be searched for Externals from LGO.

The Loader will search LIB1 for Externals from BINA.

Local Library = LIB1, LIB2

The Loader will search LIB1, LIB2 for Externals from BINB.

Ends second load sequence.

EXTERNAL REFERENCE

An External Reference is a reference to a symbol not defined in the program which makes the reference. For FORTRAN, an external reference is constructed whenever the programmer uses a CALL statement. For COBOL users, an external reference is constructed whenever the programmer uses a CALL or ENTER statement. For COMPASS, an external reference is set up whenever the programmer uses an EXT statement or the equivalent. For each external reference, the LOADER will attempt to find a program that defines that symbol as an ENTRY POINT.

Example: FORTRAN Source file (SORTER)

| | |
|--------------------|---|
| PROGRAM MAIN | The FORTRAN Compiler sets up an external |
| + | reference to SORTALL. The Loader will look |
| + | for and find the entry point SORTALL in |
| + | subroutine SORTALL and will link the two |
| CALL SORTALL | together. |
| + | |
| + | |
| + | |
| + | |
| + | |
| + | |
| STOP | |
| END | |
| | |
| SUBROUTINE SORTALL | The FORTRAN compiler sets up an Entry Point |
| + | called SORTALL. |
| + | |
| + | Question: What is an Entry Point? |
| | |
| END | |

SATISFYING EXTERNALS - PRIORITY

1. Local file, i.e., the binaries loaded through direct load calls:

Example:

```
LOAD (BINONE)      These binaries are the LOCAL FILE.  
LOAD (BINTWO)
```

2. GLOBAL Library Set (defined by LIBRARY (library 1, . . .)).
3. Local Library Set (defined by LDSET (LIB=library 1, . . .)).
4. Default Library Set

Example:

```
.PROC,PRIORITY.  
ATTACH,LIBONE.      Contains routines A, B, C, D, E, F, G.  
ATTACH,LIBTWO.      Contains routines A, B, C, D, E, F, G.  
ATTACH,LIBTHR.      Contains routines A, B, C, D, E, F, G.  
ATTACH,LIBFOUR.     Contains routines A, B, C, D, E, F, G.  
ATTACH, BIN A.       Calls A.  
ATTACH, BIN B.       Calls B.  
ATTACH, BIN C.       Calls C.  
ATTACH, BIN D.       Calls D.  
LDSET (LIB=LIBONE)  
BIN A.  
LIBRARY (LIBTWO)  
LDSET (LIB=LIBONE)  
BIN B.  
LIBRARY (LIBTHR,LIBFOUR)  
BIN C.  
LIBRARY.  
LOAD(BINA)  
SATISFY (LIBFOUR)  
LOAD(BINB)  
LDSET(LIB=LIBONE)  
LOAD(BINC)  
LOAD(BIND)  
EXECUTE
```

Question: How are Externals satisfied during each load sequence?

EXTERNALS WHICH CALL EXTERNALS

- Suppose a programmer has three libraries:

LIB1 Contains A, B, C, D, E, ONE, TWO, JANE, BETTY

LIB2 Contains ONE, TWO, THREE, FOUR, FIVE, JANE, BETTY

LIB3 Contains JANE, BETTY, MARY, A, B

- Suppose:

A calls JANE, FOUR, and MARY

ONE calls TWO, E and B

MARY calls B, D and ONE

- Assume the following procedure file:

```
.PROC,EXTERN.  
ATTACH, LIB1,LIB2,LIB3.  
FTN5,I=BELOW.  
LDSET (LIB=LIB1/LIB2/LIB3)  
LGO.  
.DATA,BELOW.  
PROGRAM MAIN (OUTPUT)  
.  
.  
.  
CALL A  
.  
.  
.  
END  
(EOR)
```

Question: From which libraries are the above routines satisfied?

COMMON BLOCKS

A program or subroutine can also make a reference to data within a Common Block. A COMMON BLOCK is a grouping of data that can be referenced by more than one routine.

COMMON BLOCKS are divided into two types:

BLANK COMMON - This is a block of data that is placed at the very end of a user's field length. The first word of this block is equated with the last word of all executable code, plus one. It cannot be preset with any data. FORTRAN programmers define Blank Common with a COMMON A(50) statement.

LABELED COMMON - An example of Labeled Common in FORTRAN is:

```
COMMON/ABC/A(50).
```

Labeled Common Blocks are placed in front of the first routine which defines them. Once defined by one routine, they cannot be redefined to a larger or smaller size by another routine. Data in these blocks can be Preset using LDSET (PRESET=option).

Example:

```
DATA A/50*0.0/
```

COBOL allows a programmer to define one label common block called CCOMMON. He does this by defining a COMMON-STORAGE SECTION.

COMMON BLOCK EXAMPLE

Example: COMMON BLOCKS USED BY FORTRAN AND COBOL

| FORTRAN FILE | COBOL |
|--------------------------|---|
| .DATA,MAIN. | .DATA,MAIN. |
| PROGRAM MAIN | IDENTIFICATION DIVISION |
| : | PROGRAM-ID. MAIN. |
| COMMON/CCOMMON/DATA(3) | . |
| : | . |
| COMMON/A/A (5) | . |
| . | COMMON-STORAGE SECTION. |
| . | 77 DATA-1 PICTURE IS 9999V99 USAGE IS COMP-1. |
| CALL=SORTALL | 77 DATA-2 PICTURE IS 9999V99 USAGE IS COMP-1. |
| C=DATA(1) | 77 DATA-3 PICTURE IS 9999V99 USAGE IS COMP-1. |
| . | : |
| . | PROCEDURE DIVISION. |
| . | INIT SECTION. |
| STOP | INIT-PARA. |
| END | . |
| | . |
| | ENTER SORTALL |
| | MOVE DATA-1 TO C. |
| | . |
| | . |
| | STOP RUN. |
| .DATA,SORTALL. | |
| SUBROUTINE SORTALL | |
| COMMON/CCCOMMON/DATA (3) | |
| COMMON/A/A (5) | |
| COMMON/SORTALL/SORT (5) | |
| . | |
| . | |
| . | |
| DATA (1) = 5.0 | |
| . | |
| . | |
| . | |
| RETURN | |
| END | |

ABSOLUTE BINARY

An ABSOLUTE BINARY is a collection of one or more relocatable blocks, organized according to some priority scheme, where references to externals have been "satisfied" wherever possible. EXTERNAL REFERENCES are SATISFIED whenever the loader can find an ENTRY point name to match the EXTERNAL REFERENCE name. Otherwise, EXTERNAL REFERENCES are UNSATISFIED.

Example: Unsatisfied External

```
FTN5(I=MAIN,B=BINMAIN)
SAVE,BINMAIN.
BINMAIN.
```

When loading BINMAIN, the LOADER will detect that there is an external reference to SORTALL. It will look for SORTALL in BINMAIN and in the system library. Since SORTALL is not in either place, the external reference remains unsatisfied.

Example: Satisfied Externals

```
FTN5, I=SORTALL, B=BINSORT.
SAVE,BINSORT.
GET,BINMAIN.
LOAD(BINSORT)
BINMAIN.
```

Question: Would this sequence work? If not, how would you fix it?

```
GET,BINMAIN.
GET,BINSORT.
BINMAIN.
```

```
BINMAIN calls SORTALL
BINSORT contains SORTALL
```

LOAD PRIORITY - LOADER MAP

- Blocks are loaded according to the following priority scheme:
 1. Labeled Common blocks in the order that they are defined.
 2. Program blocks.
 3. Blank Common.
- A LOADER MAP tells a programmer how the LOADER organized the blocks which were presented to it.

Example:

```
PROGRAM MAIN  
COMMON/DATA/DATA(5)  
COMMON A(6)  
.  
.  
.  
END
```

```
SUBROUTINE SORTALL  
COMMON/DATA/DATA(4)  
COMMON B(5),C(2)  
COMMON/SORTALL/SORT(5)  
.  
.  
.  
RETURN  
END
```

RESULTANT LOADER MAP:

| BLOCK: | ADDRESS: | LENGTH: |
|-----------|----------|---------|
| /DATA/ | 111 | 5 |
| MAIN | 116 | 100 |
| /SORTALL/ | 216 | 5 |
| SORTALL | 223 | 100 |
| / / | 323 | 7 |

FWA OF THE LOAD 111
LWA OF THE LOAD 332

LOADER SEQUENCE

There are several words reserved for usage by the loader: LOAD, LIBLOAD, SLOAD, CAPSULE, EXECUTE, NOGO, SATISFY, LDSET, SEGLOAD. Once a loader word is used in a job control sequence, the remaining job control cards must be loader commands until the loader sequence is terminated by a NOGO, EXECUTE, or program name call.

QUESTION: Are the following load sequences valid? If not, correct them.

a. JOB.
 USER(,)
 GET,BINMAIN.
 LOAD(BINMAIN)
 GET,BINSORT.
 LOAD(BINSORT)
 EXECUTE

b. JOB.
 USER(,)
 ATTACH,BINMAIN.
 BINMAIN.
 EXECUTE

c. JOB.
 USER(,)
 GET,BINMAIN.
 GET,BINSORT.
 LOAD(BINMAIN)
 LOAD(BINSORT)
 ABS(ABSMAN)
 ABSMAN.

d. JOB.
 USER(,)
 ATTACH,BINMAIN.
 ATTACH,BINMAIN.
 BINMAIN.
 LOAD(BINMAIN)
 EXECUTE
 LOAD(BINSORT)
 LOAD(BINMAIN)
 EXECUTE(BINMAIN)

RELOCATABLE BINARY

Routines MAIN and SORTALL are separate logical entities or blocks. They are called PROGRAM BLOCKS or RELOCATABLE BINARIES. After they are assembled, they can be saved as separate binary code entities. Each block of relocatable binary contains a short table at the beginning of the block which contains the name of the routine and a list of one or more entry points to that block. Also included are the addresses of all the instructions in the binary code which make external references. The LOADER can then find these instructions and insert addresses of appropriate entry points. This is called Satisfying Externals.

If any externals other than a weak external (used by capsules) remains unsatisfied, a non-fatal error results. Any address field containing an unsatisfied external is filled with $\text{addr} + 400000_8$, where addr is the address of the reference. This will cause an OUT OF RANGE reference during execution.

Example: SAVING AND USING RELOCATABLE BINARIES ON PERMANENT FILES

```
JOB1.  
USER ( , )  
FTN5, B=BINMAIN.  
SAVE, BINMAIN.  
FTN5, B=BINSORT.  
SAVE, BINSORT.  
  
JOB2.  
USER ( , )  
GET, BINMAIN, BINSORT.  
LOAD, BINMAIN, BINSORT.  
EXECUTE
```

Saves on compile time.

ALTERNATE ENTRY POINT EXAMPLE

Example: SPECIFYING AN ENTRY POINT - RELOCATABLE BINARY

```
.PROC,ENTER.  
FTN5,I=MYPROGS.  
LOAD,LGO.  
EXECUTE,ONE.  
LOAD,LGO.  
EXECUTE,LGO.  
LGO.  
.DATA,MYPROGS.  
    PROGRAM ONE  
    .  
    .  
    END  
    PROGRAM TWO  
    .  
    .  
    PROGRAM THREE  
    .  
    .  
    END  
    SUBROUTINE A  
    .  
    .  
    END
```

QUESTION: Where does EXECUTION begin with simple LGO?

Example: SPECIFYING AN ENTRY POINT - ABSOLUTE BINARY

```
.PROC,ABSBIN.  
FTN5,I=MYPROGS,L=LIST.      Same Programs as above Example  
LOAD (LGO)  
NOGO (ABS,ONE,TWO,THREE)  
LOAD (ABS)  
EXECUTE, ONE  
LOAD (ABS)  
EXECUTE, TWO.  
ABS.
```

QUESTION: Where does Execution begin with a simple ABS?

)

LOADER COMMANDS

EXECUTE COMMAND

- Causes completion of the LOAD sequence
- Execution of the loaded programs occurs immediately
- Can optionally specify entry points
- Can optionally specify execution parameters to be passed to the loaded program

FORMAT:

EXECUTE (entry point,parameter 1,parameter 2...)

It is possible to pass information to a binary program through EXECUTE parameters. e.g., EXECUTE,ONE,TAPE1,TAPE2.

These parameters are stored in the job's field length at locations RA+2, RA+3 up through RA+53. In the example, before program one begins to execute, the word TAPE 1 in display code would be placed at RA+2 and TAPE2 would be placed in RA+3. Both words would be left justified, zero filled with a special code in the lower six bits of the word to indicate what the separator is, i.e., a comma, equal sign, period.

The program can make use of this information. FORTRAN programs make special use of this information. Parameters passed in this fashion redefine files called by the FORTRAN program.

EXECUTE COMMAND EXAMPLE

Example:

```
.PROC,FORTRAN.  
NOEXIT.  
GET,TAPE1,TAPE2,TAPE3.  
GET,FILEA,FILEB.  
FTN5,I=#DATA..  
LOAD(LGO)  
EXECUTE,,INPUT,OUTPUT,FILEA.  
LGO,INPUT,OUTFILE,FILEB.  
REWIND,OUTFILE.  
COPY,OUTFILE,OUTPUT.  
LGO.
```

QUESTION: What happens at each execution?

```
.DATA  
  PROGRAM ONE(TAPE1,TAPE2,TAPE3)  
  READ (1,000)  
  READ (3,3000)  
  WRITE (2,2000)  
  STOP  
  END
```

(EOR)

Ensure no blank lines after last FORTRAN "END" statement.

COPYL COMMAND

COPYL is used to update a sequential binary file. It is very helpful in reducing compile time when developing routines that contain several subroutines and user libraries are not being used.

Example:

```
.PROC,NOULIB.  
GET,OLDBIN,IN.  
FTN5(I=IN).  
COPYL,OLDBIN,LGO,NEWBIN.  
NEWBIN.
```

OLDBIN contains subroutines A, B, C, D. LGO contains subroutine C. NEWBIN contains A,B,D from OLDBIN and C from LGO.

Example:

```
.PROC,UPDATER.  
GET,OLDPL  
UPDATE.  
FTN5 (I)  
GET,OLDBIN.  
COPYL,OLDBIN,LGO,NEW.  
NEW.  
REPLACE(NEW=OLDBIN)
```

OLDPL contains program main, subroutines A, B, C, D. Default Update. Only subroutine C was changed and compiled.

Only subroutine C is on LGO.
Test to see if subroutine C and other routines are still working.

LDSET COMMAND

- Provides user control of a variety of load operations (e.g., MAP,PRESET)
- Apply for current (local) load sequence only

Besides a GLOBAL library set, the Common Loader also recognizes a LOCAL library set. The local library is defined by an LDSET (LIB=) card. It is defined only for the load sequence. It will be searched first.

Example:

| | |
|--------------------|--|
| .PROC,LOADSET. | |
| ATTACH,LIBSUB1. | Contains routines A, B, C, D, E. |
| ATTACH,LIBSUB2. | Contains routines A, B, C, G, H, I. |
| FTN5. | Program calls A and B. |
| | Routines A, B, C are slightly different on each library. |
| LDSET(LIB=LIBSUB1) | Only LIBSUB1 is searched. |
| LGO. | |
| LDSET(LIB=LIBSUB2) | Only LIBSUB2 is searched. |
| LGO. | |
| LGO. | External references to A and B are left unsatisfied. |

LIBLOAD COMMAND

- Specifies that the loader is to load one or more programs from a particular library

FORMAT:

LIBLOAD(filename,entry point1,entry point 2 ...)

Example:

```
.PROC,LOADAB,I.  
FTN5(#I=I).  
GET,LIBFILE.  
LIBLOAD(LIBFILE,A,B)  
LGO.
```

PROGRAM ONE (Which calls A and B) LIBFILE is a library file created by the user on a previous run.

End of the load sequence.

LIBRARY COMMAND

- Specifies a set of global libraries to be searched for externals and name call statements

FORMAT:

LIBRARY(filename 1, filename 2)

Example:

```
.PROC,GLOBAL  
GET,ONE,TWO.  
ATTACH,LIBFIL1.  
ATTACH, LIBFIL2.  
LIBRARY(LIBFIL1,LIBFIL2)  
FTN5,I=ONE.  
LGO.
```

```
REWIND,*.  
FTN5,I=TWO.  
LGO.
```

LIBFIL1 contains routines A, B, C, D, E, F.
LIBFIL 2 contains routines G, H, I, J, K, L, M.

Program One (which calls A, F, G, L).
The loader will search LIBFIL1 and LIBFIL2 to satisfy the externals. A and F will be loaded from LIBFIL1 and G and L will be loaded from LIBFIL2.

Note that LIBRARY will still be available to this load sequence.

LOAD COMMAND

- Specifies files containing object programs to be loaded

FORMAT:

LOAD(lfn₁,...lfn_n)

Example:

.PROC,LOADLFN,FTNIN,COBIN.

FTN5I=FTNIN.

COBOL5,I=COBIN.

LOAD (LGO)

EXECUTE

BKSP (LGO,2)

LOAD (LGO/NR)

EXECUTE

This produces three logical records on the file LGO.

The entire file is loaded.

Only the COBOL routine is loaded and executed.

MODE COMMAND

- Defines error conditions that cause the system to exit from normal processing.
- System sets appropriate error flag, exits from normal processing, and performs the necessary error processing when specified error occurs.
- If an error occurs that was not selected, the system ignores the error and continues normal processing.
- Should not be needed for a fully debugged program.

FORMAT:

MODE,m,n.

Where:

- m CPU error exit mode 0-17.
- n Included for compatibility with earlier versions of NOS; ignored on Version 2.x.

SATISFY COMMAND

- Causes External References to be satisfied from specified library files for all routines loaded up to that point.

FORMAT:

SATISFY (library 1, library 2, . . .)

Example:

.PROC,SATED.

ATTACH, BINONE.

Calls routines A and B.

ATTACH, BINTWO.

Calls routines C and D.

ATTACH, LIBFIL1.

Contains routines A, B, C, D.

ATTACH, LIBFIL2.

Contains slightly different routines A, B, C, D.

LOAD (BINONE)

SATISFY (LIBFIL2)

QUESTION: From which libraries are routines A, B, C, D loaded?

*TRY AGAIN.

LOAD (BINTWO)

SATISFY (LIBFIL1)

EXECUTE

SLOAD COMMAND

- Specifies selected programs to be loaded from a local file

FORMAT:

SLOAD(filename, module name 1, module name 2 . . .)

Example:

```
.PROC,SELECT.
FTN5,I=IN.
SLOAD(LGO,ONE,A,B)
EXECUTE
SLOAD(LGO,THREE,C,B)
EXECUTE
.DATA,IN.
PROGRAM ONE
PROGRAM TWO
PROGRAM THREE
SUBROUTINE A
SUBROUTINE B
SUBROUTINE C
```

This will produce six records on LGO.

```

PROGRAM ONE
PROGRAM TWO
PROGRAM THREE
SUBROUTINE A
SUBROUTINE B
SUBROUTINE C
```

