HONEYWELL

# CP-6 APPLICATION PROGRAMMER HANDBOOK

SOFTWARE

# CONTROL PROGRAM-SIX (CP-6)
# APPLICATION
# PROGRAMMER HANDBOOK

SUBJECT

A Handbook for the Application Programmer

SOFTWARE SUPPORTED

Release B03

ORDER NUMBER

CE55-01

January 1984

Honeywell

# Preface

CE55, the Application Programmer Handbook for the CP-6 system, is a companion manual to CE40, the Programmer Reference Manual. It is intended for use by the applications programmer who is conversant with the CP-6 System language references and language guides, as well as with the Programmer Reference manual.

Documentation would like to express its appreciation to Marketing National Software Support/CP-6 for making selected documents available for inclusion in this handbook.

The Los Angeles Development Center (LADC) of Honeywell Information Systems has developed Computer Aided Publications (CAP). CAP is an advanced text processing system providing automatic table of contents, automatic indexing, format control, automatic output of camera-ready masters, and other features. This manual is a product of CP-6 CAP.

Readers of this document may report errors or suggest changes through a STAR on the CP-6 STARLOG system. Prompt response is made to any STAR against a CP-6 manual, and changes will be incorporated into subsequent releases and/or revisions of the manual.

The information in this publication is believed to be accurate in all respects. Honeywell Information Systems cannot assume responsibility for any consequences resulting from unauthorized use thereof. The information contained herein is subject to change. New editions of this publication may be issued to incorporate such changes.

 File No.: 1W13 CE55-01

# Contents


Page

MODULE 1-0 Section 1 - Obtaining Information . . . . . 1

MODULE 1-1 HELP. . . . . 2

MODULE 1-2 Displays. . . . . 6
CHECK and NOTIFY - Comments. . . . . 7

MODULE 2-0 Section 2 - Controlling Terminals . . . . . 8

MODULE 2-1 Setting Terminal Profiles . . . . . 9

MODULE 2-2 Cursor/Printhead Positioning. . . . . 11
Moving the Cursor. . . . . 11
ESCAPE N . . . . . 11
ESCAPE V - Moving to Character 'N' . . . . . 12
Tab Stops. . . . . 12
ESCAPE I . . . . . 13
CONTROL R - Forward Positioning. . . . . 13
CONTROL H - Backspacing. . . . . 13
ESCAPE <RET> - Position to Beginning of Record . . . . . 14

MODULE 2-3 Cancelling and Recalling Input. . . . . 15
ESCAPE X - Deleting Current Input Line . . . . . 15
ESCAPE K - Deleting From Current Edit Point. . . . . 16
ESCAPE A - Setting Pagination Mode . . . . . 16
ESCAPE R - Retyping the Current Input Line . . . . . 17
ESCAPE D - Retrieving the Last Input Line. . . . . 17

MODULE 2-4 Inserting/Replacing/Overstriking. . . . . 18
ESCAPE J, ESCAPE <CR>, ESCAPE O, ESCAPE M. . . . . 18
More About ESCAPE J. . . . . 19

MODULE 2-5 Platen Control. . . . . 20

MODULE 2-6 IMP . . . . . 21
Defining New Escape Sequences. . . . . 21
Setting Up Special Function Keys . . . . . 22
Resetting Special Function Keys. . . . . 23
Redefining the Keyboard. . . . . 24

MODULE 2-7 Dribble Files . . . . . 26
Displaying DRIBBLE Files at Your Terminal. . . . . 27

MODULE 3-0 Section 3 - Editing and Manipulating Files. . . . . 28

MODULE 3-1 Creating New Files. . . . . 29

MODULE 3-2 EDIT Command Files. . . . . 31

MODULE 3-3 Selecting Record Ranges . . . . . 32

MODULE 3-4 Moving, Merging, and Copying Files. . . . . 35

# Contents (cont)


Page

MODULE 3-5 Conditional Execution in EDIT . . . . . 39
EDIT Command Files . . . . . 40

MODULE 3-6 Listing and Reviewing Files . . . . . 43
Listing File Attributes. . . . . 43
REVIEW Command (PCL) . . . . . 45
Recovering Files . . . . . 46

MODULE 3-7 Changing File Organization. . . . . 47

MODULE 3-8 Changing File Access Attributes . . . . . 49

MODULE 3-9 Selecting Files . . . . . 51

MODULE 3-10 Wildcarding. . . . . 52
Abbreviating Account References Through Wildcarding. . . . . 53

MODULE 3-11 Maintaining File Accounts. . . . . 54
File Types . . . . . 54
Star Files . . . . . 56
Running Out Of Space . . . . . 57
Conserving Disk Space. . . . . 57
Object Units . . . . . 57
Source Programs. . . . . 57
Run Units. . . . . 58
Other Files. . . . . 58
Keyed Data Files . . . . . 58
Using Files in Other Accounts. . . . . 59

MODULE 3-12 Printing Files on the Lineprinter. . . . . 60

MODULE 4-0 Section 4 - Creating and Running Programs . . . . . 62

MODULE 4-1 IBEX Programming. . . . . 63
IBEX Programming Conventions . . . . . 63
Executing Programs . . . . . 64
Invoking Language Processors . . . . . 65
Interrupt Processing . . . . . 65
Allocating Resources and Establishing Service Limits . . . . . 66
Execute Files. . . . . 67
Batch Jobs . . . . . 67
Command Files. . . . . 69
Data Replacement . . . . . 69
IBEX Command Labels. . . . . 71
Command File Logic - Conditional Execution . . . . . 71
STEPCC . . . . . 72
Program Exit Method. . . . . 72
IBEX Expressions . . . . . 73
Expression Component - Precedence of IBEX Operators. . . . . 74
Preprocessing of Commands. . . . . 75
BATCH/XEQ Substitution . . . . . 75
Preprocessor Substitution. . . . . 75
Examples of IBEX Command Files . . . . . 76
Command File to Read Tape. . . . . 76
Command File That Interrogates User. . . . . 77
Setup File . . . . . 78

MODULE 4-2 Sample Session. . . . . 79

# Contents (cont)


Page

MODULE 4-3 FPL: Compiling, Linking, Debugging . . . . . 88

MODULE 4-4 Using LINK Overlays . . . . . 95
Resolving Differences and Ambiguities. . . . . 97
Program Trees. . . . . 97
Specifying an Overlay. . . . . 98
Using LINK to Build a Run Unit with Overlays . . . . . 102
Using LINK's PROMOTE_BLANK and PROMOTE_LABEL Options . . . . . 103
Using HELP . . . . . 104

MODULE 5-0 Section 5 - Practical Applications. . . . . 105

MODULE 5-1 Creating Sorted Indexes on CP-6 . . . . . 106

MODULE 5-2 How to Perform Compilations in Batch Mode . . . . . 109

MODULE 5-3 A CP-6 System Program with its Own 'HELP' . . . . . 111

MODULE 6-0 Section 6 - Use of Magnetic Tape in the CP-6 System . . . . 113

MODULE 6-1 Rules for Tape Usage. . . . . 114
Tape Management. . . . . 114
PCL Commands . . . . . 114
Basic Types of Magnetic Tapes. . . . . 115
ANS Levels of Protection . . . . . 116
Free and Managed Free Tapes. . . . . 117
Free Tapes . . . . . 117
Managed Free Tapes . . . . . 117
Comparisons - Free and Managed Free. . . . . 118
ANS Labeled Format . . . . . 118
EBCDIC Labeled Format. . . . . 119
Mixing of CP-6, ANS, and EBCDIC Labeled. . . . . 119
Tape Fids. . . . . 119
ANS Tape Fids. . . . . 119
Tape Fids May Need Quotes. . . . . 120
Multi-reel Tape Fids . . . . . 120
Acquiring Tape Drives. . . . . 120
MOUNT Command. . . . . 121
RESOURCE, ORESOURCE, and ACQUIRE Commands. . . . . 121
Default Tape Drive Assignments . . . . . 121
Logical Density. . . . . 122
Single-density and Mixed-density Systems . . . . . 122
Examples of Online Use . . . . . 123
File Sequence Numbers. . . . . 125
Hardware Limitations . . . . . 127
Minimum Record Size. . . . . 127
Lost Data. . . . . 127
Number of Bytes Stored on Tape . . . . . 127
File Management Buffers. . . . . 128
DCBs . . . . . 128

MODULE 6-2 Tape Commands, Options, and Calls . . . . . 129
Introduction . . . . . 129
PCL Tape Control Commands. . . . . 129
PCL and !SET Output Options. . . . . 130
!SET Command Options . . . . . 130
PCL Input Options. . . . . 132
PCL Output Options . . . . . 133
PCL vs SET . . . . . 135
Monitor Service Calls. . . . . 135
M$DCB. . . . . 136

# Contents (cont)


Page

M$OPEN - OPEN DCB. . . . . 139
M$CLOSE - CLOSE DCB. . . . . 140
M$CVOL - CLOSE VOLUME. . . . . 141
M$READ - READ RECORD . . . . . 142
M$WRITE - WRITE RECORD . . . . . 142
M$PRECORD - POSITION TO RECORD . . . . . 142
M$REW - REWIND . . . . . 143
M$PFIL - POSITION FILE . . . . . 143
M$WEOF - WRITE END-OF-FILE . . . . . 144
M$REM - REMOVE OR RELEASE VOLUME . . . . . 144
M$TRUNC - TRUNCATE BUFFERS . . . . . 144

MODULE 6-3 CP-6 System Tape Processing . . . . . 145
Automatic Volume Recognition . . . . . 145
Tape Resource Management . . . . . 146
CP-6 Tape File Management. . . . . 147
Anomalies and Errors . . . . . 148

MODULE 6-4 How to Make Tapes That Other Machines Can Read. . . . . 149
Tape Types in CP-V and CP-6. . . . . 149
Creating Tapes . . . . . 150
ANS Tape Options at a Glance . . . . . 151
ANS Tape Options - Complete Description. . . . . 151
Block Sizes for ANS Tapes. . . . . 152

MODULE 6-5 Converting Imported Tapes For CP-6 Use. . . . . 153
Introduction . . . . . 153
How to Make ANS Tapes on CP-V for CP-6 Use . . . . . 153
Making ANS Tapes on Multics for CP-6 Use . . . . . 154
ANS Tapes Made On Other Systems. . . . . 155

MODULE 6-6 How to Copy Tapes . . . . . 156
Making Tape Copies . . . . . 156
Making Copies of Labeled Tapes . . . . . 156
Copying Managed Free Tapes . . . . . 157
Copying Free Tapes . . . . . 157
Extension of Tape Files. . . . . 157
ASCII TO EBCDIC Conversion . . . . . 158
Making a Tape Copy for Export. . . . . 158
Copying Binary Data to Tape. . . . . 158
Copying ANS Tapes Made on Other Systems. . . . . 159
Block Size . . . . . 159
Blocked Tapes and FPOOL Buffers. . . . . 159
Block Size Versus Efficiency . . . . . 159

MODULE 6-7 Multi-reel Tapes. . . . . 161
Creating Multi-volume Tape Sets. . . . . 161
Using Volume Sets. . . . . 162

MODULE 6-8 Tape Formats. . . . . 164
CP-6 Tape Formats. . . . . 164
User Labels. . . . . 165
Additional ANS Labels. . . . . 166
ANS Labeled Structure. . . . . 166

# Contents (cont)


Page

MODULE 6-9 TAPE ERRORS . . . . . 168
Introduction . . . . . 168
I/O Errors . . . . . 168
Free Tape Errors . . . . . 169
Volume End Errors. . . . . 170
Volume Change Errors . . . . . 171
Data Record Structure Errors . . . . . 171
Break Error Messages . . . . . 173
Operator-generated Errors. . . . . 173
Errors and Protection Level. . . . . 174
Access Limitation Errors . . . . . 175
Tape Type and Tape Format Errors . . . . . 175
Miscellaneous Errors . . . . . 176
Tape Error Message Summary . . . . . 177

Index. . . . . i-1


TABLES


Types of Tape . . . . . 115
ANS Levels of Protection. . . . . 116
PCL Tape Commands . . . . . 129
!SET Options (Tape) . . . . . 131
PCL Input Tape Options. . . . . 132
PCL Output Tape Options . . . . . 134
M$DCB and M$OPEN Tape Options . . . . . 136
Additional M$OPEN Tape Options. . . . . 140
Primary ANS Tape Labels . . . . . 165
User Labels . . . . . 166
Additional ANS Labels . . . . . 166
I/O Errors. . . . . 168
Free Tape Errors. . . . . 169
Volume End Errors . . . . . 170
Volume Change Error Messages. . . . . 171
Data Record Structure Errors. . . . . 172
Break Error Messages. . . . . 173
Operator-generated Errors . . . . . 174
Protection Level Errors . . . . . 174
Access Limitation Errors. . . . . 175
Tape Type/Format Errors . . . . . 175
Miscellaneous Tape Error Messages . . . . . 176
Tape Error Message Summary. . . . . 177


FIGURES


Tape Structure. . . . . 167

# About This Manual

This is a user's handbook and is intended to be function oriented, i.e., to show "how to do" various tasks that represent the capabilities of the CP-6 system. It should be used in conjunction with the Programmer Reference Manual, CE40, which is organized as an encyclopedic manual containing all of the commands and syntaxes for various CP-6 processors, along with detailed descriptions of the CP-6 file system and terminal editing features.

Users will find the CP-6 HELP facility useful in providing on-line access to reference data such as command syntax.

This manual is organized into six sections:

Section 1 -- Obtaining Information
Section 2 -- Controlling Terminals
Section 3 -- Editing and Manipulating Files
Section 4 -- Creating and Running Programs
Section 5 -- Practical Applications
Section 6 -- Use of Magnetic Tape in the CP-6 System

Module 4-4 and Section 6 are new additions to this revision of the manual.

The CP-6 Application Programmer Handbook is conceived as a dynamic document which can grow and be modified in future editions to reflect user needs and input.

# Notation Conventions

Notation conventions used in command specifications and examples are listed below.

| Convention | Meaning |
|---|---|
| CAPITAL LETTERS | Capital letters must be entered as shown for input, and will be printed as shown in output. |
| lowercase letters | Lowercase letters identify an element that must be replaced with a user-selected value.<br><br>AP i could be entered as AP 2. |
| Special Characters | Numbers that appear on the line (i.e., not subscripts), special symbols, and underlines appear as shown in output messages and must be entered as shown when input.<br><br>#xxx could be entered as #011. |
| Brackets | An element inside brackets is optional. Several elements separated by an "or" bar inside a pair of brackets means that the user may select any one or none of those elements.<br><br>[key] - means a key value may be entered.<br><br>When enclosing keywords, brackets signify that all or part of the bracketed portion may be entered.<br><br>K[EY] can appear as K, KE, or KEY. |
| Braces | Elements placed inside a pair of braces identify a required choice. These are always used with the Or bar (\|). |

| Convention | Meaning |
|---|---|
| Or Bar | |
| | The Or bar also separates elements in a required choice.<br><br>{A\|id} - means that either the letter A or the value of id must be entered. |
| Ellipsis | |
| | The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.<br><br>option[,option]... - means that one or more options may be entered, with a comma inserted between each variable. |
| Careted Letters | |
| | Letters inside carets indicate the keys on a physical terminal device.<br><br><ESC> <BS> - means press the escape key and then the backspace key. |

## Related Manuals

The following manuals are available to users of the CP-6 System.

| ORDER NUMBER | TITLE |
|---|---|
| CE26 | CP-6 Concepts and Facilities |
| CE28 | CP-6 SORT/MERGE Reference Manual |
| CE29 | CP-6 COBOL Reference Manual |
| CE30 | CP-6 IDP Reference Manual |
| CE31 | CP-6 FORTRAN Reference Manual |
| CE32 | CP-6 BASIC Reference Manual |
| CE33 | CP-6 Monitor Services Reference Manual |
| CE34 | CP-6 Operations Reference Manual |
| CE35 | CP-6 I-D-S/II Reference Manual |
| CE36 | CP-6 I-D-S/II DBA Reference Manual |
| CE37 | CP-6 RPG/II Reference Manual |
| CE38 | CP-6 APL Reference Manual |
| CE39 | CP-6 DELTA Reference Manual |
| CE40 | CP-6 Programmer Reference Manual |
| CE41 | CP-6 System Support Reference Manual |
| CE42 | CP-6 Pocket Guide |
| CE44 | CP-6 PL-6 Reference Manual |
| CE45 | CP-6 Primer |
| CE46 | CP-6 COBOL Programmer Guide |
| CE47 | CP-6 FORTRAN Programmer Guide |
| CE48 | CP-6 Text Processing Reference Manual |
| CE49 | CP-6 TP Applications Programmer Guide |
| CE50 | CP-6 TP Administrator Guide |
| CE51 | CP-6 FPL Reference Manual |

| | |
|---|---|
| CE52 | CP-6 Text Processing Administrator Guide |
| CE53 | CP-6 Text Processing Primer |
| CE54 | CP-6 I-D-S/II Guide |
| CE55 | CP-6 Application Programmer Handbook |
| CE56 | CP-6 Pocket Guide to User Documentation |
| CE62 | CP-6 System Programmer Guide |
| CE64 | CP-6 Operations Pocket Guide |
| HA01 | CP-6 Introduction to ARES |
| HA02 | CP-6 ARES Reference Manual |
| HA03 | CP-6 Introduction to MAIL |
| HA04 | CP-6 MAIL Reference Manual |
| HA05 | CP-6 Introduction to PCF |
| HA06 | CP-6 PCF Reference Manual |

Manuals may be ordered from:

Honeywell Information Systems, Inc.
Publications Distribution Center
47 Harvard Street
Westwood, Massachusetts 02090

Telephone: Customers (617) 392-5235
Honeywell (HVN) 273-5215 (HED MA06)

# MODULE 1-0

## Section 1 - Obtaining Information

This section shows how to obtain information concerning the CP-6 system through the the HELP facility and by means of the various displays available to the user.

# MODULE 1-1

# HELP

SETUP: Alas. Somebody swiped my CP-6 reference manuals again, and I need to get some work done. I seem to recall a highly touted CP-6 HELP facility. Hmmm...

Let's see. I want to know about the LDEV command so I can set a logical device. It would be pretty logical to say "HELP LDEV" to find out about LDEV. I'll try it.

```
!HELP LDEV
```

Syntax:

```
LDEV lname [fid][,optionlist]
```

Hey, it works! But I want to know what the options are. When I get an error message on CP-6, I type a question mark to get more information. Maybe this works with this HELP facility, too.

```
!?
```

Parameters:

```
lname   -   is a user-assigned logical device name, or the system logical
    .
    .
    .
HE[ADER]=NO
IN[DENT]=value
LI[NES]=value
    .
    .
    .
```

There's the option I wanted, INDENT. But I'm still not sure how to use the command. Maybe I can get an example.

```
!HELP LDEV EXAMPLES
```

Examples:

```
!LDEV LP01 LP@NORDOR,COPIES=7

means, when LP is used as an output device, the system should send that output
    .
    .
    .
```

Alright! Now I can issue my LDEV command, but I'll use my own workstation name, and the INDENT option.

```
!LDEV LP01 LP@INFERNO,IN=100
```

The other day I saw Dennis getting information about the ASCII character set on his terminal. I wonder if that's in HELP.

```
!HELP ASCII
To obtain information about the ASCII character set, enter:

    !HELP ASCII character

where character is:

    A character or mnemonic (for unprintables)
    An octal value in the form          0'ooo'
    A decimal value in the form         D'ddd'
    A hexadecimal value in the form     X'xx'

To list the entire ASCII character set, merely type ??<CR> at the next prompt.
```

I don't want the entire ASCII set, but HELP tells me that I can obtain information about an individual character. I'll try that.

```
!HELP ASCII 4
4        0'064'    D'052'    X'34'       Four
!HELP ASCII 0'123'
S        0'123'    D'083'    X'53'       Uppercase Letter
```

HELP gave me the information I requested. Now I want to ask for information about escape sequences:

```
!HELP ESCAPE A
<ESC> <A>
            Toggles (reverses the setting) CRT pagination mode.  When on,
            output will halt at the end of each screen full of output,
            relative to the last input line (depends upon whether relative
            pagination is on.)  Hitting carriage return resumes output.
```

This is all very interesting. I wonder if there's an easy way to find out about all the information I can get from this HELP file. I think I'll try just a plain, old "HELP".

```
!HELP
Welcome to B03 HELP!

The HELP command prints information about CP-6 and its processors
(system programs).

The message you are reading tells how to use the HELP command,
and a little about how the HELP information is structured.

We suggest that you read this entire message before actually
issuing HELP commands.

Now type a question mark "?" followed by a carriage return.
to get the next part of this message.

Type     <?><CR>
!?
You have now learned an important lesson about HELP.  Typing a
question mark displays next part of the message.

Just for your information, typing a double question mark (??) will
print all the parts of the message.

For example, if you typed     ??<CR>  at the next "!" prompt,
the entire remaining description of HELP will be printed.
```

But don't type a double question mark yet. Just enter a single question mark for more information about HELP.

Type <?><CR>

```
!?
Let's look at the general rules about HELP.  You can almost always
find out how to call a processor by entering:
   HELP (processor)
For example:
   HELP (BASIC)
prints the syntax of the call to BASIC.  In some processors, a
description of the processor is printed instead.
      .
      .
      .
```

That tells me a lot. I can put a processor name in parentheses to get more information about that processor. Like this, I suppose:

```
!HELP (EDIT) RR
Syntax:
RR [n1[-n2]]
```

I wonder if I can get the same information if I'm actually in EDIT?

```
!EDIT
EDIT B03 HERE
*HELP RR
Syntax:
RR [n1[-n2]]
*END
```

This is beginning to make sense. I wonder what's in the COBOL HELP facility?

```
!HELP(COBOL)
The COBOL compiler is invoked by using the standard CP-6 processor
invocation line.
Format:
                          [{ON  }                          ]
!COBOL [source[, update]] [{OVER} [object][, listout]] [(optionlist)]
                          [{INTO}                          ]
```

That gave me the COBOL compiler invocation. Now I'll try COBOL OVERVIEW...

```
!HELP(COBOL)OVERVIEW
The following list gives an overview of the
major groups of topics provided by the COBOL
HELP (COBOL) facility
COBOL_DIV for COBOL divisions
COMPILER
IMPERATIVE_VERBS
VERB_CATEGORIES
      .
      .
      .
SEGMENTATION
SAMPLE_PROGRAM
Each of the above topics, when requested provides
general information and a HELP (COBOL) pointer or pointers
to specific information at the end of the topic.
For a complete list of topics, enter:
HELP (COBOL) TOPICS
```

Hmmm... I think I'll try PCL TOPICS...

```
!HELP (PCL) TOPICS
BREAK_RESPONSE        BUILD       COMMANDS    COMPLEX     COMPONENTS COPY
COPYALL     COPYSTD   DCBS        DELETE      DELETESTD   DESTINATION
    .
    .
    .
WILD_DESTINATION
```

That seems like a pretty reasonable list for the PCL processor. Too bad that the top lines went off my screen (I forgot to set <ESC><A> for CRT pagination). I'll try to get a shortened list:

```
!HELP (PCL) TOPICS B - C
BREAK_RESPONSE      BUILD       COMMANDS    COMPLEX    COMPONENTS     COPY
COPY      COPYALL   COPYSTD
```

Or maybe even shorter:

```
!HELP (PCL) TOPICS B
BREAK_RESPONSE        BUILD
```

I think I'll try some more... (FADE TO BLACK)

SETUP: You have logged on and submitted an important batch job. It is necessary to check on the status of the job continually, while at your terminal.

```
!BATCH $CE32:HELP
47989  submitted.
```

You use the CHECK command to see the status of your job, and are told that it is still waiting to run in the batch queue. When using CHECK, you do not have to specify the sysid, since this was "remembered" by the system when you BATCHed the job.

```
!CHECK
47989 .XXZZYYQ waiting 0 to run at prio 7
```

While waiting, you display your current profile and all other available profiles are displayed as well.

```
!DI PROFILE
   PROFILES:
      CURRENT: TTY
      AVAILABLE:  $ASYNC      $HDLC       $RBT        $RBTD       $URP
      7802X72     ADDS200     ADDS25      ADDS60      ADDS980     BEEDM20
      CDI1203     CDI1203S    CDI1205     CDI1205W    CTR6300     CUMSLV
```

You ask for your type of setup file.

```
!DI SETUP
   SETUP: !XEQ SETUP
```

You take another look at the batch job. It is still waiting in the batch job queue.

```
!CHECK
47989 .XXZZYYQ waiting 4 to run at prio 7
```

You decide to check your disk space. DISPLAY USER shows you what user number you have, how much disk space is left in your account, your logon, directory, and setup command file.

```
!DI USER
   XXZZYYQ,   473SINCLAIR  SYSID = 47984  USER NUMBER = 80
   DIRECTORY = XXZZYYQ   DISK SPACE REMAINING = 501
   SETUP: !XEQ SETUP
```

You check your job again, and find that it is running.

```
!CHECK
47989 .XXZZYYQ running 0:35/29:42
```

While your job is running, you check on general system usage: DISPLAY shows number of current users, ETMF (execution time multiplication factor) which relates program CPU time to job throughput time, RESPONSE (number of milliseconds that just exceed response time of 90% of responses to terminal requests), and date/time. For a complete listing of available DISPLAY options, use the HELP facility.

```
!DI
   USERS = 75
   ETMF =  1
   90% RESPONSE < 50 MSECS
   FEB 03 '83   10:35
```

The STATUS command will display accounting information concerning your system usage.

```
!STATUS
CON=000:19:33 EX=000:00:04.09 SRV=000:00:23.95 PMME= 4893 CHG=   2.11
```

You check your job once more and are told it is printing. You log off.

```
!CHECK
47989-1 .XXZZYYQ printing on LP04@LOCAL
!OFF
```

## CHECK and NOTIFY - Comments

The CHECK command reports the current status of jobs for the current user (i.e. waiting to run, running, printing, etc.). Once the job has finished, CHECK tells you how it was completed. A word of caution: once CHECK has reported a job as completed, it forgets the SYSID. If CHECK is used after this, the sysid must be specified.

If you use the NOTIFY command, it will call on CHECK to report on any of your jobs which have changed status since the last report. This occurs upon the completion of each job step.

# MODULE 2-0

## Section 2 - Controlling Terminals

This section shows how to control terminals and terminal input, and how the user can record a terminal session through use of the DRIBBLE command.

# MODULE 2-1

# Setting Terminal Profiles

The CP-6 system accommodates a number of terminal types (i.e, Diablo 1550, Honeywell VIP 7802, etc.) Use the IBEX DISPLAY PROFILE command to obtain a listing of the available profile types.

Immediately after logging on, you should set your terminal profile to tell the CP-6 system what terminal you are using. After logging on successfully, and receiving the IBEX prompt, use the PROFILE command:

```
!PROFILE DBL1620
```

This sets the profile for a Diablo 1620.

```
!PROFILE VIP7801
```

sets the profile for a Honeywell VIP 7801.

If you use the same terminal every time you log on (or the same terminal type) you can use the permanent form of the PROFILE command:

```
!PROFILE VIP7801(PERM)
```

Now every time you log on to the same account, your profile will be automatically set. You can also make the PROFILE command part of a setup file, which will be executed every time you log on:

```
!BUILD SETUPS
EDIT B03 HERE
    1.000 !PROFILE DBL1620
    2.000
*END
```

To execute the above file every time you log on, use the SETUP command:

```
!SETUP !XEQ SETUPS
```

You have now instructed the CP-6 system to execute the file SETUPS every time you log on. This will remain in effect until cleared by the SETUP command, i.e., !SETUP RESET. Of course your setup file may be expanded to perform added functions in addition to setting your profile:

```
!EDIT SETUPS
EDIT B03 HERE
*IN 2
    2.000 !DONT ECHO
    3.000 !PLATEN W=80
    4.000 !TABS 3,11,19,27,37,49,57,63
    5.000
```

You have added some new instructions to your setup file, and now request to see the entire file:

```
*TY 1-99
    1.000 !PROFILE DBL1620
    2.000 !DONT ECHO
    3.000 !PLATEN W=80
    4.000 !TABS 3,11,19,27,49,63
*   EOF hit after 4.000
```

The above setup file will set your terminal profile, inhibit the printing of commands from your setup file at your terminal, set the platen width, and preset tabs at your terminal each time you log on.

# MODULE 2-2

# Cursor/Printhead Positioning

When editing or building a file, a quick and easy way to control the cursor/printhead positioning is to use the many control sequences available on the CP-6 system. While it is not necessary to memorize all of the escape and control sequences, you should familiarize yourself with some of the more useful controls. The HELP file is a handy tool to list the different sequences and their definitions. To access the HELP File, type the following after the exclamation point (!) prompt:

```
!HELP ESCAPE_SEQUENCES
```

## Moving the Cursor

Following are some examples to show you how to use a few of the different control and escape keys. The underscore (_) denotes the placement of the cursor position.

### ESCAPE N

To move the cursor to the end of an input line the <ESC> N sequence is used.

```
1.000 The rain in_Spain falls
```

To move the cursor to the end of the line simply type <ESC> N. The cursor moves to the following position and you can resume typing:

```
1.000 The rain in Spain falls_
```

## ESCAPE V - Moving to Character 'N'

<ESC> V are the first two characters of a three character sequence. The third character 'n' is the criteria for the search of the input record. If 'n' is not found, no action is taken.

Consider the following example with the cursor at the end of the line:

1.000 I have one smale skill which_

Type <ESC> Ve
The cursor searches the record and
stops at 'e'.

1.000 I have one smale skill which

Then type 'l' and your mistake is fixed.

1.000 I have one small_skill which

## Tab Stops

Input (TAB) stops are set differently on the computer than on a typewriter. Instead of manually setting the tabulation stops, you use a "TABS" command. Up to 32 tabs can be set in a sequence. Tabs can be set after the exclamation point (!) prompt or after the asterisk (*) prompt in EDIT.

- Tabs set after the exclamation point prompt (!) remain effective until you log off or until you turn the command off with the TABS {O|NO|OFF} command.

  Example:

  !TABS 10,20,30,40

  means set the tab stops at 10, 20, 30, and 40.

- Tabs set after the asterisk prompt (*) in EDIT remain effective until you end the editing session or issue the TABS OFF command.

## ESCAPE I

If your terminal does not have a "TAB" key you have the option of moving the cursor to the next input stop with the <CNTL> I or the <ESC> I sequences.

### CONTROL R - Forward Positioning

The <CNTL> R sequence is the opposite of backspacing; it moves the cursor one space to the right without altering any skipped characters.

Example:

1.000 I have one small skill wh_ich

By holding down <CNTL> and depressing R three times your terminal will display the following:

1.000 I have one small skill which_

### CONTROL H - Backspacing

The <BS> key moves the cursor one space to the left. If your terminal does not have a BACKSPACE key simply type <CNTL> H and the cursor will backspace.

Example:

1.000 The rain in sp_ain

Hold down <CNTL> and type H twice and the cursor will move accordingly:

1.000 The rain in _spain

## ESCAPE <RET> - Position to Beginning of Record

After typing a line and finding that it is not right you can return to the beginning of a line by typing <ESC><RET>.

Example:

1.000 I don't like this_

Type <ESC><RET>

1.000 I don't like this

and begin typing the line over.

# MODULE 2-3

# Cancelling and Recalling Input

The CP-6 system has a series of escape keys that make life a lot easier for those programmers who change their minds in mid-stream. These escape sequences allow you to cancel or recall input with ease. To obtain a complete list of the escape sequences from the HELP file, type the following after the exclamation point (!) prompt:

```
!HELP ESCAPE_SEQUENCES
```

In the following examples the underscore (_) denotes the cursor position.

## ESCAPE X - Deleting Current Input Line

If you change your mind while typing a line, the <ESC> X sequence will clear the current input record to blanks and you can begin retyping the line.

Example:

```
1.000 Little Bo Peep has lost her ship_
```

If you decide this isn't exactly what you want to say, simply type <ESC> X and your input record will look like this:

```
1.000 Little Bo Peep has lost her ship<X>

1.000 _
```

Now you can begin retyping the line.

## ESCAPE K - Deleting From Current Edit Point

If you want to delete only a portion of a record, you can use <ESC> K to delete from the current edit point to the end of the record.

Example:

1.000 The rain in Spain falls mainly on the plain._

You reposition the cursor by backspacing:

1.000 The rain in Spain_falls mainly on the plain.

Now you use <ESC> K

1.000 The rain in Spain_falls mainly on the plain.
<K>
1.000 The rain in Spain_

As a result the line to the right of the cursor or edit point has been deleted.

## ESCAPE A - Setting Pagination Mode

SETUP: You want to use TEXT to display your file named ZEBRA on your CRT. You want to check for proper page breaks, and also to look for typos, mispelled words, and other errors. You enter:

!TEXT ZEBRA

The computer responds with:

TEXT C01

The file starts printing on the CRT but the lines go by so fast that you can't read a word.

PROBLEM: How can you TEXT your file, but halt the output at the end of each full CRT screen?

SOLUTION: After referring to the HELP file under ESCAPE_SEQUENCES, you find that by typing <ESC> A you can control the pagination mode on the CRT. The CRT halts at the end of each full screen of output and waits for you to hit <CR> to resume output.

Example:

!<ESC> A
!TEXT ZEBRA

Since the <ESC> A toggles (reverses the setting), you need only to press <ESC> A to resume normal output.

<ESC> A may be used to control the pagination mode on the CRT for any type of display, so it may be used while in EDIT or while COPYing a file to your CRT.

## ESCAPE R - Retyping the Current Input Line

While editing at a hard copy terminal, a line might look like this:

```
1.000 AD
       \BC
```

If you're not sure what the record actually contains, just type <ESC> R; this cancels the insert mode, and the current line is retyped like this:

```
1.000 ABCD
```

## ESCAPE D - Retrieving the Last Input Line

The <ESC><D> sequence retrieves the last line of input as if you had just retyped it. This is especially useful if you are typing a long line with basically the same information.

Example:

```
1.000              WRITE OUT-PUT RECORD FROM CARD-A. <CR>
2.000 <ESC><D>
```

The terminal displays:

```
1.000              WRITE OUT-PUT RECORD FROM CARD-A.
```

Backspace and change the 'A' to 'B', and the terminal displays:

```
1.000              WRITE OUT-PUT RECORD FROM CARD-B.
3.000 _
```

# MODULE 2-4

# Inserting/Replacing/Overstriking

You are reviewing a file called KIWI. You realize that there are certain words that would be more distinguishable if they were underlined. You also notice that you have a few mispelled words.

How can you insert the required letters into your mispelled words, and underline the words that you want to emphasize?

## ESCAPE J, ESCAPE <CR>, ESCAPE O, ESCAPE M

By using several different escape sequences, you can both underline words and insert letters.

Example:

| STEP | YOU ENTER | COMPUTER RESPONDS |
|---|---|---|
| 1 | EDIT KIWI | EDIT B03 HERE<br>* |
| 2 | RR 19 | 19.000 Now is the tme for all |
| 3 | (backspace to m)<br><ESC><J> | (carriage moves up one line and prints \) |
| 4 | (Strike 'i' and <CR>) | (i is inserted) |
| 5 | RR 19 | 19.000 Now is the time for all |
| 6 | <ESC><CR> | (carriage backspaces to beginning of line) |
| 7 | <ESC><O><br>(underscore 'Now')<br><ESC><M> <CR> | (sets overstrike mode "on")<br>19.000 Now is the time for all<br>(resets overstrike mode to "off") |
| 8 | TY19 | 19.000 Now is the time for all |

COMMENTS: The CRT screen does not have the capacity to display the underscore as well as the overstruck character; it only displays the last character typed. However, it is possible to input the underscore first, and then use <ESC><O> to enter text characters. In this way, you will be able to read the text on the CRT, although you won't see the underscore.

The overstrike mode is a useful one and it can be used for other tasks besides underlining, such as combining 'b' and '/' to create a 'ƀ', or '=' and '/' to create '≠'.

When using the <ESC><J> sequence the CRT doesn't display the '\' or move down a line, although it still allows you to insert a character.

When using <ESC><O> (overstrike mode), remember that you will remain in that mode until you use <ESC><M> to exit.

## More About ESCAPE J

It is sometimes useful to combine <ESC><J> with <ESC><Vn>. <ESC><Vn> moves you to an insert point; <ESC><J> can then be used to insert the desired character. In the following example, the underscore (_) denotes the placement of the cursor.

| STEP | YOU ENTER | COMPUTER RESPONDS |
|---|---|---|
| 1 | EDIT PFILE | *EDIT B01 HERE |
| 2 | RR 1 | 1.000 PL6 XXX.Y OVER ZZZ.Y_ |
| 3 | <ESC><V.> | 1.000 PL6 XXX.Y OVER ZZZ_.Y |
| 4 | <ESC><J>Q | 1.000 PL6 XXX.Y OVER ZZZ.Y<br>\Q_ |
| 5 | <ESC><J> | 1.000 PL6 XXX.Y OVER ZZZQ_.Y |
| 6 | <ESC><V.> | 1.000 PL6 XXX_.Y OVER ZZZQ.Y |
| 7 | <ESC><J>Q | 1.000 PL6 XXX.Y OVER ZZZQ.Y<br>\Q_ |
| 8 | <CR> | |
| 9 | TY | 1.000 PL6 XXXQ.Y OVER ZZZQ.Y |

# MODULE 2-5

# Platen Control

SETUP: You want a print-out of a file with the following parameters in the final print-out:

- No more than 65 characters in a terminal line.
- No more than 60 lines to a page.
- No more than 3 lines between the last printable line in the page and the page perforation.
- 0 lines between the perforation and the first printed line.

SOLUTION: You use the PLATEN command to tell the computer that you wish to define the parameters instead of using the defaults. You find that:

- WIDTH (or W) = number of characters in a terminal line.
- LENGTH (or L) = number of printable lines in a page.
- LIMBO (or LI) = number of lines between the last printed line and page bottom.
- FIRST (or F) = number of lines between the top of the page and the first printed line.

You insert the following:

```
!PLATEN L=60,W=65,F=0,LI=3
!TEXT MYFILE
```

This gives you the desired results.

## Determining Platen Settings

When you log on to the CP-6 system, the PROFILE setting for your terminal automatically sets the platen variables. You can use the PLATEN command, followed by a carriage return, to display your current setting. For example:

```
!PLATEN <CR>
   PLATEN: WIDTH = 80  LENGTH = 0
```

## Resetting PLATEN Parameters

Use the PLATEN command to reset any of the variables such as length or width:

```
!PLATEN W=80
```

The above command resets the platen width to 80 characters.

# MODULE 2-6

# IMP

You can use the Input Manipulation Processor (IMP) to redefine keystroke sequences and special characters on any terminal. The new sequences or characters may be unique combinations of system escape sequences and special characters, or new special purpose functions.

You can use IMP to redefine the keyboard of one terminal so that it performs like the keyboard of another terminal. Redefined keys can perform commonly used functions or commands, or generate frequently used strings.

This module covers the following three topics:

- Defining new escape sequences
- Setting up function keys
- Redefining the keyboard

The first two show how to create temporary IMP function keys; the last shows how to create a permanently reuseable IMP function.

## Defining New Escape Sequences

Try the following example at your terminal to see how IMP works:

```
!IMP (ADD PRIMARY I=ESC '1' TEXT='xanthocyanopia' I TY)
```

You now can use the new IMP keyin <ESC><1> to type the word 'xanthocyanopia' whenever it occurs. You build a file to try this out:

```
!BUILD PATIENT_HISTORY
EDIT B03 HERE
    1.000 The patient is suffering from acute
    2.000
```

Here you press <ESC><1>; the word 'xanthocyanopia' is printed at the terminal. You can then enter the rest of the text.

```
          xanthocyanopia which makes her unsuitable
    3.000 as a color specialist at the paint factory.
    4.000

!TEXT PATIENT_HISTORY ON ME
TEXT C01

The patient is suffering from acute xanthocyanopia which makes
her unsuitable as a color specialist at the paint factory.
```

When you TEXT the file, the word 'xanthocyanopia' is included as part of the text.

The above IMP command has been used to add a temporary primary escape sequence. When this escape sequence is activated, the text of the IMP command is placed in the typeahead buffer. Then, the word 'xanthocyanopia' is entered into the stream of text as if it has just been typed at the terminal.

You can also use IMP to issue frequently used commands, such as might be used when you batch a daily report:

```
!IMP (ADD PRIMARY I=ESC '2' TEXT = 'BATCH DAILY_RPT', CR I TY)
```

Now by touching <ESC><2> the report is entered into batch queue. Note that the presence of CR in the IMP command results in an automatic carriage return.

```
!BATCH DAILY_RPT
29859 DAILY_RPT.MYACCT running 0:00/9:59
```

This command also adds a temporary primary escape sequence. The text of the command is read immediately and placed in the typeahead buffer. The command 'BATCH DAILY_RPT' is then initiated as if it has just been entered by the user.

The above examples create temporary IMP function keys which are in effect only for the duration of the logon session.

## Setting Up Special Function Keys

If you have an often used source file, the following IMP command can be used to open the file in EDIT and print the desired number of lines, in this case 0 through 10:

```
!IMP (ADD SPECIAL '#' TEXT='EDIT MY_FILE',CR,'TY 0-10',CR TY R
```

Entering the # character then executes the following sequence at your terminal:

```
!EDIT MY_FILE
EDIT B03 HERE
*TY 0-10
    1.000 .*****************************************************************
    2.000 .*                                                               *
    3.000 .*   Copyright (c), Honeywell Information Systems Inc., 1983     *
    4.000 .*                                                               *
    5.000 .*****************************************************************
    6.000 .*
    7.000 .FBB
    8.000 ||%PAGENO%||
    9.000 .FBE
   10.000
```

This example shows how IMP can be used to provide combinations of often typed keystroke sequences. The special character # is defined by the IMP command to activate many keystrokes at once. The text will be echoed to your terminal (if echoplex is on), and the series of commands will be activated using one keystroke instead of several.

The next example adds a special character \ to be used in place of an escape key on the keyboard:

```
!IMP (ADD SPECIAL '\' T=ESC R I IN)
```

A command such as this could be used if the escape key on a keyboard is inconvenient or nonexistent. The \ now becomes an escape key.

These examples create temporary IMP function keys which are in effect only for the duration of the logon session. See Redefining The Keyboard, below, for an example of how to create permanently reuseable IMP functions.

## Resetting Special Function Keys

Once you have used IMP to set a special function key, you may need to reset the definition for that key. There are several methods you can use to accomplish this. You can use the IMP DELETE ALL to delete all user-defined input functions. You can also use the IMP DELETE command to delete the special function key, which can then be reset.

CAUTION: You cannot insert the special character as part of the DELETE command, as this character will always trigger the IMPed substitute. The way out of this 'dilemma' is to use the ASCII octal, decimal, or hexadecimal string equivalent of the character in question.

Examples:

You have set up a special IMP function for the # character. You want to change this function.

```
!IMP
IMP B03 HERE
->DELETE ALL
->END
!
```

You invoke IMP and use the DELETE ALL to delete the special character input function. Remember, this also deletes any other user-defined IMP input function as well.

```
!IMP
IMP B03 HERE
->DEL SPECIAL O'043'
->END
!
```

You invoke IMP and use the octal equivalent of the # character to delete the special function for that character.

Finding the ASCII Equivalent

To find the ASCII octal, decimal, or hexadecimal string equivalent of a character, use HELP to ask for information about ASCII characters:

```
!HELP (IBEX) ASCII
```

You ask HELP for information about ASCII characters, and HELP responds:

> To obtain information about the ASCII character set, enter:
>
> !HELP (IBEX) ASCII character
>
> where character is:

```
        A character or mnemonic (for unprintables)
        An octal value in the form          O'ooo'
        A decimal value in the form         D'ddd'
        A hexadecimal value in the form     X'xx'

    To list the entire ASCII character set, merely type ??<CR> at the
    next prompt.

    The table has the following format:
  CHAR      OCTAL      DECIMAL    HEX          MEANING
```

In answer to your request, HELP has given you the format of the ASCII table. You now enter a double question mark (??).

```
!??
  NULL      0'000'     D'000'     X'00'        NULL of time fill character
  SOH       0'001'     D'001'     X'01'        Start Of Heading
  STX       0'002'     D'002'     X'02'        Start Of Text
  ETX       0'003'     D'003'     X'03'        End Of Text
  ECT       0'004'     D'004'     X'04'        End Of Transmission
    .
    .
    .
  US        0'037'     D'031'     X'1F'        Unit Separator
  SP        0'040'     D'032'     X'20'        SPace
  !         0'041'     D'033'     X'21'        Exclamation Point
  "         0'042'     D'034'     X'22'        Quotation Mark
  #         0'043'     D'035'     X'23'        Number Sign
-
```

When you arrive at the # sign in the table, you interrupt with the BREAK key. You now see that the equivalent for # is 0'043' or D'035' or X'23'. Any of these three strings may be used in conjunction with the IMP ADD or DELETE command in place of the # character.

## Redefining the Keyboard

A series of IMP functions can be made into an IMP command file and initiated as part of a normal setup process which occurs at logon. Following is a sample IMP command file built in EDIT for a VIP7801. Notice the abbreviation of ADD (A) and PRIMARY (P) at the beginning of each command, and the abbreviation of IMMEDIATE READ (I R) in lines 9, 13, and 14. The file name is IMP7801.

```
 1.000 D ALL
 2.000 A P I=ESC 'e' TEXT='TN 10',CR I R TY ECH "TYPE NEXT 10"
 3.000 A P I=ESC '0' TEXT=ESC,'V ' IMMEDIATE READ TYPEAHEAD ECHO
 4.000 A P I=ESC '2' TEXT=DC2 I EC READ INPUT
 5.000 A P I=ESC '6' TEXT=ESC,CR   READ INPUT
 6.000 A P I=ESC '8' TEXT=ESC,'N'  READ INPUT
 7.000 A P I=ESC ':' TEXT=ESC,'V'  READ INPUT
 8.000 A P I=ESC '<' TEXT=ESC,'J'  READ INPUT
 9.000 A P I=ESC '>' TEXT='BATCH STAR_RPT',CR I R TYPEAHEAD ECHO
10.000 A P I=ESC 'P' TEXT=ESC,'D' READ INPUT
11.000 A P I=ESC 'R' TEXT=ESC,'X' READ INPUT
12.000 A P I=ESC 'T' TEXT=ESC,CR,ESC,'V ',DC2,ESC,'J',DEL,DEL,DEL,
     DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,
     DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,DEL,
     DEL,DEL,DEL,DEL,DEL,DEL,DC2,BS IMMEDIATE READ TYPEAHEAD ECHO
13.000 A P I=ESC '\' TEXT=ESC,'N',ESC,'V ',ESC,'K' I R TYPEAHEAD ECHO
14.000 A P I=ESC ' ' TEXT=BS,ESC,'O_',ESC,'M',BS I R TYPEAHEAD ECHO
15.000 A P I=ESC 'i' TEXT='TP 10',CR I R TY ECH "TYPE PRIOR 10"
16.000
```

Note that line 12.000 is actually one continuous line, limit 255 characters.

The functions provided by the IMP command file are:

EDIT

| LINE # | CODE KEY | FUNCTION |
|---|---|---|
| LINE 1 | -- | DELETES ALL IMP COMMANDS CURRENTLY EFFECTIVE |
| LINE 2 | CLEAR/RESET | TYPE NEXT 10 LINES |
| LINE 3 | F1 | FIND NEXT BETWEEN WORD BLANK |
| LINE 4 | F2 | FORWARD SPACE WITHOUT DELETING (CTL R) |
| LINE 5 | F3 | BEGINNING OF LINE (ESC CR) |
| LINE 6 | F4 | END OF LINE (ESC N) |
| LINE 7 | F5 | FIND CHAR (ESC V) |
| LINE 8 | F6 | INSERTION MODE (ESC J) |
| LINE 9 | F7 | BATCH STAR_RPT |
| LINE 10 | F8 | RECALL LAST LINE (ESC D) |
| LINE 11 | F9 | DELETE LINE (ESC X) |
| LINE 12 | F10 | DELETE FIRST WORD |
| LINE 13 | F11 | DELETE LAST WORD |
| LINE 14 | F12 | UNDERSCORE BACKWARDS |
| LINE 15 | TRANSMIT | TYPE PREVIOUS 10 LINES |

The special set of IMP commands shown above is then automatically activated each time you log on, if you insert the following IBEX command into a setup command file associated with your user logon.

```
!IMP IMP7801
```

This command initiates the IMP processor which reads the IMP commands; the commands take effect immediately. Each command is echoed to the terminal as it is read by IMP.

A second method is to create an object file with the following command:

```
!IMP IMP7801 OVER IMP7801_OU
```

IMP then reads the object file as part of the setup process using this IBEX command:

```
!IMP IMP7801_OU
```

These commands are also effective immediately, but the commands are not echoed to the terminal during the reading of the file, and are processed slightly faster.

# MODULE 2-7

# Dribble Files

Have you ever watched a crucial piece of information scroll beyond your view on a CRT terminal and wished you could scroll backwards to see it again? Most CRT terminals do not have this backwards scrolling capability, but with CP-6 dribble files you can capture your terminal session on a disk file for later examination.

The relevant command to accomplish this wonder is:

```
!DRIBBLE [(ON|OVER|INTO) fid]
```

The DRIBBLE command directs your terminal output into a file for later replay. Examples of the command are:

```
!DRIBBLE OVER ANYFILE
```

which causes unconditional replacement of the existing file.

```
!DRIBBLE INTO OTHERFILE
```

which will append the forthcoming terminal session onto the end of file OTHERFILE, or create OTHERFILE if it did not already exist.

```
!DRIBBLE ON NEWFILE
```

will only start off the dribble process if NEWFILE did not exist when the DRIBBLE command was issued.

The dribble process is turned off with the command:

```
!DONT DRIBBLE
```

And after the above command you can copy the file just created to the line printer, edit the file looking for the useful piece of information that scrolled off your screen, or delete the file because you don't need to record the terminal session any longer.

## Displaying DRIBBLE Files at Your Terminal

If you want to display a DRIBBLE file at your terminal, a word of caution is in order. If the terminal session you recorded with DRIBBLE includes control functions such as IMP settings, TAB settings, etc., the DRIBBLE file, when copied to your terminal, will cause these control functions to be performed again. There are a number of ways in which this can be avoided. One way is to use the PCL NVFC option:

```
<COPY DRIBBLEFILE TO ME(NVFC)
```

This inhibits the function of the VFC (vertical format control) characters at your terminal, although the VFC characters will be retained in the DRIBBLE file. Or, PCL may be used to copy the DRIBBLE file and delete the VFC characters:

```
<COPY DRIBBLEFILE OVER MYFILE(NVFC)
```

This results in a new file, MYFILE, from which the VFC characters have been removed.

An alternate method of removing the VFC characters from a DRIBBLE file is to use ELBBIRD.X, if this capability is available at your site:

```
!ELBBIRD.X DRIBBLEFILE
```

This copies the file named DRIBBLEFILE over itself as an EDIT keyed file. Now you have a keyed file, DRIBBLEFILE, with line numbers starting at 1.000, in increments of 1.000 (the default).

Please note that ELBBIRD.X may not be available at all sites.

# MODULE 3-0

## Section 3 - Editing and Manipulating Files

This section shows how to create, edit, and manipulate files.

# MODULE 3-1

## Creating New Files

In this example, the BUILD command is used to create four files: FILE1, FILE2, FILE3 and FILE4. FILE1 is created via EDIT, using the default values. FILE2, FILE3 and FILE4 are created after the user calls EDIT.

FILE3 and FILE4 are then listed to illustrate the difference between your choosing the file type, and allowing the file type to be chosen by default.

Starting from the IBEX (!) level, you enter the BUILD command:

```
!BUILD FILE1
EDIT B03 HERE
    1.000
```

EDIT responds with a message and line number prompt.

```
    1.000 Mary had a little lamb.
    2.000 Its fleas were white as snow.
    3.000
```

You enter a line of input after each prompt and a <CR> only after the third line prompt to indicate completion of data entry.

```
!EDIT
```

You call EDIT in response to the ! prompt.

```
EDIT B03 HERE
*
```

EDIT responds with a message and asterisk prompt.

```
*BUILD FILE2
    1.000 And everywhere that Mary went,
    2.000 The fleas were sure to follow.
    3.000
```

You build FILE2.

```
*BUILD FILE3,10,10
*   EDIT stopped
   10.000 They followed her to school one day,
   20.000 Which was against the rules.
   30.000
```

You build FILE3, starting at line number 10 and incrementing by 10.

```
*BUILD FILE4,,QT
*   EDIT stopped
    1.000 It made the children itch and scratch
    2.000 and unteachable as mules.
    3.000
```

You build FILE4, selecting file type QT.

```
*L FILE4
ORG TY  GRAN  NGAV    REC   LAST MODIFIED   NAME
KEY QT     2     0      2                   FILE4

*L FILE3
ORG TY  GRAN  NGAV    REC   LAST MODIFIED   NAME
KEY SE     1     0      2                   FILE3
```

You list FILE3 and FILE4. Note that FILE4 has a type of QT (user selected) and FILE3 has a type of SE (the default).

```
*END
```

You type END to return to IBEX.

In the following example, you create FILE5 consisting of two records (using the COPY ME command), list FILE5 to illustrate that COPY ME creates a consecutive file, and delete all five files created during the two examples.

```
!COPY ME TO FILE5
```

You use the COPY ME command to create FILE5.

```
 .RECORD1
 .RECORD2
 .<F>
```

You create two records, the contents of which are "RECORD1" and "RECORD2". The COPY ME routine prompts with a period (.). After the second record is entered, you input <ESC> <F> (end of file) to exit the routine.

```
!L FILE5
ORG  TY   GRAN   NGAV    REC   LAST MODIFIED   NAME
CON         1      0      2                   FILE5
```

You list FILE5, showing the organization to be consecutive.

```
!DEL FILE?
DELETE FILE?.MYACCT ?YES$
FILE1       FILE2       FILE3       FILE4       FILE5
     5 files,     9 granules deleted
```

You use DEL FILE? to request that all your files starting with "FILE" be deleted. PCL seeks a confirmation of the deletions by asking DELETE FILE? A response of YES$ causes multiple deletions. The files are listed as they are deleted and the number of files and granules freed are listed after the deletions are complete. The use of FILE? is an example of wildcarding. See Wildcarding module, CE55.

# MODULE 3-2

# EDIT Command Files

SETUP: You have a file, COBOL-EXAMPLE, that you want to make changes to using EDIT commands. You do not want to make the changes to the original program, because you want to use it 'as is' at a later date. You know that you can make these changes by entering them directly into the terminal before running the program, but the list of commands is quite lengthy.

PROBLEM: How can you edit the file with ease, retaining the original program?

SOLUTION: You decide to build a file of EDIT commands and then, by using the EDIT READ command, execute that file. You call your file EDIT-COMMAND-FILE, and build it as follows:

```
!E
EDIT B03 HERE
*BUILD EDIT-COMMAND-FILE
    1.000 COPY COBOL-EXAMPLE OVER COBOL-EXAMPLE2
    2.000 SE 0-1000;/1 LINES/s/2 LINES/
    3.000 /DATA-FILE/s/REPORT-FILE/
    4.000 0/PIC X(80)/s/PIC X(132)/
    5.000 /SEPTEMBER 29, 1983/s/OCTOBER 27, 1983/
    6.000
*E
```

The first line of your EDIT command file copies your COBOL program, COBOL-EXAMPLE, to COBOL-EXAMPLE2. Then you set the selection criteria in COBOL-EXAMPLE2 and specify changes to the file.

After building the EDIT-COMMAND-FILE all you need do is:

```
!EDIT
EDIT B03 HERE
* READ EDIT-COMMAND-FILE
```

This means read and execute the EDIT commands contained within the EDIT command file.

After using the READ command to execute your EDIT command file, you will find a new file, COBOL-EXAMPLE2, in your account which contains the desired changes. The original file, COBOL-EXAMPLE, remains intact.

# MODULE 3-3

# Selecting Record Ranges

The EDIT selection criteria sets the record range over which subsequent EDIT commands take effect. The range is set by the EDIT SE command, directly, or by using many other EDIT commands which set the range implicitly. For example,

```
!EDIT
EDIT B03 HERE
*E S8127A
```

Using the E or EDIT command above sets the selection range over the entire file named. You can determine this by requesting the STATUS display:

```
*STATUS
FILENAME = S8127A.MYACCT
DIRECTORY: .MYACCT
EDIT FILE
BP ON
CR OFF
RP OFF
TABX OFF
TABC OFF
VE OFF
NO TABS SET
SE     0.000-99999.999,001,256
*
```

The selection range covers the entire file, records 0-99999. This means, for example, that if you now enter the DE command without specifying any line numbers, the entire range of records will be deleted! So it is important to know what your current selection criteria is when editing a file.

Some EDIT commands, like the RR command, set the selection range to the record numbers indicated. Others, like SS, ST, TS, TY, etc., set the range to the end of the file if no range is specified. To find out for yourself how the SE range is set, build a test file and use various EDIT commands, calling up the STATUS display after each to see what the SE range is.

The SE range is also set by the linefeed function, which allows you to step through the records of a file:

```
*E S8127A
*RR 3
 3.000 The IBEX PREPROCESSOR enables the user to define new commands <CR>
<linefeed>
 4.000 by substitution into normal commands.  This is done by means
```

By using the linefeed key, the SE range is reset, and the next line is displayed. The linefeed function enables you to step through a file record by record, making changes at each step if desired.

Now, let us look at a sample session showing the use of the SE setting:

SETUP: In a creative frenzy, a programmer builds a keyed file with EDIT and later discovers that it contains a lot of errors. Selecting record ranges can speed up the process of editing a file. Record ranges are specified to EDIT in the form starting line number, hyphen, ending line number.

One portion of the file is shown below. (To try the record range selection techniques described below, use EDIT to BUILD a similar file requesting to append lines starting at 101.)

```
  101.000          01  VIEW_VARS              OUTPUT
  102.000              REPEATS 19 TIMES.
  103.000              02  VALUE              PIC 99        POSITION 4, 1
  104.000              02  FILLER             PIC X         POSITION 4, 3
  105.000                  CONSTANT VALUES "="
  106.000              02  SUFFIX             PIC 99        POSITION 4, 4
```

Problem: The ending period is missing on several lines.

Solution: Reread the range of lines that requires change. Use the RR (Reread) command and as each line is displayed, enter a period (if appropriate) or a carriage return if no period is required. In the following sample, periods are added to lines 103, 105, and 106.

```
*RR103-106
  103.000               02  VALUE              PIC 99        POSITION 4, 1.
  104.000               02  FILLER             PIC X         POSITION 4, 3
  105.000                   CONSTANT VALUES "=".
  106.000               02  SUFFIX             PIC 99        POSITION 4, 4.
```

Problem: The identifier VALUE on line 103 is a reserved word which the programmer realizes must not be used as an identifier.

Solution: To find all occurrences of VALUE in the program, enter the Find and Type (FT) command for the entire file. Specifying the range 0-99999 ensures that the complete file is searched.

```
*FT0-99999,/VALUE/
  103.000               02  VALUE              PIC 99        POSITION 4, 1.
  105.000                   CONSTANT VALUES "=".
```

Problem: Some occurrences of VALUE are legitimate uses of the reserved word (line 105, for instance). In other cases, VALUE is mistakenly used as the identifier for a variable (as in line 103, for example).

Solution: Select only the record ranges where VALUE is misused and use the Substitution (S) intrarecord command to change the occurrences when the word is used as a variable.

```
*SE103;/VALUE/S/CENUM/;TY
  103.000               02  CENUM              PIC 99        POSITION 4, 1.
*   1 strings changed
```

Problem: The items CENUM, FILLER, and SUFFIX should be grouped under one identifier, CEVAR.

Solution: Insert CEVAR as a level 02 item by inserting (IN command) line 102.1. Select the items (lines 103-106) to be subordinated to CEVAR; change them from 02 level items to 03 level items (by the Substitution [S] intrarecord command) and indent those items by four additional spaces (by the insert Preceding [P] intrarecord command).

```
*IN102.1

  102.100              02  CEVAR           PIC X(5)      POSITION 4,1.
*se103-106;1P/    /;/02/s/03/;tx
  103.000                   03  CENUM           PIC 99        POSITION 4, 1.
  104.000                   03  FILLER          PIC X         POSITION 4, 3
  105.000                           CONSTANT VALUE "-".
  106.000                   03  SUFFIX          PIC 99        POSITION 4, 4.
*   7 strings changed
```

Problem: After CEVAR is inserted it becomes clear to the programmer that lines 103-106 are unnecessary.

Solution: Delete the unnecessary lines using the Delete command (DE) specifying the range 103-106; then Type that portion of of the file to verify the deletion.

```
*DE103-106
*   4 records deleted
*TY102-106
  102.000              REPEATS 19 TIMES.
  102.100              02  CEVAR           PIC X(5)      POSITION 4,1.
```

# MODULE 3-4

# Moving, Merging, and Copying Files

SETUP: You have just built two files, A and B, which are parts of a COBOL program. You need to incorporate your files into one common file, move a few lines around and generally do some manipulating of data. Notice that the two files have different numbering schemes; this will affect the way in which they are later combined.

```
!EDIT
EDIT B03 HERE
*BU A
    1.000 IDENTIFICATION DIVISION.
    2.000       CHAPTER1.
    3.000       D D MCCRACKEN.
    4.000       5 FEBRUARY 1983.
    5.000
*END
!EDIT
EDIT B03 HERE
*BU B,1.1,1
    1.100 PROGRAM-ID.
    2.100 AUTHOR.
    3.100 DATE-WRITTEN.
    4.100
    5.100 ENVIRONMENT DIVISION.
    6.100
```

An easy way to create a backup file is to use the COPY command. So, you make a duplicate file which is stored in *BLOB. The asterisk (*) designates *BLOB as a CP-6 star file, a file used for temporary storage which will be automatically deleted at the end of a job, or when you log off. See description of star files in the module entitled Maintaining File Accounts (CE55).

```
*COPY B TO *BLOB
*   EDIT stopped
*   Copying
*   COPY done
```

File A requires some changes also, so you copy it into file *ARK which was a previously existing file. Since you specify COPY A OVER *ARK, the information in *ARK is replaced by file A.

```
*COPY A OVER *ARK
*   EDIT stopped
*   Copying
*   COPY done
*TY
    1.000 IDENTIFICATION DIVISION.
    2.000       CHAPTER1.
    3.000       D D MCCRACKEN.
    4.000       5 FEBRUARY 1983.
*   EOF hit after 4.000
*END
```

After all necessary changes have been made to files A and B, you enter the PCL processor to transfer all of file A into file B. Notice that file A and B now have interlaced line numbers.

```
!PCL
PCL B03 here
<COPY A INTO B
 ..COPYing
<END

!EDIT B
EDIT B03 HERE
*TY
    1.000 IDENTIFICATION DIVISION.
    1.100 PROGRAM-ID.
    2.000       CHAPTER1.
    2.100 AUTHOR.
    3.000       D D MCCRACKEN.
    3.100 DATE-WRITTEN.
    4.000       5 FEBRUARY 1983.
    4.100
    5.100 ENVIRONMENT DIVISION.
*   EOF hit after 5.100
```

The line numbers of file B are not in their original sequence, as the file has been modified. To correct this, you copy file B over itself without specifying a target file. Now the entire file is renumbered (i.e. 1,2,3,4...) for easy reading.

```
*COPY B
*   EDIT stopped
*   Copying
*   COPY done
*TY
    1.000 IDENTIFICATION DIVISION.
    2.000 PROGRAM-ID.
    3.000       CHAPTER1.
    4.000 AUTHOR.
    5.000       D D MCCRACKEN.
    6.000 DATE-WRITTEN.
    7.000       5 FEBRUARY 1983.
    8.000
    9.000 ENVIRONMENT DIVISION.
*   EOF hit after 9.000
```

In the meantime, you build another file called X. It contains vital information pertaining to the COBOL program. How do you incorporate it into file B? Using the MERGE command, specify the lines to be transferred into file B, and at which place in file B. File X is then incorporated into file B.

```
*BU X
*   EDIT stopped
    1.000 INPUT-OUTPUT SECTION.
    2.000 FILE-CONTROL.
    3.000 SELECT LINE-OUT-FILE ASSIGN TO UT-S-PRINTER.
    4.000
```

```
*MERGE X, 1-3 INTO B, 10
*   EDIT stopped
*   MERGE started
*   EOF hit after 3.000
*   Done at 12.000
*   3 records moved
*   MERGE done
*TY1-12
    1.000 IDENTIFICATION DIVISION.
    2.000 PROGRAM-ID.
    3.000       CHAPTER1.
    4.000 AUTHOR.
    5.000       D D MCCRACKEN.
    6.000 DATE-WRITTEN.
    7.000       5 FEBRUARY 1983.
    8.000
    9.000 ENVIRONMENT DIVISION.
   10.000 INPUT-OUTPUT SECTION.
   11.000 FILE-CONTROL.
   12.000 SELECT LINE-OUT-FILE ASSIGN TO UT-S-PRINTER.
```

After examining your file, you feel line number 7 should be at the end of the file. With the MD(Move/Delete) command, the original line number 7 is deleted and appears as line number 13.

```
*MD 7,13
*   EOF hit after 12.000
*   Done at 13.000
*   1 records moved
 *TY
   13.000       5 FEBRUARY 1983.
*TY1-13
    1.000 IDENTIFICATION DIVISION.
    2.000 PROGRAM-ID.
    3.000       CHAPTER1.
    4.000 AUTHOR.
    5.000       D D MCCRACKEN.
    6.000 DATE-WRITTEN.
    8.000
    9.000 ENVIRONMENT DIVISION.
   10.000 INPUT-OUTPUT SECTION.
   11.000 FILE-CONTROL.
   12.000 SELECT LINE-OUT-FILE ASSIGN TO UT-S-PRINTER.
   13.000       5 FEBRUARY 1983.
```

Well, you've changed your mind again. You want to move line 13 to line 6.5 (you specify the line number). You use the MK (Move/Keep) command; line 13 remains in its original position. That way you can keep track of where lines were positioned originally in case you want to change them again.

```
*MK 13,6.5
*   Done at 6.500
*   1 records moved
*TY1-12
    1.000 IDENTIFICATION DIVISION.
    2.000 PROGRAM-ID.
    3.000       CHAPTER1.
    4.000 AUTHOR.
    5.000       D D MCCRACKEN.
    6.000 DATE-WRITTEN.
    6.500       5 FEBRUARY 1983.
    8.000
    9.000 ENVIRONMENT DIVISION.
   10.000 INPUT-OUTPUT SECTION.
   11.000 FILE-CONTROL.
   12.000 SELECT LINE-OUT-FILE ASSIGN TO UT-S-PRINTER.
   13.000       5 FEBRUARY 1983
```

So you will have an easy-to-read file, use the RN (renumber) command, which renumbers an individual line number to whatever value you choose. In this case, line number 6.5 is changed to line number 7. Line number 13 is deleted, and you display the final updated version of file B.

```
*RN 6.5,7
*TY
    7.000       5 FEBRUARY 1983.

*DE 13
*   1 record deleted

*TY 1-12
    1.000 IDENTIFICATION DIVISION.
    2.000 PROGRAM-ID.
    3.000       CHAPTER1.
    4.000 AUTHOR.
    5.000       D D MCCRACKEN.
    6.000 DATE-WRITTEN.
    7.000       5 FEBRUARY 1983.
    8.000
    9.000 ENVIRONMENT DIVISION.
   10.000 INPUT-OUTPUT SECTION.
   11.000 FILE-CONTROL.
   12.000 SELECT LINE-OUT-FILE ASSIGN TO UT-S-PRINTER.

*   EOF hit after 12.000
*END
```

You've finally made all the necessary changes and log off...

```
!OFF
```

# MODULE 3-5

# Conditional Execution in EDIT

It is possible to use the EDIT IF command to evaluate a string selection in an EDIT file, and to modify subsequent EDIT processing of that file depending upon the "true" or "false" findings of the evaluation.

This type of processing operates upon the current selection criteria.

Example:

```
SE 1-10
IF/#/,1,1;DE;EI
```

means, for records 1 through 10, search beginning in column 1, and ending in column 1, for all records that have a pound sign in that column, and delete these records. The EI command terminates an IF block.

Let's see how this operates on an EDIT file:

```
!EDIT
EDIT B03 HERE
*BUILD TEMP0
    1.000 100-XXX-XXX
    2.000 126-456-742
    3.000
    4.000 #100-987-90
    5.000 #200-XXX-XXX
```

You can now try the IF command on the above file. Remember that EDIT commands follow the EDIT asterisk prompt:

```
*SE 1-5
*IF/#/,1,1;DE;EI
*  2 records deleted
*TY 1-5
 1.000 100-XXX-987
 2.000 126-456-742
 3.000
 EOF hit after 3.000
```

The records beginning with a pound sign have been deleted.

In the next example, you ask EDIT to look for two designated strings after rebuilding the file as TEMP01:

```
!EDIT
EDIT B03 HERE
*BUILD TEMP01
    1.000 100-XXX-XXX
    2.000 126-456-742
    3.000
    4.000 #100-987-90
    5.000 #200-XXX-XXX
*SE 1-5
*IF 2/XXX/;DE;EI
*   2 records deleted
*TY 1-5
    2.000 126-456-742
    3.000
    4.000 #100-987-90
*   EOF hit after 5.000
```

Now those records which had two occurrences of the string XXX have been deleted. In specifying two strings, you made use of the EDIT string selection expression. For more information about it enter HELP (EDIT) STRING SELECTION.

## EDIT Command Files

EDIT commands, including conditional execution commands, can be incorporated into an EDIT command file. Such a file can be given any legal name and is executed by using the EDIT READ command. In the following example, an EDIT command file named EDIT_NUMB is built:

First you build a file that contains a single, multi-line EDIT command: Then you create a short file consisting of numbers. When the EDIT command file is executed, it reverses the numbers.

```
*BUILD EDIT_NUMB
    1.000 IF/ /,1,1;/ /D;
    2.000 EL/1/,1,1;/1/D;/ /F/1/;TX;RL;
    3.000 EL/2/,1,1;/2/D;/ /F/2/;TX;RL;
    4.000 EL/3/,1,1;/3/D;/ /F/3/;TX;RL;
    5.000 EL/4/,1,1;/4/D;/ /F/4/;TX;RL;
    6.000 EL/5/,1,1;/5/D;/ /F/5/;TX;RL;
    7.000 EL/6/,1,1;/6/D;/ /F/6/;TX;RL;
    8.000 EL/7/,1,1;/7/D;/ /F/7/;TX;RL;
    9.000 EL/8/,1,1;/8/D;/ /F/8/;TX;RL;
   10.000 EL/9/,1,1;/9/D;/ /F/9/;TX;RL;
   11.000 EL/0/,1,1;/0/D;/ /F/0/;TX;RL;
   12.000 EI;TX
   13.000
```

Next, a short file consisting of numbers is created. This file will be acted upon by the command file.

```
*BUILD *INVERSE
*   EDIT stopped
    1.000 3412344
    2.000 123412346
    3.000 123448
    4.000
```

After building the file, you set the selection criteria to include the entire file:

```
*SE
```

Now you use the READ command to invoke the EDIT command file:

```
*READ EDIT_NUMB
IF/ /,1,1;/ /D;
EL/1/,1,1;/1/D;/ /F/1/;TX;RL;
EL/2/,1,1;/2/D;/ /F/2/;TX;RL;
EL/3/,1,1;/3/D;/ /F/3/;TX;RL;
EL/4/,1,1;/4/D;/ /F/4/;TX;RL;
EL/5/,1,1;/5/D;/ /F/5/;TX;RL;
EL/6/,1,1;/6/D;/ /F/6/;TX;RL;
EL/7/,1,1;/7/D;/ /F/7/;TX;RL;
EL/8/,1,1;/8/D;/ /F/8/;TX;RL;
EL/9/,1,1;/9/D;/ /F/9/;TX;RL;
EL/0/,1,1;/0/D;/ /F/0/;TX;RL;
EI;TX
    1.000 412344 3
    1.000 12344 43
    1.000 2344 143
    1.000 344 2143
    1.000 44 32143
    1.000 4 432143
    1.000  4432143
    1.000 4432143
    2.000 23412346 1
    2.000 3412346 21
    2.000 412346 321
    2.000 12346 4321
    2.000 2346 14321
    2.000 346 214321
    2.000 46 3214321
    2.000 6 43214321
    2.000  643214321
```

```
    2.000 643214321
    3.000 23448 1
    3.000 3448 21
    3.000 448 321
    3.000 48 4321
    3.000 8 44321
    3.000  844321
    3.000 844321
*   EOF hit after 3.000
*   47 strings changed
```

When you print out the file INVERSE, you see that the execution of the command file has reversed the numbers.

```
*TY
    1.000 4432143
    2.000 643214321
    3.000 844321
*   EOF hit after 3.000
*END
```

# MODULE 3-6

# Listing and Reviewing Files

SETUP: On occasion, you may want to clean up files in your account. You may find that some are no longer important for your records and should be deleted. In addition, you may need to make a quick check of how much disk storage space you still have available, of file backup dates, and file creation dates.

To receive a quick listing of files in your account, use the LIST command. Although LIST is a PCL command, it may be called from IBEX as L, from EDIT as L[IST], or from PCL as L[IST].

LIST prints a summary of disk and labeled tape storage. Enter HELP (PCL) LIST for more information.

In the following example, the command is called from IBEX. A word of caution: If you use the LIST command at the ! (bang), be sure to only key in the abbreviated form, L. Keying in LIST at the ! invokes a different LIST command, the IBEX LIST command.

```
!L
:MAILBOX    A           B           COMP        COMP_HISTORY
COM_WORLD   D           DISP_PRHD   FIGURES     FIGURES_1   LISTREVW
MOVING      MR_SMITH    NEWDEST     OC_MSG_S    SAMPLE1     SETUP
 ..  17 files listed
```

## Listing File Attributes

If you want a listing of files in your account with a summary of attributes for each file, use the A (attribute) option. Attributes listed include: file organization, file type, granules used, granules remaining, length of file, date last modified, and name of file.

```
!L(A)
15:02 SEP 08 '83 .MYACCT
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY DM    13     1     339 10:40 SEP 08 '83  :MAILBOX
KEY SE     1     0       4 13:14 SEP 04 '83  A
KEY SE     1     0      13 13:21 SEP 04 '83  B
KEY SE     1     0       4 07:53 AUG 21 '83  COMP
KEY SE     1     0      53 07:53 AUG 21 '83  COMP_HISTORY
KEY SE     1     0       5 14:40 SEP 03 '83  COM_WORLD
KEY SE     1     0       4 10:31 SEP 04 '83  D
KEY        3     0     203 13:26 SEP 03 '83  DISP_PRHD
KEY       18     0    1301 13:01 AUG 27 '83  FIGURES
KEY       19     0    1371 13:41 AUG 28 '83  FIGURES_1
CON        2     0      19 15:02 SEP 08 '83  LISTREVW
KEY        3     0     258 10:31 SEP 08 '83  MOVING
KEY SE     1     0       6 15:02 SEP 03 '83  MR_SMITH
KEY SE     1     0      21 15:07 SEP 03 '83  NEWDEST
KEY SE     3     0     113 10:39 JUL 31 '83  OC_MSG_S
KEY SE     1     0       6 15:02 SEP 03 '83  SAMPLE1
KEY SE     1     0       8 07:53 AUG 21 '83  SETUP
 ..  17 files,    71 granules listed
```

The example below shows a listing of files in a specified account called :MEMO.

```
!L .:MEMO
:END          :FMT
 ..  2 files listed
```

You can also list files with extended attributes (EA). Extended attributes include: last access date, backup date, and creation date.

```
!L .:MEMO(EA)
15:03 SEP 08 '83 .:MEMO
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY TM     2     0     107 10:55 AUG 11 '83  :END
    LAST ACCESS  07:52 AUG 21 '83
    BACKED UP    01:55 AUG 22 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      10:55 AUG 11 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDLO   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP
KEY TM     3     0     147 10:55 AUG 11 '83  :FMT
    LAST ACCESS  07:52 AUG 21 '83
    WILL EXPIRE   38676 DAYS FROM CREATION
    BACKED UP    01:55 AUG 22 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      10:55 AUG 11 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDLO   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP
 ..   2 files,    5 granules listed
```

In the previous example, the LIST command was called from IBEX. In the example below, the LIST command is called directly from PCL.

```
!PCL
PCL B03 here
<L
:MAILBOX    A           B           COMP        COMP_HISTORY
COM_WORLD   D           DISP_PRHD   FIGURES     FIGURES_1   LISTREVW
MOVING      MR_SMITH    NEWDEST     OC_MSG_S    SAMPLE1     SETUP
 ..  17 files listed
```

In the following example, extended attributes are requested for the file NEWDEST:

```
<L NEWDEST(EA)
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0      21 15:07 SEP 03 '83  NEWDEST
    LAST ACCESS  15:07 SEP 03 '83
    BACKED UP    01:31 SEP 05 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      15:07 SEP 03 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDLO   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP
```

## REVIEW Command (PCL)

The REVIEW command gives you a sequential listing of your files and also prompts you with a period after each file name. At that point you can choose to make several adjustments which affect your file listings. For example, you can see a summary of attributes or extended attributes. You can also skip files, delete them, and make copies.

```
!PCL
PCL B03 here
<REVIEW
:MAILBOX.
A.
B.
```

After the period prompt for files :MAILBOX, A, and B, you enter CR only, indicating that no additional information is needed.

```
COMP.A
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0       4 07:53 AUG 21 '83  COMP.
```

An A entered after the period prompt for file COMP produces the above listing of attributes. After the second period prompt for COMP, you enter CR to indicate that no additional information is needed.

```
COMP_HISTORY.EA
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0      53 07:53 AUG 21 '83  COMP_HISTORY
    LAST ACCESS  07:53 AUG 21 '83
    BACKED UP    01:57 AUG 22 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      14:41 AUG 03 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDLO   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP.JUMP SAMPLE1
```

An EA entered after the period prompt for file COMP_HISTORY produces the above listing of extended attributes. After the period prompt following BACKUP, you enter JUMP SAMPLE1, which causes PCL to jump to the designated file, skipping the intervening files.

```
SAMPLE1.EA
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0       6 15:02 SEP 03 '83  SAMPLE1
    LAST ACCESS  15:02 SEP 03 '83
    BACKED UP    01:32 SEP 05 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      14:55 SEP 03 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDLO   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP.END
 ..     5 files listed
 <
```

An EA entered after the prompt for SAMPLE1 produces a listing of extended attributes. Following the period prompt after BACKUP, you enter END. This ends the reviewing process, and returns you to the PCL command level.

The REVIEW command may also be used to specify an individual file:

```
<REVIEW COMP_HISTORY
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0      53 07:53 AUG 21 '83  COMP_HISTORY.EA
KEY SE     1     0      53 07:53 AUG 21 '83  COMP_HISTORY
    LAST ACCESS  07:53 AUG 21 '83
    BACKED UP    01:57 AUG 22 '83  ON  DP#D00121
    XTNSIZE =    2
    FITSIZE =   54
    CREATED      14:41 AUG 03 '83
    LSLIDE  =  510    SLIDE   =    1
    LRDL0   =    3    TDALVL  =    0    SPARE   =   50
    BACKUP.END
<END
```

## Recovering Files

SETUP: Inadvertently, you deleted a large number of records in a file. Is it possible to recover these lost records?

First get a listing with attributes of that file, which shows you when the file was last backed up, created, and modified. This information can help you recover the file.

Procedures will vary from site to site, but let's consider a site at which files are backed up daily, weekly, and monthly, using an odd-even rotation. All changed or new files are copied (backed up) to tape daily. Then all existing files are saved weekly, and finally monthly. If your file has been backed up, it may be possible for you to have the lost records restored. The restored version may not contain your most recently entered changes; this will depend on when the backup file was created.

Suppose you deleted a file three days ago, and now the boss asks to see that particular file. Can that file be recovered? There may be times when you can recover your file if the timing is right.

Find out what the procedures are at your site for backing up files. Then, if you need to recover a file you will be well informed. With your knowledge about file recovery, you may be able to recover the file your boss urgently requested. He'll be happy, and so will you.

# MODULE 3-7

# Changing File Organization

CP-6 file structures include keyed, indexed, consecutive, relative, random, IDS, and unit record. See the CP-6 Programmer Reference, CE40, for a detailed description of these types of files.

Perhaps the most commonly used types of files are keyed, consecutive, and indexed. Let's consider some examples in which it may be necessary for you to change the organization from one file type to another.

SETUP: You want to copy a file of error messages which exist in account :CENTRAL in a file named COBRA, and edit your copy. You decide to use EDIT.

```
!EDIT
EDIT B03 HERE
*E COBRA.:CENTRAL
      File COBRA.:CENTRAL is not a keyed file - limited updating - RP
      mode set
```

EDIT has informed you that COBRA is not a keyed file. You request to see the file at your terminal:

```
*TY
       1 %U1 is of type character. Illegal in a VIRTUAL statement.
       2 %U1 is not an array or common block.
       3 Common block %U1 exceeds previous allocation
       4 Equivalenced scalar %U1. Too many subscripts.
```

EDIT has provided abbreviated line numbers because this is not a keyed file. You decide to make a keyed copy for your own account:

```
!EDIT
EDIT B03 HERE
*COPY COBRA.:CENTRAL OVER MYFILE
```

By using the EDIT COPY command, you have asked EDIT to make a keyed copy of file COBRA. By default, your copy will start with line number 1.000, incremented by 1.

You can also make a keyed copy of a consecutive file using the PCL processor:

```
!PCL
PCL B03 here
<COPY COBRA.:CENTRAL ON MYFILE (ORG=KEYED)
```

You have copied file COBRA on to your own file "MYFILE" as an EDIT keyed file. Or, if you you want to designate the beginning line number:

```
<COPY COBRA.:CENTRAL ON MYFILE (LN=4)
```

You have specified that the input file COBRA is to be copied as an EDIT keyed file with beginning line number 4.000, i.e.

```
4.000 %U1 is of type character. Illegal in a VIRTUAL statement.
5.000 %U1 is not an array or common block.
6.000 Common block %U1 exceeds previous allocation.
7.000 Equivalenced scalar %U1.  Too many subscripts.
```

The options specified in parentheses in the above PCL commands are PCL output options. These options enable you to change file organization from one type to another. If you want to change file organization to indexed, you must use both the KEYLENGTH option to specify the number of characters for the key, and the KEYX option to specify the character position, as well as ORG = INDEXED.

When you build a file using the BUILD command, it will be an EDIT keyed file. Suppose you want to build a consecutive file. You could build a keyed file and then use the PCL copy command to change the file organization to consecutive, using either the NLN output option, or the ORG=CONSECUTIVE output option. You can also use the COPY ME command to build a consecutive file directly. See the module entitled Creating New Files (CE55) for an example of using the COPY ME command.

The principal benefit to the user of consecutive files over keyed files is a reduction in the amount of space required for the file, and a consequent reduction in the time required to traverse the file. So, for very large files which will be used in a sequential manner only, the consecutive file has an advantage.

The type of file organization you select will affect the results you get when using the PCL COPY command. Suppose you have two files, ALTA and DENA.

```
<COPY ALTA INTO DENA
```

If the files are not keyed, the records in ALTA are appended at the end of file DENA.

If, however, the files are keyed, the records in ALTA are "woven" into destination file DENA (perhaps replacing existing records). All keyed files always look like they have been sorted by keys.

# MODULE 3-8

# Changing File Access Attributes

What is file access? In the CP-6 system, the term fid is used as a name to refer to both files on disk or tape, and to devices such as terminals and printers. File access is access to a fid.

It is possible for you to specify the type of file access available to other users for disk or tape files in your account. This may be done by:

- Specifying SHARE and FUN attributes through the IBEX SET command
- Establishing access to users holding specified accounts, also by means of the SET command (ACCESS SET option).
- Using a PCL command to specify the desired PCL output option.

Let's look at some examples of each of the above.

```
!SET M$SI MYFILE,SHARE=ALL
```

This command assigns M$SI DCB to disk file MYFILE, which can be opened by all qualified users while you have it open.

```
!SET M$SI MYFILE,SHARE=IN
```

specifies that qualified users can open MYFILE for input only (while you have it open).

```
!SET M$SI MYFILE,SHARE=NONE
```

specifies that no sharing users can open the file while you have it open.

These SET command options remain in effect for the duration of that job or session, unless reset by the RESET command. You can, for example, start a job by allowing input from other users, and then use RESET to change to SHARE=NONE.

The SET command FUN option specifies whether an existing file is to be read or updated, or whether a new file is to be created.

```
!SET M$SI MYFILE,FUN=IN
```

The above means that MYFILE can only be read; no modification can be made to the file.

```
!SET M$SI MYFILE, FUN=UPDATE
```

means that MYFILE can both be read and modified.

```
!SET M$SI MYFILE,FUN=CREATE
```

means that a new file is to be created.

Now let's look at the SET command ACCESS option, which allows you to specify those accounts which may access your file:

```
!SET M$SI MYFILE, ACCESS=(ALL,READ)
```

means that any and all accounts may access MYFILE. You may also indicate a single account, or a number of accounts (called an accountlist), or a range of accounts specified by the use of wildcard characters, i.e.,

```
!SET M$SI MYFILE, ACCESS=(477?,READ)
```

means that any account beginning with 477 has access to MYFILE.

PCL output options may also be used to limit access to a fid. The PCL output options we are concerned with are ACCESS, ACSVEHICLES, READ, and WRITE.

| | |
|---|---|
| ACCESS | Specifies which accounts are permitted access to the file, and what control applies to those accounts. |
| ACSVEHICLES | Specifies which processors are permitted access to the file, and which controls apply to those processors. A processor is defined as any run unit with optional account specification. If the account is not specified, :SYS is assumed (BASIC, EDIT, etc.) |
| READ | Specifies which accounts can read records from a file. |
| WRITE | Specifies which accounts can write records to a file. |

Examples:

```
!PCL
<MOD MYFILE TO (READ=SAM)
```

means that MYFILE may be read only by account SAM and the creating account.

```
<MOD MYFILE TO (ACC=(?,NOLIST))
```

will make the file MYFILE visible only to the creating account.

SETUP: You want a data file to be given READ and WRITE access, but want to prohibit the use of PCL to copy all or portions of the file.

SOLUTION: You use the following SET options:

```
!SET MYFILE , ACCESS=(accountlist1,READ,EXEC),;
              ACCESS=(accountlist2,WRITE,UPDATE,DELR,EXEC),;
              ACSVEH=(PCL)
```

where acountlist1 is the READ accountlist, and accountlist2 is the WRITE accountlist. The use of EXEC implements the ACSVEH option, which is used to specify a processor or processors, (i.e., PCL), and the access permissions to be granted to them. In this case, PCL has been specified with no permissions. Therefore, the use of the PCL processor by the accounts specified is prohibited for the designated file.

Additional information concerning PCL file access output options and SET options is available from the HELP facility.

# MODULE 3-9

# Selecting Files

The CP-6 PCL processor provides the user with the capabilty to select files in a variety of ways. It is possible to select a range of files, or to select files by organization and type.

Suppose you have a number of files designated FILE-A to FILE-Z. You want to select only FILE-A to FILE-P (inclusive):

```
!PCL
PCL B03 here
<COPY FILE-A > FILE-P
```

This will copy the designated range of disk files to your terminal (the default designation). The range designated is sorted alphabetically.

If the files are not in your account, the account must be specified:

```
<COPY FILE-A > FILE-P FROM .CENTREX
```

When using a tape file, you may want to copy all files physically located between two files.

```
<COPY LT#1489/FILE-A > FILE-P (PHY) TO MYFILE
```

The COPY command above designates a physical range, i.e., the files copied from tape 1489 will be those located physically between FILE-A and FILE-P. The range is inclusive, so FILE-A and FILE-P are included.

The examples above show the use of source parameters to designate a range of files. For more information, see the range source component in CE40.

The range source component can also be used to specify a range of files selected by password:

```
<COPY FILE-A..QBL > FILE-P
```

The files selected must all have the password "QBL", or be un-passworded.

Further options can be added to designate additional criteria for selecting such as file type or file organization:

```
<COPY FILE-A..QBL > FILE-P (ORG=K)
```

Now, in addition to the other criteria, the files selected must be keyed.

# MODULE 3-10

# Wildcarding

Wildcarding is one of those CP-6 features that you wonder how you ever got along without. Simply stated, wildcarding allows the wildcard character '?' to be used in place of a string of zero or more characters. Several CP-6 processors allow the use of wildcard characters, although not all of them allow exactly the same syntax.

PCL's use of wildcarding is a good example. The wildcard character is allowed to replace any characters in file names. For example, the construct 'L FOR?' will list all files in the user's current file management account whose names begin with 'FOR' and end in anything. PCL's response might be something like

```
    FOR     FORTRAN     FORTRAN_TEST
```

Note that 'FOR' satisfies the request since the wildcard matched a null string of characters. Similarly, the request 'L ?HOPE.ZED' will list all the files whose names end in 'HOPE' from the ZED account.

If your site runs a production shop, this makes management of a library of source files and run units much easier. If you establish a naming convention using prefixes or suffixes to identify the parts of a system, PCL's wildcarding feature can be used to list, copy, or delete them without having to build a standard file. For example, a facility might use the following suffixes to identify files in account ZED:

```
    _SIpn - identifies a Source Input file requiring compiler 'p';
            if there are multiple parts to the source file, 'n' is
            used to identify them, 1 through 9.
    _OUn  - Identifies Object Unit (optional 'n')
    _CRU  - A job file to create the Run Unit.
    _HELP - A HELP file to explain the program's usage.
```

The command 'L EXTRACT?.ZED' therefore would list all the pieces of the EXTRACT program.

Using wildcards in multiple places in a file-name fragment allows you to search an account for a file that you don't quite remember the name of. 'L ?PROG?', for example, might find files with names of 'COPYPROG', 'PROGO1', 'PROG', and 'MYPROG_SI'.

Finally, the wildcard allows you to delete all the files in your account. This is done by using DEL ?, a PCL command which can also be given at the IBEX prompt (!). This deletes all the files in your account (after asking permission, of course).

## Abbreviating Account References Through Wildcarding

Through use of wildcarding, account access attributes may be set to limit the accessibility (READ, WRITE, NOLIST, etc.) of individual files to account groups.

By means of illustration, assume an installation has assigned accounts as follows:

```
RD002A00
```

```
Where:   RD    Represents the department code
         002   Represents the project code
         A     Represents the users job title
         00    Represents the programmer number
               (Only project leaders receive a 00 number)
```

If the following 'MODIFY' command were issued:

```
!MOD FID TO (ACC=(RD?,READ),ACC=(RD?00,WNEW),ACC=(?,NOLIST))
```

All accounts with the 'RD' prefix would have 'READ' access, those accounts with an 'RD' prefix and a '00' suffix would have 'WNEW' access (the ability to write new records), and no accounts would be able to 'LIST' the file , except for those with 'READ' or 'WRITE' access and the account within which the file resides.

This technique might be particularly useful when utilized within a software factory environment, where programmers are given separate accounts for each project to which they're assigned. Similarly, in an academic environment this might apply where students are given an account for each computer-utilizing course in which they're enrolled.

# MODULE 3-11

# Maintaining File Accounts

In order to remember what the files in your account contain, it is a good idea to name your files in relation to their contents.

Suppose you have a FORTRAN program in a file named FORT. If it is absolutely necessary for you to have permanent object unit or run unit files (see CONSERVING DISK SPACE, in this module), it is sometimes desirable to name your object unit file FORT_OU, and your run unit file FORT_RU. This has several advantages. When you list your files, you will immediately be able to see which files are related to program FORT. If you want to delete all of the files related to program FORT, this can be done in a single command (provided that you don't have any other files starting with FORT). Note that the DELETE command with a wildcard character (?) is a PCL command, and must be taken from PCL or IBEX; the EDIT DELETE command does not allow the use of a wildcard character.

```
!DELETE FORT?
DELETE FORT?.MYACCT ?YES$
FORT        FORT_OU      FORT_RU
     3 files,     9 granules deleted
```

When you enter the command, PCL requests that you confirm the DELETE action, which you do by entering YES$ as shown above.

## File Types

Your files are all assigned a file type automatically by the processor you are operating in. The file type is a two-letter designator, which can be modified to meet your needs if necessary. For example, you could assign all files which contain memos for the office a file type MM. This way you could quickly get a listing of all memos you wrote just by specifying the file type MM.

> You build four files and assign file type ZZ to FILE1 and FILE2. File type YY is assigned to FILE3 and FILE4. Now you can list only file type ZZ's, or only file type YY's.

```
!EDIT
EDIT B03 HERE
*TYPE ZZ
*BUILD FILE1
    1.000 Twas brillig and the slithy toves
    2.000
*BUILD FILE2
*   EDIT stopped
    1.000 Did gyre and gimble in the wabe
    2.000
```

```
*TYPE YY
*BUILD FILE3
*   EDIT stopped
    1.000 All mimsy were the borogroves
    2.000
*BUILD FILE4
*   EDIT stopped
    1.000 And the mome raths outgrabe
    2.000
*END


!L ?(TY=ZZ)
FILE1       FILE2
 ..   2 files listed
!L ?(TY=YY)
FILE3       FILE4
 ..   2 files listed
```

You can use file types to collect files with related contents and print them.

```
!C ?(TY=ZZ),?(TY=YY)
Twas brillig and the slithy toves
Did gyre and gimble in the wabe
All mimsy were the borogroves
And the mome raths outgrabe
```

After running a short BASIC program, you can get a listing with attributes of FILE5. Notice that SB (Source Basic) is listed under file type (column 2). This is the file type assigned by the BASIC processor. You can copy FILE5 into the MEMO account with an added option "NFA". This requests that no file attributes be copied.

```
!BASIC
BASIC C02 HERE
>10 REM A RATHER SHORT PROGRAM.
>20 END
>SAVE FILE5
FILE5 SAVED
>SYS
!L FILE5
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SB     2     0       2 12:21 SEP 14 '83  FILE5
!COPY FILE5 TO FILE5.:MEMO(NFA)
 ..COPYing
!L FILE5.:MEMO
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY        1     0       2 12:21 SEP 14 '83  FILE5
```

A listing of FILE1-FILE5 shows you all the file types which have been assigned so far.

```
!L (A) FILE?
12:28 SEP 14 '83 FILE?.MYACCT
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY ZZ     1     0       1 12:16 SEP 14 '83  FILE1
KEY ZZ     1     0       1 12:17 SEP 14 '83  FILE2
KEY YY     1     0       1 12:17 SEP 14 '83  FILE3
KEY YY     1     0       1 12:17 SEP 14 '83  FILE4
KEY SB     2     0       2 12:21 SEP 14 '83  FILE5
 ..   5 files,     6 granules listed
```

You can request that these files be tagged as file types SE (Source Edit) again, if you prefer.

```
!MOD FILE? TO (TY=SE)
FILE1       FILE2       FILE3       FILE4       FILE5
!L (A) FILE?
12:29 SEP 14 '83 FILE?.MYACCT
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SE     1     0       1 12:28 SEP 14 '83  FILE1
KEY SE     1     0       1 12:28 SEP 14 '83  FILE2
KEY SE     1     0       1 12:28 SEP 14 '83  FILE3
KEY SE     1     0       1 12:29 SEP 14 '83  FILE4
KEY SE     1     0       2 12:29 SEP 14 '83  FILE5
```

## Star Files

CP-6 star files are temporary job files that are automatically deleted at the end of the job. They can be created by the user or the system. A star file is designated any time you use a file name which begins with an asterisk (*) or "star".

Star files are handy for use as scratch files or quick test case files that you don't want to keep around. How many times have you tried to create file TRASH or SCRAP or JUNK and had the monitor reply that the "FILE EXISTS" because you forgot to delete it from the last time? Imagine the wasted system file space that these files occupied. Getting into the habit of making them star files automatically keeps your account clean.

You can build a star file with PCL by COPYing something TO (or OVER or INTO) *filename or in EDIT by BUILDing *filename. "filename" is any legal CP-6 file name, but names longer than one character are recommended. "*TRASH", "*SCRAP", and "*JUNK" are popular names. Any number of users logged onto the same account can create star files with the same names simultaneously, since the file names are not entered into your directory. You can't create a star file in someone else's account, nor can you look at someone else's star file(s), even though PCL will accept an account specification on a star file name without complaint (e.g., COPY ME TO *SCRAP.YOURACCT is equivalent to COPY ME TO *SCRAP).

If you create a star file and later decide you want to keep it, simply use PCL to COPY it to a permanent file before you log off. You cannot use MOD to rename a star file as a permanent file, though.

Star files are also handy since they don't take up space in your account; they are charged against temporary file space and reside in "public" disk space.

Star files won't be listed or deleted by PCL commands "L" or "DEL ?", since those commands scan the specified file directory. You can list your accumulated star files and delete them by using the PCL commands "L *?" and "DEL *?".

## Running Out Of Space

You will know when you have run out of disk space when a message suddenly pops up on your terminal. If this happens, go to the person in charge of allocating more disk pack set storage space. If you are told that you cannot be allocated more space, you have three options:

1. Delete all files in your account which are of little or no value to you. They just take up valuable disk space.

2. If you need all the files in your account, have them copied onto tape, so that you can free some disk space in your account.

3. Compress or purge unneccessary information from your existing files, as described below.

## Conserving Disk Space

There are several techniques that can be used to conserve disk space, and the techniques differ slightly depending on what the file contains.

### Object Units

In general, object units need not be retained on disk once the associated run unit has been created. (This is because an object unit can ALWAYS be recreated by recompiling the source unit.) If an object unit contains commonly used routines, it should be merged into a library file (with LEMUR) with other commonly used routines; then the individual object unit should be deleted. If it is absolutely necessary to retain individual object units on disk, space may be saved by compiling WITHOUT full debug schema...(i.e. MINI schema, the compiler default). It is also possible to save even more space by requesting absolutely NO debug schema with the NSCHEMA option.

### Source Programs

Source programs (and, in general, any file containing all ASCII characters) may be forced to occupy less disk space by employing several methods. One method is to simply strip the EDIT line number keys from the file by copying the file over itself with PCL using the (NLN) option.

Even more space may be saved by employing a method called file compression. A source file may be compressed by copying it over itself with PCL and using the (C) option:

```
<C fid OVER fid(C)
```

Of course, the most space may be saved by copying the file over itself:

```
<C fid (NLN,C)
```

NOTE: These methods apply only to source files. Application of these methods to object units, run units, workspaces or binary data files may render them unusable or cause them to occupy MORE disk space than before.

NOTE: Once a file has been compressed, you may use EDIT only to EXAMINE the file. If you wish to make a modification to a file that has been compressed, you must rekey it with EDIT.

```
!E fid
***   File fid is not a keyed file - limited updating - RP mode set
*C fid
 ....NOW you can use EDIT to update the file
```

## Run Units

You can conserve disk space occupied by run units by always using the NODEBUG option when LINKing a run unit. This will eliminate all but the most primitive debug schema from the run unit. Even more space may be saved by using the following. Please note, however, that MPUR.X may not exist at all sites.

```
!LINK *G OVER rufid
!MPUR.X rufid,NO OVER rufid
```

## Other Files

BASIC and APL workspaces, binary data files, data files that require their keys, and databases should not have the above space-saving techniques applied to them.

## Keyed Data Files

Keyed data files that are updated frequently (or, in particular, have records deleted from them frequently), can begin to accumulate space and occupy more disk space than necessary (space where records and keys are deleted are retained for possible future re-use). To recover this space, you can copy a file over itself with PCL (provided enough space is available on your packset). If you don't have enough space, you can copy the file to tape, delete the file from disk, and restore the file from tape. Of course, you should NOT strip the keys from the file in either step, as the programs that access these keyed data files almost always depend on the keys in the file to get their job done correctly.

## Using Files in Other Accounts

SETUP: Suppose you are working on a project with another person who has a file named PROJECT1 which you need to complete a section of your work. How can you put that file into your own account and then work on it?

All you need to know is the file name, which is PROJECT1 and your co-worker's account name which is COMPWORK. With this information you can copy PROJECT1 into your account:

```
COPY PROJECT1.COMPWORK INTO MYPROJECT
```

You must specify a file name to which PROJECT1 is copied. Also, be sure to insert a period between the file name and the account name.

Or if you like, you can request a formatted version of PROJECT1 which is printed at a line printer so that you have a hard-copy:

```
TEXT PROJECT1.COMPWORK TO LP
```

So with the file name and account name of a particular person, you have access to his/her files in various forms, whatever is most convenient for you.

# MODULE 3-12

## Printing Files on the Lineprinter

SETUP: You own a file called CHIMERA, and want to distribute copies of CHIMERA to various people. Twenty feet from your terminal is a lineprinter which belongs to workstation GRYPHON. Your account, by default, uses this lineprinter. At the other side of the city is another workstation called PHOENIX, which also has a lineprinter. The copies of CHIMERA that you want are:

- 1 copy for your notebook
- 2 copies for your manager
- 1 copy on 8 1/2 by 11 for the home office (to be mailed)
- 3 copies for the programming staff at workstation PHOENIX

PROBLEM: How can you make and deliver all these copies with a minimum of hassle and legwork?

SOLUTION: You create the following XEQ file:

```
!COPY CHIMERA TO LP
```

First, you make a copy for yourself, using the simple form of the copy command. You copy it to LP because you know that the system is smart enough to realize that LP means the printer that's twenty feet down the hall.

```
!LDEV LP03 LP@GRYPHON,COPIES=2
```

Next, you create a logical device for the lineprinter at GRYPHON. Anything sent to LP03 will be printed twice. These will be the copies for your manager.

```
!LDEV LP04 LP@GRYPHON,FORM='LONG'
```

Now you create another logical device that has form LONG. The system manager at your site has told you that form LONG is 8 1/2 by 11 paper, and the operations staff already knows how to mount it.

```
!LDEV LP05 LP@PHOENIX,COPIES=3,TITLE='CHIMERA by YOURNAME'
```

Now you create yet another logical device, one that will print three copies at workstation PHOENIX. You put a title on each page that has your name and the name of the file.

```
!COPY CHIMERA TO LP03
!COPY CHIMERA TO LP04
!COPY CHIMERA TO LP05
!PRINT LP03
```

Now that you have created your logical devices, all that you have left to do is copy CHIMERA to each. You use the PRINT command to specify that the LP03 copies are to be printed immediately.

EPILOGUE: Two copies of file CHIMERA are immediately printed at the lineprinter near your office. The LONG form is not immediately printed, because the system manager has instructed the operations staff to mount form LONG only when a sizeable amount of output has accumulated for that form. Eventually you do obtain this copy and send it to the home office. The three copies sent to workstation PHOENIX are printed as planned.

COMMENTS: In addition to sending output to lineprinter devices, the COPY command has many other uses.

A logical device is a user-created profile for a device. It should not be confused with the actual physical device on which the output (in the case of lineprinters) is printed.

Using the LDEV command, you can set certain attributes for a printer, such as number of copies, the type of form (paper), or title to be added. You may also set printing width, lines per page, etc. See the LDEV command in the Programmer Reference (CE40), or in the IBEX HELP facility.

Workstations are, by and large, physical areas where work takes place on the computer. Workstations are, however, defined as part of the system software, and there is no reason why a remotely located printer can not "belong" to the same workstation as the main computer facility. Each user, regardless of physical terminal location, has a "workstation of origin", which determines the default lineprinter, cardreader, and other system devices.

# MODULE 4-0

## Section 4 - Creating and Running Programs

Module 4-1 in this section discusses and gives examples of IBEX programming. Module 4-2 presents an actual CP-6 session, including examples incorporating FORTRAN, COBOL, and BASIC programs. Module 4-3 presents an example of the use of FPL on the CP-6 system. Module 4-4 discusses and gives examples of LINK overlay programs.

# MODULE 4-1

# IBEX Programming

Interactive and Batch Executive (IBEX) commands identify the user job, the tasks to be performed by the job, and the resources required by the job. IBEX commands also control interactive terminal operations. All batch jobs and interactive sessions require the use of IBEX commands.

To obtain a list of available HELP topics for IBEX, enter:

```
!HELP (IBEX) TOPICS
```

## IBEX Programming Conventions

The following conventions apply to IBEX programming:

- Only one IBEX command may be used per line. If the command and its options extends over multiple lines, a semicolon must be used at the end of each line, except the last.

- IBEX command labels (see IBEX Command Labels in this module) may not be used with the following commands:

  ```
  !JOB
  !RESOURCE
  !INCLUDE
  !DEFAULT
  ```

  Labels may be used with all other IBEX commands.

- Comments may be included as part of a series of IBEX commands and may appear anywhere arbitrary blanks are allowed. Comments must be enclosed in double quotes ("), but the closing quote is not required at the end of a line. If a comment extends over multiple lines, each line is treated as a separate comment.

- The following IBEX commands may not be indented (i.e., no blanks are allowed after !):

  ```
  !JOB
  !FIN
  !EOD
  !DEFAULT
  !INCLUDE
  !ASC
  !BIN
  !NCTL
  ```

  For all other IBEX commands, indentation is allowed; i.e., any number of intervening blanks may appear between ! and the command, as long as the command appears on a single line.

## Executing Programs

You can initiate program execution through three IBEX commands:

| | |
|---|---|
| rununit | Fetches and initiates execution of a run unit |
| RUN | Links an object file (or files), and fetches and initiates execution of the resulting run unit. |
| START | Fetches and initiates execution of a run unit. |

Examples:

```
!FORT_RU.
```

This rununit command fetches and starts execution of the run unit FORT_RU.

```
!RUN FORT_OU
```

This command links the object file FORT_OU, and fetches and initiates execution of the resultant run unit.

```
!EXAMPLE SOURCE,UPDATE INTO OUTPUT,LIST
```

This rununit command fetches the run unit EXAMPLE into memory from :SYS and starts execution using SOURCE as the source input file, UPDATE as the run unit update file, OUTPUT as the run unit output file, and LIST as the run unit listing file.

```
!START EXAMPLE
```

Starts execution of the run unit EXAMPLE without an associated debug processor.

```
!START EXAMPLE UNDER DELTA
```

Starts execution of the run unit EXAMPLE under control of DELTA.

```
!START EXAMPLE UNDER MYBUG
```

Starts execution of the run unit EXAMPLE under control of the debug processor named MYBUG.

## Invoking Language Processors

The commands used to invoke language processors are IBEX rununit commands.

Examples:

```
!COBOL
```

invokes the COBOL compiler and sets the default options for COBOL.

```
!COBOL MYFILE OVER MYFILE_OU
```

invokes the COBOL compiler with the default options, using MYFILE as a source file and MYFILE_OU as the object unit file.

```
!FORTRAN FILE_X OVER *MYFILE
```

invokes the FORTRAN compiler with the default options for FORTRAN, using FILE_X as a source file and designating a temporary star file, *MYFILE, as the object unit file.

For information concerning the various compiler invocations and options, use HELP, i.e.,

```
!HELP (COBOL) COMPIL

!HELP (COBOL) COMPILER_OPTIONS

!HELP (FORTRAN) COMPIL

!HELP (FORTRAN) OPTIONS
```

etc.

## Interrupt Processing

You can interrupt processing of an activity by entering <CNTL><Y>. This suspends the current process, and IBEX initiates interrupt mode in which you are prompted with a double bang (!!). If you then respond by entering an IBEX command, this command will be processed, unless processing will result in making the interrupted activity unresumable and the PROTECT command has been issued earlier in the job stream.

For more information, enter:

```
!HELP (IBEX) PROTECT
```

## Allocating Resources and Establishing Service Limit

The following IBEX commands allow you to allocate resources and establish service limits:

| | |
|---|---|
| ACQUIRE | Requests and allocates additional resources for your use if available (only available online). |
| ORESOURCE | Establishes resources and global limits required to run an online job. |
| RELEASE | Releases or deallocates previously allocated resources. |
| RESOURCE | Establishes resources and global limits required to run a batch job (must immediately follow the JOB command if it exists.) |
| LIMIT | Sets maximum values for various system services required by a job. |

The ACQUIRE, ORESOURCE, and RESOURCE commands are used with the Resourcelist Component. Enter !HELP (IBEX) RESOURCE for more information.

Examples:

```
!ACQUIRE MT=2
```

indicates that two additional tape drives are to be allocated to this job.

```
!ORESOURCE MEM=256
```

indicates that 256K words of memory are to be allocated for this online session.

```
!RELEASE DP01,DP02
```

releases two disk drives identified as DP01 and DP02.

```
!RESOURCE DP(100MEG)=4,LP(OVERPRINT)
```

indicates that the current job will require four 100MEG disk packs and a line printer with an overprint capacity.

```
!RESOURCE CP(BIN),MEM=4,TIME=04:20:13
```

indicates that the current job is to be allocated a binary card punch, 4K of memory, and that the execution time is not to exceed 4 hours, 20 minutes, and 13 seconds.

```
!RESOURCE ACC(MYACCT,YOURACCT)
```

indicates that the packset containing the accounts MYACCT and YOURACCT is a public packset for this job.

```
!RESOURCE DP#LIBSET(PUBLIC)
```

allocates the packset LIBSET as a PUBLIC packset for this job.

```
!LIMIT TIME=:05,LO=125
```

indicates that the current job is not to use more than 5 seconds of execution time or produce more than 125 pages of printed output.

```
!LIMIT STEP,DO=50,PDIS=18,TDIS=24
```

indicates that this LIMIT command is to apply to the following job step only, with these limits in effect: 50 pages of diagnostic output, 18 permanent disk granules, and 24 temporary disk granules.

## Execute Files

An execute (XEQ) file can contain IBEX commands, run unit calls, and data. It provides a convenient method of executing a frequently used sequence of commands.

The XEQ command is used to initiate execution of an XEQ file. The ECHO command controls printing of the commands in the file at your terminal or output destination as the commands are processed. DONT ECHO disables ECHO.

Examples:

```
!XEQ RUNJOB(START=10,P,'/*'=' ')
```

inserts the disk file RUNJOB starting at record 10 into the command stream. In addition, each record of the file is printed on the output destination, and each occurrence of the string '/*' is converted to a blank.

```
!XEQ FORTCOMP(TEST,COMMON=GLOBAL)
```

prints each record of the file FORTCOMP that is altered by the replacement specification COMMON=GLOBAL. The file is not executed.

## Batch Jobs

Batch files are used to submit jobs to the system batch queue for execution. You can submit a batch job from a card reader, an interactive terminal, or a running batch job.

If the job is submitted from a card reader, the command stream can be contained in the card deck, which is converted into a job file, or the command stream can be read from a pre-stored job file.

If the job is submitted from an interactive session, the command stream may be contained in a pre-stored job file, or may be created from input typed at the terminal.

A pre-stored job file may be built through EDIT, PCL, or a user program. A record in a job file may not exceed 256 characters.

When a batch file is submitted for batch processing, it is queued, scheduled, and executed. The system schedules jobs for execution based on priority and resource requirements. You can influence scheduling of your job by assigning a higher or lower priority than the one assigned by the system through default.

You define a batch job and its priority through the JOB command, and then submit the batch job with the BATCH command. Once a batch job is submitted, the status of the job can be determined through the CHECK and NOTIFY commands. The job can be cancelled through the CANCEL command.

At the time a batch job is submitted for processing, you can replace values in selected strings and fields in the file being submitted. This data substitution feature is described under Command Files in this module.

| | |
|---|---|
| JOB | defines the beginning of a batch job, and must be the first command in each job command stream |

BATCH specifies one or more files that are to be submitted for batch job execution; can include data replacement specifications.

PRIORITY establishes the default priority for a job. The default is utilized if a priority is not specified on the JOB command.

Examples:

```
!JOB MYACCT,JOHN,MYPASSWORD ORDER,NRERUN
```

defines a batch job from account MYACCT, user-name JOHN with password MYPASSWORD. Also indicates that this job is to be scheduled in the order submitted among other jobs which specify the ORDER option in this account, and that the job is not to be reinitiated automatically if the system should crash.

```
!JOB MISACCT,GENERAL
```

defines a batch job from account MISACCT with user name GENERAL.

```
!BATCH RUNEXAMPLE(P),COBOLCOMP('YOURACCT'=MYACCT)
```

places the jobs RUNEXAMPLE and COBOLCOMP into the batch queue. The job RUNEXAMPLE will have all records of its command stream displayed as they are submitted for execution, and when the job COBOLCOMP is submitted for execution, all occurrences of the string YOURACCT will be replaced by the field MYACCT.

```
!BATCH NEWJOB
```

places the file NEWJOB into the batch queue for execution.

The system responds to the BATCH command by assigning each batch job a system identification (sysid) and sending the following message for each batch job submitted to the user's output destination:

'JOB sysid SUBMITTED current time and date'

Examples:

```
!CANCEL EXAMPLE
```

deletes the job named EXAMPLE from the batch queue.

```
!CANCEL 389
```

deletes the job with the sysid of 389.

```
!CANCEL 390 (OUTPUT=NO)
```

deletes the job with the sysid of 390, and deletes its output.

## Command Files

Both XEQ and batch files are command files, files that contain IBEX commands that control execution. As such, they share the following common characteristics and capabilities.

### Data Replacement

Prior to submitting a command file for execution, you can replace existing values with new ones through the use of the Replacement Component, which may be used with the BATCH, XEQ, XMIT, DEFAULT, and GLOBAL commands. This can be used to:

- define and modify data replacement parameters for a single BATCH or XEQ file (through the DEFAULT command).
- define and modify data replacement parameters for all BATCH and XEQ files in the session (through the GLOBAL command).
- override DEFAULT and GLOBAL data replacements for a file by specifying data replacement parameters as part of the BATCH or XEQ command that submits the file.

Through data replacement parameters, you can specify which string or field is to be changed, and what the new value is to be. Every occurrence in the file of the specified field or string will be changed.

A field is a contiguous set of nondelimiter characters bounded on either side by a delimiter character, or by the left or right record boundary. Nondelimiter characters and some sample delimiter characters are:

| Sample Delimiters | Nondelimiters |
|---|---|
| ( ) + | A-Z |
| ß ' ' | a-z |
| ' = | 0-9 |
| . / | - (hyphen) |
| ; | ? $ * : |
| | _ (underscore) |

The list of nondelimiters is complete; anything else is considered a delimiter.

A string can be part of a field, contiguous parts of two fields, or one or more contiguous fields. A string is designated in replacement equations by enclosing it in apostrophes.

Data replacement parameters are entered as replacement equations in BATCH, XEQ, DEFAULT, and GLOBAL commands. The replacement equations have the same syntax regardless of which command they appear in.

Examples:

1. Field replacement: ABC=DEF
   Command file content: ABCABC ABC ABCABC
   Modified command file: ABCABC DEF ABCABC

2. 'String' replacement: 'ABC'=DEF
   Command file contents: ABCABC ABC ABCABC
   Modified command file: DEFDEF DEF DEFDEF

In example 1, the value to be replaced is the field ABC; in example 2 the value to be replaced is the string ABC.

If a string or field is identified for replacement more than once, (i.e., the field or string specified on the left-hand side of the replacement expression is specified in more than one command) it will be modified only once in accordance with the following three rules of precedence:

1. Within a job command stream, XEQ and BATCH command data replacement specifications have precedence over DEFAULT and GLOBAL replacements.

2. Within a session, GLOBAL data replacement specifications have precedence over DEFAULT replacements. Also, a later GLOBAL specification has precedence over an earlier one.

3. Within a file, a later DEFAULT replacement specification has precedence over an earlier one.

Examples:

```
!DEFAULT abc=DEF
```

All subsequent occurrences of the field abc will be replaced by the field DEF.

```
!DEFAULT 'NOT'=NON,EQUAL=GREATER,'o'=' '
```

All subsequent occurrences of the string NOT will be replaced by the field NON, the field EQUAL will be replaced by the field GREATER, and the string o will be replaced by a blank.

```
!DEFAULT DELETE
```

deletes all previous DEFAULT replacement specifications.

```
!GLOBAL AND=OR,DO=FOR
```

causes the field OR to replace the field AND, and the field FOR to replace the field DO.

```
!GLOBAL DELETE DO
```

terminates substitution of the field FOR for the field DO.

## IBEX Command Labels

Command labels provide you with the means to direct branching. You can precede most IBEX command with a label in the format:

```
!label: command
```

Note that a blank must separate the colon from the command.

The value for a label is a 1 to 31 character alphanumeric name. The following characters are allowed (at least one character must be non-numeric):

```
A-Z
a-z
0-9
$ _ # @
```

You direct branching to the labeled command by using a GOTO command.

Example:

```
!IF OK THEN GOTO SAVIT
!SAVIT: OUTPUT 'PROCEDURE OK'
```

If the preceding program was terminated with a M$EXIT, skip to the command labeled SAVIT, which outputs the message 'Procedure OK'.

```
!IF A=B THEN GOTO ENDLBL
```

If variable A is equal to variable B, skip to the command labeled ENDLBL.

## Command File Logic - Conditional Execution

If IBEX is reading commands from a command file (i.e., a command stream created by an XEQ command or a batch file), the flow of control from one job step to the next may be affected by the IF, LET, and GOTO commands.

These commands operate on the Step Condition Code (STEPCC), command variables, and command labels, and give you the ability to conditionally modify command stream flow.

The IF command allows you to establish the conditions for affecting a change in the command stream flow (the IF clause) and to redirect the flow (the object clause). The IF clause establishes a relational test which, if true, causes the object clause that follows it to be executed.

The object clause consists of any IBEX command.

The LET command establishes values; the GOTO command branches forward or back within the command stream, and the QUIT command terminates processing of the command stream.

## STEPCC

STEPCC is a system variable that is modified by the system each time program processing terminates. You can also set STEPCC directly through the LET command. STEPCC's contents reflect the nature of program termination.

STEPCC is a one-byte field in the JIT that can take on values from 0 to 511, and can be set by the program using M$EXIT, M$ERR, and M$XXX monitor services. CP-6 processors use the values 0, 4, and 6 for exit control. The default behavior of each of the exit monitor services is to set STEPCC as follows:

| Monitor Service | Default STEPCC Setting | Termination Type |
|---|---|---|
| M$EXIT | 0 | Normal |
| M$ERR | 4 | Error Condition |
| M$XXX | 6 | Program Abort |

Note that the STEPCC setting is under the programmer's control and can be set to any value between 0 and 511 for any case.

### Program Exit Method

The form of exit (M$EXIT, M$ERR, or M$XXX) taken by the program at the last job step determines the settings of the OK, ERROR, and ABORT flags. These one-bit indicators are either on or off. The form of exit in relationship to the indicators and the type of termination is as follows:

| Monitor Service | Flag Setting | | | Termination Type |
|---|---|---|---|---|
| | OK | ERROR | ABORT | |
| M$EXIT | 1 | 0 | 0 | Normal |
| M$ERR | 0 | 1 | 0 | Error Condition |
| M$XXX | 0 | 0 | 1 | Program Abort |

The flag settings always refer to the type of exit chosen, independent of the value of STEPCC. Both STEPCC and the exit flags (OK, ERROR, ABORT) may be tested with the IF command and appropriate action taken depending on the value or setting.
Note that these settings of STEPCC by M$EXIT, M$ERR, and M$XXX are defaults only; the exiting program may specify a different value for STEPCC for any of these services.

STEPCC's contents can be used as part of the relational expression in an IF clause. The relational expression that includes STEPCC can test:

1. Whether STEPCC is

   Equal to
   Greater than
   Greater than or equal to
   Less than
   Less than or equal to
   Not equal to

   a command variable or a constant, or

2. Whether a command variable bears any of these relationships to STEPCC.

The object clause of the IF command will direct processing based on whether the test is true.

Examples:

```
!IF ERROR THEN QUIT
```

means that if the preceding program terminated with a M$ERR, abort the remainder of the job.

```
!IF STEPCC>VAR THEN LET STEPCC=0
```

If the value of STEPCC is greater than the value of the command variable VAR, set STEPCC=0.

```
!LET STEPCC=A
```

sets STEPCC equal to the value of the command variable A.

```
!LET C=3
```

sets command variable C equal to 3.

```
!LET X=STEPCC
```

sets command variable X equal to the current contents of STEPCC.

## IBEX Expressions

STEPCC as used above is an example of the use of an IBEX expression. To obtain more information, enter:

```
    !HELP (IBEX) EXPRESSION COMPONENT
```

The expression component is a numeric or string representation that may combine a unary modifier, a primary expression, and an operator in one expression. The expression component may form part of the IF, LET, and OUTPUT commands.

Examples:

```
!LET A='123'
!LET B=100
!LET C=A+B
```

Everything is stored as strings. Thus 123 is the same as '123'. Conversions are made as required by the operation type. The final value of C is '223'.

```
!LET A='123'
!LET B=A || ' is a string.'
```

This example illustrates the use of the concatenation operator. The value B is '123 is a string.'.

```
!IF $DAY='TUE' THEN XEQ FID
```

This example shows the use of the function $DAY. If today is Tuesday, then the file FID will be XEQed. This would be useful for running jobs that need only run on specific days or dates.

```
!LET A=1
!LET B=2
!LET C=A & B
```

This example shows the use of a TRUTH VALUE operator. Any NUMERIC values may be operated on by AND, OR and NOT. A numeric value is considered TRUE if greater than zero, and FALSE if zero or negative. Thus in this example A and B are TRUE and the result of A & B is TRUE also.

```
        .
        .
        .
    !LET A=$INPUT('FULL, PARTIAL or NO SAVE tonight? F/P/N')
    !IF $EOF THEN QUIT   "Esc F input. Stop processing"
    !IF A='' THEN GOTO NEXT   " Null input. Assume N"
    !IF $SUBSTR(A,0,1)='F' THEN BATCH FULL_SAVE
    !IF $SUBSTR(A,0,1)='P' THEN BATCH PART_SAVE
    !IF $SUBSTR(A,0,1)='N' THEN GOTO NEXT
        .
        .
        .
    !NEXT: "Continue processing"
        .
        .
        .
```

The partial file above shows a fairly complex command file. This could be XEQed by the operator to prompt for all the things to be done at the end of the day. Note that an END-OF-FILE on the INPUT would cause $EOF to return a 1 or TRUE value. This would allow aborting of the rest of the job. A CARRIAGE RETURN or null input could be used to choose a default. And by using $SUBSTR(A,0,1) only the first character is checked so that F or FU or FULL could be entered.

## Expression Component - Precedence of IBEX Operators

Expressions in IBEX are evaluated from left to right with operators of the higher precedence being performed first. The precedence may be modified by the use of enclosing parenthesis.

| OPERATION | PRECEDENCE | OPERATOR |
|---|---|---|
| $functions | 9 | |
| UNARY PLUS | 8 | + |
| UNARY MINUS | 8 | - |
| MULTIPLY | 7 | * |
| DIVIDE | 7 | / |
| PLUS | 6 | + |
| MINUS | 6 | - |
| CONCATENATION | 5 | \|\| or !! |
| EQUAL | 4 | = or .EQ. |
| NOT EQUAL | 4 | ~= or .NE. |
| LESS THAN | 4 | < or .LT. |
| LESS THAN OR EQUAL | 4 | <= or .LE. |
| GREATER THAN | 4 | > or .GT. |
| GREATER THAN OR EQUAL | 4 | >= or .GE. |
| UNARY NOT | 3 | ~ or .NOT. |
| LOGICAL AND | 2 | & or .AND. |
| LOGICAL OR | 1 | \| or .OR. |

## Preprocessing of Commands

The IBEX PREPROCESSOR enables you to define new commands by substitution into normal commands. This is done by means of a preprocessor expression. A preprocessor expression consists of a command variable which may be modified by a unary modifier, and which may also be combined with an operator expression to perform a variety of functions. The preprocessor expression is defined under EXPRESSION COMPONENT in CE40, Programmer Reference Manual.

IBEX preprocessor expressions can be used with any IBEX command. Immediately after the command is read, IBEX looks for a single % sign. This character signals the start of a preprocessor expression. IBEX then evaluates the expression and substitutes the result into the command variable

Please note that there are two types of substitutions which occur in IBEX: 1) BATCH/XEQ substitutions performed when a file is XEQed, BATCHed, or XMITed, and 2) preprocessor substitutions performed in the online IBEX command stream.

### BATCH/XEQ Substitution

When a file is XEQed, BATCHed, or XMITed, expressions may be defined by !DEFAULT, !GLOBAL, and the replacement components of the BATCH and XEQ commands. As the file is read, a routine makes substitutions, checks syntax if requested (SCAN option), and prints the file (PRINT option) if requested. The resultant file is passed to IBEX for interpretation. The !DEFAULT and !INCLUDE directives are stripped from the file at this time. When IBEX interprets this file, it checks to see if a bang (!) is present, and alerts you if it is not. At this point, IBEX looks at the file for preprocessor expressions, and the file is processed in the same manner as an online command stream described below.

### Preprocessor Substitution

For an online IBEX command stream, IBEX looks for preprocessor expressions, and, when one is found (on IBEX commands only), the routine is called which evaluates the expression and returns a resultant string. This string is then inserted into the IBEX command in place of the preprocessor expression, the resultant command is parsed by IBEX, and interpreted. Following is an example of preprocessor substitution:

```
!LET CMD='DI'
!%CMD
```

In this example, the command variable CMD is given a value of 'DI'. Then the substituted string ('DI') is processed as an IBEX command. The result is that a DISPLAY command is executed. Every subsequent occurrence of !%CMD in the command stream will be processed as a DISPLAY command.

Another example:

```
!LET FID='XYZ'
!FORTRAN %FID OVER %(FID||':OU')
```

The result is that FORTRAN compiles XYZ and creates an object unit in XYZ:OU. Notice that in this example the expression is enclosed in parentheses. This is necessary for expressions which have multiple elements.

```
!FORTRAN %$INPUT('What fid?') OVER OU (NLS)
```

In the above example, you are prompted with 'What fid?'. IBEX then waits for input and substitutes this input back into the command. If your input is 'KUHENBEAKER' then the command as executed by IBEX will be:

```
!FORTRAN KUHENBEAKER OVER OU (NLS)
```

The next example shows how to use the preprocessor to abbreviate frequently used long commands.

```
!LET LINK='LINK A,B,C,D OVER FRED'
!%LINK
```

You need only type %LINK and the contents of the variable LINK will be inserted into the command record and then executed.

The final example shows where parentheses prevent ambiguities.

```
!LET FID='TEST'
!PL6 %FID OVER %(FID||':OU') (SR(.:LIB_B01),NLS)
```

This results in PL6 compiling 'TEST' and creating a file 'TEST:OU'.

Your program may access preprocessor command variables via the M$CMDVAR monitor service, and affect the execution of JCL. See CE33, Monitor Services Manual.

Since the percent sign (%) has been reserved as a special character for IBEX, if you actually need a percent character, you must double the percent sign (%%) for each required character.

## Examples of IBEX Command Files

Following are some samples of command files.

### Command File to Read Tape

SETUP: In the following command stream, you want to read some data from a tape; if the data comes in without errors, you want to process it; otherwise, you want the whole process to be terminated.

Processing the data involves compiling a program without errors, linking the object unit into a run unit (also without errors), and then using the run unit to process the data. You want to use command file logic to control processing so that if either the compile or link portion have errors, the data will be processed by the previous copy of the run unit, which is assumed to be good.

```
!JOB
!RESOURCE LT01(1600BPI)=1
```

Initiates job processing, and allocates a tape device.

```
!PCL
C LT01#DATA/DATA on *DATA
END
```

Invokes PCL, reads the data, then terminates PCL. Note that while the IBEX prompt (!) must appear in the command file, the PCL prompt does not. It is provided by the PCL processor when the file is executed.

```
!IF ERROR THEN QUIT
```

```
                    If PCL was exited with a M$ERR because of a bad read,
                    terminate processing.

!COBOL MUNGE_SI OVER MUNGE_OU

                    Invoke the COBOL compiler.

!IF ERROR THEN GOTO MUNGE

                    If COBOL was exited with M$ERR because of a compilation
                    error, branch to the command labeled MUNGE; otherwise,
                    continue.

!PCL
COPY MUNGE ON *MUNGE
END

                    Invoke PCL and save current program; terminate PCL.

!LINK MUNGE_OU OVER MUNGE
!IF OK THEN GOTO MUNGE

                    Create a new MUNGE program.  If LINK exited with a M$EXIT
                    indicating no error, branch to the command MUNGE;
                    otherwise, continue.

!COPY *MUNGE OVER MUNGE

                    Restore saved version.

!MUNGE: MUNGE *DATA

                    This is the command transferred to, which processes data
                    using MUNGE.
```

## Command File That Interrogates User

SETUP: You would like a program to ask you some questions, then modify its output depending upon your answer:

```
!BEGIN_ASK_NAME: LET FNAME=$INPUT('Enter your first name>')
!LET LNAME=$INPUT('Enter your last name>')
!OUTPUT 'Then your name must be %FNAME %LNAME,'
!LET QSTRING=$INPUT('True?>')
!LET QSTRING=$SUBSTR(QSTRING,0,1)
!IF QSTRING~='Y'&QSTRING~='y' THEN GOTO BEGIN_ASK_NAME
!LET INITIALS=$SUBSTR(FNAME,0,1)||'. '||$SUBSTR(LNAME,0,1)||'.'
!LET NAME = '%FNAME %LNAME'
!OUTPUT 'Do you mind if I call you %INITIALS?'
!LET QSTRING=$INPUT('O.K.?>')
!LET QSTRING=$SUBSTR(QSTRING,0,1)
!IF QSTRING='Y'|QSTRING='y' THEN LET NAME='%INITIALS'
!OUTPUT 'O.K. SO LONG, %NAME'
```

SETUP: You want to modify a setup file to allow you to choose from several terminal profiles:

```
!BEGIN_ASK: OUTPUT 'Terminal 1, 2, or 3?'
!LET ASKER=$INPUT('Which>')
!IF ASKER=1 THEN GOTO VIP_PLACE
!IF ASKER=2 THEN GOTO DBL_PLACE
!IF ASKER=3 THEN GOTO TTY_PLACE
!OUTPUT 'Terminal 1 is a Honeywell VIP 7801.'
!OUTPUT 'Terminal 2 is a DIABLO 1620.'
!OUTPUT 'Terminal 3 is a Teletype 33.'
!GOTO BEGIN_ASK
!VIP_PLACE: PROFILE VIP7801
!TABS 20,40,60
!END
!DBL_PLACE: PROFILE DBL1620
!TABS 10,20,30,40,50
!END
!TTY_PLACE: PROFILE TTY33
!TABS 5,10,20,30,40,50,60
!END
```

# MODULE 4-2

## Sample Session

This module presents a terminal session performed on the CP-6 system at LADC. The purpose of this module is to present examples of the following:

- FORTRAN - compiling, linking, debugging, and running of a FORTRAN program.
- COBOL - compiling, linking, and running of a COBOL program.
- PL6 - compiling, linking, and batching of a PL-6 program.
- IDP - execution of an IDP source program and creation of a dictionary.
- BASIC - creation of a BASIC program, which is then compiled and executed.

In the course of this session, use of DELTA, the LEMUR processor, the SORT processor, and TEXT are also briefly shown. All explanatory comments are indented.

```
!DONT ACCEPT(ALL)
```

The DONT ACCEPT command disables the printing of messages from the operator.

```
!DISPLAY
   USERS = 92
   ETMF =  2
   90% RESPONSE < 50 MSECS
   OCT 17 '83  15:08
```

The DISPLAY command prints the number of users, the current CPU throughput average (ETMF), the response time, and the date/time.

```
!DISPLAY USER
   MYACCT,   433USERNAME  SYSID = 48586  USER NUMBER = 62
   DIRECTORY = MYACCT   DISK SPACE REMAINING = 364
   SETUP: !XEQ SETUPS
```

The DISPLAY USER command prints information about the current user.

```
!EDIT
EDIT B03 HERE
*B FORT-TEST
    1.000        DOUBLE PRECISION A
    2.000        READ(10,10)I,J,A
    3.000 10     FORMAT(I3,1X,I3,/A7)
    4.000        OUTPUT,I,J
    5.000        WRITE(108,12)A
    6.000 12     FORMAT(' A= ',A7)
    7.000        END
    8.000
*END
```

The EDIT processor is entered and a FORTRAN source program named FORT-TEST is constructed.

```
!FORTRAN FORT-TEST OVER FORT_OU (OU)
FORTRAN 77 VERSION C00  OCT 17 '83
*     1.000>     1:        DOUBLE PRECISION A
      7.000>     7:        END
 ERRORS FOUND       : 0        TOTAL ERRORS FOUND: 0
```

The FORTRAN program is compiled, with the object unit stored as file FORT_OU. No errors are discovered by the compiler.

```
!HELP (LINK) LINK
Syntax:
{LINK|LOAD|LYNX} fid[,fid]...[tree][{ON|OVER} rununit[,listfid][(optionlist)]
```

The HELP command prints the syntax of the LINK command.

```
!LINK FORT_OU ON FORT_RU
*   :SHARED_COMMON.:SYS (Shared Library) associated.
*   No linking errors.
*   Total program size = 3K.
```

FORT_OU is LINKed and loaded; a run unit named FORT_RU is created. The LINK processor prints an allocation summary.

```
!BUILD TESTFILE
EDIT B03 HERE
    1.000 999,777
    2.000 ELEAZER
    3.000
```

Data file TESTFILE is built.

```
!SET F$10 TESTFILE
!FORT_RU.
I = 999
J = 777
A= ELEAZER
 *STOP*
```

The SET command associates the data control block F$10 (defined in line 2 of FORT-TEST) with TESTFILE. The run unit FORT_RU is executed. The program reads and prints the data in TESTFILE.

```
!UNDER DELTA
!FORT_RU.
DELTA B03 here IC = 4MAIN :1 [PROC]
>AT 4
>GO
```

The UNDER DELTA command associates the DELTA processor with the terminal session. FORT_RU is executed again; DELTA takes control. The AT 4 directive sets a breakpoint at line 4 of the FORTRAN code. The GO directive resumes execution of FORT_RU.

```
01 A_BRK@ 4MAIN :4 [I/O]
>DISPLAY I
I = 999
>DISPLAY J
J = 777
>LET J 555
>GO
```

Execution halts at line 4. The variables I and J are displayed, and variable J is altered.

```
I = 999
J = 555
A= ELEAZER
 *STOP*
M$EXIT @ XPM_1EXIT_+.17
>END
```

GO resumes execution of FORT_RU. The altered data is printed. DELTA takes control at the end of execution and the END directive exits the DELTA processor.

```
!LEMUR
LEMUR B03 here
*COPY FORT_OU TO MY_LIBRARY
  ... COPYing
*LIST MY_LIBRARY
DBG SEV COMPILER       LEMUR TIME       COMPILE TIME    OBJECT UNIT NAME
 Y  0   FORT  C00   15:10 OCT 17 '83  15:06 OCT 17 '83  4MAIN
*END
```

The LEMUR processor is entered. The COPY command creates a library file MY_LIBRARY. The LIST command prints information about MY_LIBRARY.

```
!EDIT
EDIT B03 HERE
*B COBOL-TEST
    1.000        IDENTIFICATION DIVISION.
    2.000        PROGRAM-ID.
    3.000            *COBOLTEST.
    4.000        ENVIRONMENT DIVISION.
    5.000        CONFIGURATION SECTION.
    6.000        SOURCE-COMPUTER.  LEVEL-66-ASCII.
    7.000        OBJECT-COMPUTER.  LEVEL-66-ASCII.
    8.000        INPUT-OUTPUT SECTION.
    9.000        FILE-CONTROL.
   10.000        SELECT PRINT-FILE
   11.000        ASSIGN TO PRINTER.
   12.000        SELECT INPUT-FILE
   13.000        ASSIGN TO DISK.
   14.000        DATA DIVISION.
   15.000        FILE SECTION.
   16.000        FD  PRINT-FILE
   17.000            LABEL RECORDS ARE OMITTED.
   18.000        01  PRINT-RECORD PIC X(132).
   19.000        FD  INPUT-FILE
   20.000            LABEL RECORDS ARE OMITTED.
   21.000        01  INPUT-RECORD PIC X(132).
   22.000        WORKING-STORAGE SECTION.
   23.000        01  MORE-INPUT    PIC 9  VALUE 1
   24.000        01  TRUE          PIC 9  VALUE 1.
   25.000        01  FALSE         PIC 9  VALUE 0.
   26.000        PROCEDURE DIVISION.
   27.000            PERFORM OPEN-FILES.
   28.000            PERFORM READ-WRITE-FILE UNTIL MORE-INPUT = FALSE.
   29.000            PERFORM CLOSE-FILES.
   30.000            STOP RUN.
   31.000        OPEN-FILES.
   32.000            OPEN INPUT INPUT-FILE.
   33.000            OPEN OUTPUT PRINT-FILE.
   34.000        READ-WRITE-FILE.
```

```
   35.000             MOVE SPACES TO INPUT-RECORD.
   36.000             READ INPUT-FILE
   37.000                  AT END MOVE FALSE TO MORE-INPUT.
   38.000             IF MORE-INPUT = TRUE
   39.000                  MOVE INPUT-RECORD TO PRINT-RECORD
   40.000                  WRITE PRINT-RECORD.
   41.000        CLOSE-FILES.
   42.000             CLOSE INPUT-FILE, PRINT-FILE.
   43.000
*END
```

A COBOL source program named COBOL-TEST is built.

```
!COBOL COBOL-TEST ON COBOL_OU(NLS,NLO)

00082985 06/17/83 09:13 HISI SERIES 60 LEVEL 66 COBOL CB4.2U2 B05 2U2 B05

    ESN         ISN        C    TEXT

   24.000        24         01  TRUE           PIC 9  VALUE 1.
                            1
 *** 1 3-35  AN UNRECOGNIZABLE DATA ATTRIBUTE IS ENCOUNTERED, OR
             PERIOD IS MISSING.

 THERE WERE 42 SOURCE INPUT LINES.
 THERE WERE 1 FATAL MESSAGES.
```

COBOL-TEST is compiled, but a diagnostic message indicates the presence of an error.

```
!EDIT
EDIT B03 HERE
*E COBOL-TEST
*TY 23-25
   23.000         01  MORE-INPUT     PIC 9  VALUE 1
   24.000         01  TRUE           PIC 9  VALUE 1.
   25.000         01  FALSE          PIC 9  VALUE 0.
*RR 23
   23.000         01  MORE-INPUT     PIC 9  VALUE 1.
*END
```

A period is missing in line 23. The RR command is used to correct the error.

```
!COBOL COBOL-TEST OVER COBOL_OU(NLS,NLO)

 THERE WERE 42 SOURCE INPUT LINES.
 THERE WERE NO DIAGNOSTICS.
```

COBOL-TEST is compiled successfully; the object unit is named COBOL_OU.

```
!SET INPUT-FILE TESTFILE
!SET PRINT-FILE ME
```

The data control block INPUT-FILE is set to TESTFILE; PRINT-FILE is set to the terminal (ME).

```
!RUN COBOL_OU OVER COBOL_RU
*  :SHARED_COBOL.:SYS (Shared Library) associated.
*  No linking errors.
*  Total program size = 8K.

999,777
ELEAZER
```

The RUN command causes two events: COBOL_OU is linked into run unit COBOL_RU, and COBOL_RU is executed. The data from TESTFILE is read and printed.

```
!EDIT
EDIT B03 HERE
*B PL6-TEST
    1.000 RW: PROC MAIN;
    2.000 %INCLUDE CP_6;
    3.000 %F$DCB;
    4.000 DCL M$SI DCB;
    5.000 DCL M$LO DCB;
    6.000 DCL BUFFER CHAR(80) STATIC;
    7.000 %FPT_READ (FPTN=READBUFFER,BUF=BUFFER,DCB=M$SI);
    8.000 %FPT_WRITE (FPTN=WRITEBUFFER,BUF=BUFFER,DCB=M$LO);
    9.000 DO WHILE('1'B);
   10.000       CALL M$READ(READBUFFER)ALTRET(BYEBYE);
   11.000       WRITEBUFFER.BUF_.BOUND=DCBADDR(DCBNUM(M$SI))->F$DCB.ARS#-1;
   12.000       CALL M$WRITE(WRITEBUFFER);
   13.000       END;
   14.000 BYEBYE:;
   15.000       CALL M$EXIT;
   16.000 END RW;
   17.000
```

A PL6 program named PL6-TEST is built.

```
*B PL6-JOB
    1.000 !JOB NAME=PL6JOB
    2.000 !RESOURCE TIME=2,MEM=80
    3.000 !PL6 PL6-TEST OVER PL6_OU(NLS,NLO)
    4.000 !LINK PL6_OU OVER PL6_RU
    5.000 SET M$SI TESTFILE
    6.000 !PL6_RU.
    7.000
*SE5;1P/!/;TX
    5.000 !SET M$SI TESTFILE
*   1 strings changed
*END
```

A file consisting of IBEX commands and named PL6-JOB is built and edited.

```
!BATCH PL6-JOB
Job 48671  submitted.
!CHECK
48671 PL6JOB.MYACCT running 0:06/1:56
!CHECK 48671
48671-1 PL6JOB.MYACCT completed successfully at 15:21 06/16/83
!TIME
   OCT 17  '83   15:32
```

The BATCH command submits PL6-JOB to the batch queue. The batch job is named PL6JOB and has the system identification 48671. The first CHECK indicates that the job is running. The second CHECK (using for variation the id 48671) indicates that the job is now printing. TIME displays the current date/time.

```
!EDIT
EDIT B03 HERE
*B IDP-TEST
    1.000 DICT.
    2.000 CREATE ENTRY FISH.
    3.000 CREATE FIELD ONLYFIELD
    4.000 FIRST BYTE = 1
    5.000 LENGTH = 7
    6.000 TYPE = CHARACTER.
    7.000 END DICT.
    8.000 QUERY FILE TESTFILE USE FISH.
    9.000 DISPLAY ONLYFIELD.
   10.000 RETRIEVE.
   11.000 END.
   12.000
*END
```

An IDP source program named IDP-TEST is built.

```
!IDP IDP-TEST
IDP B03 Here
:DICT.
Dict:CREATE ENTRY FISH.
Dict:CREATE FIELD ONLYFIELD
Create>FIRST BYTE = 1
Create>LENGTH = 7
Create>TYPE = CHARACTER.
Dict:END DICT.
:QUERY FILE TESTFILE USE FISH.
:DISPLAY ONLYFIELD.
:RETRIEVE.
ONLYFIELD

 999,777
 ELEAZER

***  Report Statistics for Display Number    1  ***

  File Name:  TESTFILE.MYACCT
     Number of Records Read                      2
     Number of Records Selected                  2
:END.
```

IDP-TEST is executed. The data from TESTFILE is read and printed. During this process, the dictionary file DICT is created.

```
!BASIC
BASIC C02 HERE

>AUTO
10 OPEN "TESTFILE" TO 1,INPUT
20 INPUT#1,X,Y,C$
30 PRINT X
40 B=7
50 FOR A = 1 TO 7
60 PRINT TAB(A*2);SST$(C$,1,B)
70 B = B -1
80 NEXT A
90 FOR A = 8 TO 1 STEP -1
100 PRINT TAB(A*2);SST$(C$,1,B)
110 B = B + 1
120 NEXT A
130 PRINT Y
140 CLOSE 1
150 END
160
```

The BASIC processor is entered. AUTO causes prompting of line numbers. A BASIC program is built.

```
>RUN
 999
 ELEAZER
   ELEAZE
     ELEAZ
       ELEA
         ELE
           EL
             E

             E
           EL
         ELE
       ELEA
     ELEAZ
   ELEAZE
 ELEAZER
 777
HALT AT LINE 150
```

The RUN command compiles and executes the program.

```
>SAVE BASIC-TEST
BASIC-TEST SAVED
>SYS
```

The program is stored under the name BASIC-TEST.

```
!BUILD TESTSORT1
EDIT B03 HERE
    1.000 ELEAZER
    2.000 LEAZERE
    3.000 EAZEREL
    4.000 AZERELE
    5.000 ZERELEA
    6.000 ERELEAZ
    7.000 RELEAZE
    8.000 ELEAZER
    9.000
```

A data file called TESTSORT1 is constructed.

```
!SET F$SORTIN TESTSORT1
!SET F$SCRF1 TEMP,ORG=RANDOM,IXTNSIZ=30
!SET F$SORTOUT ME
```

The data control blocks for the SORT processor are set.

```
!SORT
* SORT STARTS B06    OCT 17 '83  15:43:29.12
*REC INLEN=7
*KEY START=1,LEN=3
*END


AZERELE
EAZEREL
ELEAZER
ELEAZER
ERELEAZ
LEAZERE
RELEAZE
ZERELEA
```

```
* SORT STOPS B06    OCT 17  '83  15:44:02.37
```

The SORT processor is entered. The length of the input records, the start and length of the SORT keys are specified. The file TESTSORT1 is read, and its contents sorted and printed alphabetically.

```
!BUILD TEXT-TEST
EDIT B03 HERE
    1.000 .PDW 25
    2.000 .PDL 14
    3.000 .FBB
    4.000 ||Page=%PageNo%||
    5.000 .FBE
    6.000 .VM 0,0,0,2
    7.000 ECHO prints IBEX commands contained in a command file as
    8.000 they are read from the command
    9.000 stream.  If the
   10.000 ECHO
   11.000 command
   12.000 is
   13.000 not
   14.000 issued,
   15.000 this
   16.000 echoing of commands
   17.000 read from a
   18.000 command file
   19.000 does
   20.000 not
   21.000 occur.
   22.000
```

A file named TEXT-TEST is built. It contains TEXT control words and textual data.

```
!TEXT TEXT-TEST
TEXT C01

ECHO prints IBEX commands
contained in a command
file as they are read
from the command stream.
If the ECHO command is
not issued, this echoing
of commands read from a
command file does not
occur.

         Page=1
```

The TEXT processor is invoked to format and print the file TEXT-TEST.

```
!PCL
PCL B03 here
<HELP LIST
Syntax:
L[IST][(listopt)] [sourcelist [{TO|ON|OVER|INTO} destination][ FROM fid]
```

The PCL processor is entered. The HELP command requests information about the LIST command.

```
<LIST (A) ?TEST?,?_?
ORG TY  GRAN  NGAV     REC   LAST MODIFIED   NAME
KEY SB     1     0      16 15:37 OCT 17 '83  BASIC-TEST
KEY SE     1     0      42 14:30 OCT 17 '83  COBOL-TEST
KEY SE     1     0       7 15:08 OCT 17 '83  FORT-TEST
KEY SE     1     0      12 15:33 OCT 17 '83  IDP-TEST
KEY SE     1     0      17 15:13 OCT 17 '83  PL6-TEST
KEY SE     1     0       2 15:35 OCT 17 '83  TESTFILE
KEY SE     1     0       8 15:42 OCT 17 '83  TESTSORT1
KEY SE     1     0      22 11:00 OCT 14 '83  TEXT-TEST
KEY 01     2     0      31 15:11 OCT 17 '83  COBOL_OU
KEY R     12     0      19 15:12 OCT 17 '83  COBOL_RU
KEY OF     1     0      20 15:09 OCT 17 '83  FORT_OU
KEY R     10     0      13 15:09 OCT 17 '83  FORT_RU
KEY LE     1     0      21 14:29 OCT 17 '83  MY_LIBRARY
KEY 06     1     0      17 15:21 OCT 17 '83  PL6_OU
 ..  14 files,    38 granules listed
```

The files created during this session are listed with their attributes.

```
<DELETE ?TEST?,?_?
DELETE ?TEST?.MYACCT ?YES$
BASIC-TEST   COBOL-TEST   FORT-TEST    IDP-TEST     PL6-TEST     TESTFILE
TESTSORT1    TEXT-TEST
DELETE ?_?.MYACCT ?YES$
COBOL_OU     COBOL_RU     FORT_OU      FORT_RU      MY_LIBRARY   PL6_OU
     14 files,    38 granules deleted
<END
```

The DELETE command erases the files created during this session.

```
!OFF FULL

15:46:38 OCT 17 '83 MYACCT,433USERNAME    @UPSTAIRS             ID= 81426
SUBMITTED      15:05:56 OCT 17 '83
STARTED        15:05:56 OCT 17 '83       STEP                        38
ELAPSED                  40:41.93
PERMANENT DISK USED            -3       STEP CONDITION CODE          0
PEAK TEMPORARY DISK           111       END STATUS: NORMAL
PEAK MEMORY                   126
RESOURCE MEMORY REQUIRED      126
CPU TIME USED              0:36.00
  PROC EXEC TIME           0:05.65 @   .8250/min                   .08
  USER EXEC TIME           0:00.03 @   .8250/min
  PROC SERVICE TIME        0:29.74 @   .8250/min                   .41
  USER SERVICE TIME        0:00.58 @   .8250/min                   .01
CONNECT TIME              40:41.93 @   .0200/min                   .81
TERMINAL INTERACTIONS          121 @   .0100/tinter                1.21
JOB STEPS                       38 @   .0000/step
PMMES ISSUED                  4217 @   .0000/pmme
PROC MEMORY USED              6.92 @   .0045/pg-min                 .03
USER MEMORY USED               .10 @   .0045/pg-min
TEMP DISK USED              529.01 @   .0000/pg-min
DISK ACCESSES                 3251 @   .0004/access                1.46
-------------------------------------------------------------------------
TOTAL CHARGE =      4.01                                           4.01
```

The session ends; a complete accounting of the session is printed. The RESOURCE MEMORY REQUIRED field is an indication of how much memory to appropriate when the job is run in the future.

# MODULE 4-3

# FPL: Compiling, Linking, Debugging

SETUP: A programmer has created a Forms Program using the Forms Programming Language (FPL). The source program is a file built via the EDIT processor. The Forms Program must be compiled and linked on the CP-6 host system; it can then be loaded into a front-end processor and tested in coordination with a Transaction Processing (TP) application program operating in the CP-6 host.

First, invoke the FPL compiler for the source program FORM:S to produce an object unit as a temporary file, *OU. The source program includes a COPY statement that names a file PO-STATE-CODES already existing in the same account from which the compilation is requested.

```
!FPL FORM:S OVER *OU
  1          1.000      IDENTIFICATION DIVISION.
  2          2.000     *---------------------
  3          3.000      PROGRAM-ID.  FORM:CDA.
  4          4.000
  5          5.000      ENVIRONMENT DIVISION.
  6          6.000     *---------------------
  7          7.000      CONFIGURATION SECTION.
  8          8.000     *----------------------
  9          9.000      SPECIAL-NAMES

 *Error*     9.000 Required period is missing (in the vicinity of
                   'SPECIAL-NAMES').

  10        10.000     *--------------
  11        11.000
  12        12.000          ALPHABET ALPHA-UPPER-LOWER IS
  13        13.000                   "A" THRU "Z", "a" THRU "z", " ".
  14        14.000
  15        15.000      INPUT-OUTPUT SECTION.
  16        16.000     *---------------------
  17        17.000      FILE-CONTROL.
  18        18.000     *-------------
  19        19.000
  20        20.000          SELECT XACTION          ASSIGN TO COMMUNICATIONS
  21        21.000                                  TYPE IS XACTION-TRANTYPE.
  22        22.000          SELECT ENTRY-SCREEN     ASSIGN TO TERMINAL-IO.
  23        23.000
  24        24.000      DATA DIVISION.
  25        25.000     *--------------
  26        26.000      FILE SECTION.
  27        27.000     *-------------
  28        28.000      FD  XACTION.
  29        29.000     *------------
  30        30.000      01  TRANSACTION-RECORD.
  31        31.000          02  FUNCTION            PIC XXX     FIELD F-FUNC
  32        32.000              LEGAL ARE "ADD", "DIS", "END"
  33        33.000              ERROR-MESSAGE "*Invalid Function".
  34        34.000          02  ERRCODE             PIC 9(5).
  35        35.000              88  ACCOUNT-EXISTS              VALUE 0.
  36        36.000              88  ADD-WAS-SUCCESSFUL          VALUE 0.
  37        37.000          02  ACCOUNT-NUMBER      PIC 9(9)    FIELD F-ACCT.
  38        38.000          02  CUSTOMER-NAME       PIC X(30)   FIELD F-NAME.
  39        39.000          02  STREET-ADDRESS      PIC X(30)   FIELD F-STRT.
```

```
40       40.000          02  CITY                PIC X(30)  FIELD F-CITY.
41       41.000          02  STATE               PIC XX     FIELD F-STATE
42       42.000              LEGAL ARE POST-OFFICE-STATE-CODES
43       43.000              ERROR-MESSAGE IS "*Invalid U.S. state code".
44       44.000          02  ZIPCODE             PIC 9(5)   FIELD F-ZIP.
45       45.000
46       46.000       FD  ENTRY-SCREEN
47       47.000      *---------------
48       48.000          FRAMES ARE TITLES, DATA-ENTRIES.
49       49.000
50       50.000       WORKING-STORAGE SECTION.
51       51.000      *-----------------------
52       52.000       01  XACTION-TRANTYPE        PIC X(8)   VALUE "EX1APPL".
53       53.000
54       54.000       COPY PO-STATE-CODES.
55        1.000+      01  OFFICIAL-POST-OFFICE-CODES.
56        2.000+          02  P-O-CODES.
57        3.000+              03  ALABAMA              PIC XX VALUE "AL".
58        4.000+              03  ALASKA               PIC XX VALUE "AK".
59        5.000+              03  ARIZONA              PIC XX VALUE "AZ".
60        6.000+              03  ARKANSAS             PIC XX VALUE "AR".
61        7.000+              03  CALIFORNIA           PIC XX VALUE "CA".
62        8.000+              03  COLORADO             PIC XX VALUE "CO".
63        9.000+              03  CONNECTICUT          PIC XX VALUE "CT".
64       10.000+              03  DELAWARE             PIC XX VALUE "DE".
65       11.000+              03  DISTRICT-OF-COLUMBIA PIC XX VALUE "DC".
66       12.000+              03  FLORIDA              PIC XX VALUE "FL".
67       13.000+              03  GEORGIA              PIC XX VALUE "GA".
68       14.000+              03  HAWAII               PIC XX VALUE "HI".
69       15.000+              03  IDAHO                PIC XX VALUE "ID".
70       16.000+              03  ILLINOIS             PIC XX VALUE "IL".
71       17.000+              03  INDIANA              PIC XX VALUE "IN".
72       18.000+              03  IOWA                 PIC XX VALUE "IA".
73       19.000+              03  KANSAS               PIC XX VALUE "KS".
74       20.000+              03  KENTUCKY             PIC XX VALUE "KY".
75       21.000+              03  LOUISIANA            PIC XX VALUE "LA".
76       22.000+              03  MAINE                PIC XX VALUE "ME".
77       23.000+              03  MARYLAND             PIC XX VALUE "MD".
78       24.000+              03  MASSACHUSETTS        PIC XX VALUE "MA".
79       25.000+              03  MICHIGAN             PIC XX VALUE "MI".
80       26.000+              03  MINNESOTA            PIC XX VALUE "MN".
81       27.000+              03  MISSISSIPPI          PIC XX VALUE "MS".
82       28.000+              03  MISSOURI             PIC XX VALUE "MO".
83       29.000+              03  MONTANA              PIC XX VALUE "MT".
84       30.000+              03  NEBRASKA             PIC XX VALUE "NE".
85       31.000+              03  NEVADA               PIC XX VALUE "NV".
86       32.000+              03  NEW-HAMPSHIRE        PIC XX VALUE "NH".
87       33.000+              03  NEW-JERSEY           PIC XX VALUE "NJ".
88       34.000+              03  NEW-MEXICO           PIC XX VALUE "NM".
89       35.000+              03  NEW-YORK             PIC XX VALUE "NY".
90       36.000+              03  NORTH-CAROLINA       PIC XX VALUE "NC".
91       37.000+              03  NORTH-DAKOTA         PIC XX VALUE "ND".
92       38.000+              03  OHIO                 PIC XX VALUE "OH".
93       39.000+              03  OKLAHOMA             PIC XX VALUE "OK".
94       40.000+              03  OREGON               PIC XX VALUE "OR".
95       41.000+              03  PENNSYLVANIA         PIC XX VALUE "PA".
96       42.000+              03  RHODE-ISLAND         PIC XX VALUE "RI".
97       43.000+              03  SOUTH-CAROLINA       PIC XX VALUE "SC".
98       44.000+              03  SOUTH-DAKOTA         PIC XX VALUE "SD".
99       45.000+              03  TENNESSEE            PIC XX VALUE "TN".
100      46.000+              03  TEXAS                PIC XX VALUE "TX".
101      47.000+              03  UTAH                 PIC XX VALUE "UT".
102      48.000+              03  VERMONT              PIC XX VALUE "VT".
103      49.000+              03  VIRGINIA             PIC XX VALUE "VA".
104      50.000+              03  WASHINGTON           PIC XX VALUE "WA".
105      51.000+              03  WEST-VIRGINIA        PIC XX VALUE "WV".
106      52.000+              03  WISCONSIN            PIC XX VALUE "WI".
```

```
107      53.000+             03  WYOMING                 PIC XX VALUE "WY".
108      54.000+             03  BLANKS                  PIC XX VALUE SPACES.
109      55.000+         02  POST-OFFICE-STATE-CODES     REDEFINES P-O-CODES
110      56.000+                                         OCCURS 52 TIMES
111      57.000+                                         PIC XX.
112      55.000
113      56.000       FRAME SECTION.
114      57.000      *--------------
115      58.000       FR  TITLES.
116      59.000      *-----------
117      60.000       01  TITLES-RECORD              OUTPUT.
118      61.000           02  F-T1                  PIC X(16)  POSITION 8, 32
119      62.000               CONSTANT VALUE "Example Number 1".
120      63.000           02  F-T2                  PIC X(9)   POSITION 10, 23
121      64.000               CONSTANT VALUE "Function:".
122      65.000           02  F-T3                  PIC X(15)  POSITION 10, 38
123      66.000               CONSTANT VALUE "Account Number:".
124      67.000           02  F-T4                  PIC X(14)  POSITION 12, 18
125      68.000               CONSTANT VALUE "Customer Name:".
126      69.000           02  F-T5                  PIC X(15)  POSITION 13, 17
127      70.000               CONSTANT VALUE "Street Address:".
128      71.000           02  F-T6                  PIC X(5)   POSITION 14, 27
129      72.000               CONSTANT VALUE "City:".
130      73.000           02  F-T7                  PIC X(11)  POSITION 15, 21
131      74.000               CONSTANT VALUE "State Code:".
132      75.000           02  F-T8                  PIC X(9)   POSITION 16, 23
133      76.000               CONSTANT VALUE "Zip Code:".
134      77.000
135      78.000       FR  DATA-ENTRIES.
136      79.000      *-----------------
137      80.000       01  F-FUNC-RECORD              I-O.
138      81.000           02  F-FUNC                 PIC XXX    POSITION 10, 33
139      82.000                                       ENTRY-REQUIRED.
140      83.000           02  F-ACCT                 PIC 9(9)   POSITION 10, 54
141      84.000               PROMPT IS "Please enter an account number".
142      85.000
143      86.000       01  F-CUSTOMER-RECORD          I-O.
144      87.000           02  F-NAME                 PIC X(30)  POSITION 12, 33
145      88.000                                       ENTRY-REQUIRED
146      89.000                                       CLASS ALPHA-UPPER-LOWER.
147      90.000           02  F-STRT                 PIC X(30)  POSITION 13, 33
148      91.000                                       DEFAULT VALUE SPACES.
149      92.000           02  F-CITY                 PIC X(30)  POSITION 14, 33
150      93.000                                       DEFAULT VALUE SPACES.
151      94.000           02  F-STATE                PIC XX     POSITION 15, 33
152      95.000                                       DEFAULT VALUE SPACES.
153      96.000           02  F-ZIP                  PIC 9(5)   POSITION 16, 33
154      97.000                                       MINIMUM SIZE IS 5
155      98.000                                       DEFAULT VALUE ZEROES.
156      99.000      /
157     100.000       PROCEDURE DIVISION.
158     101.000      *------------------
159     102.000       MAINLINE.
160     103.000      *---------
161     104.000           OPEN I-O XACTION, ENTRY-SCREEN.
162     105.000           ACTIVATE TITLES, DATA-ENTRIES.
163     106.000           DISABLE INPUT FOR ENTRY-SCREEN.
164     107.000           WRITE TITLES-RECORD.
165     108.000           PERFORM 100-CONTROL THRU 199-END-CONTROL
166     109.000               UNTIL FUNCTION = "END".
167     110.000           CLOSE XACTION, ENTRY-SCREEN.
168     111.000           STOP RUN.
169     112.000      /
170     113.000       100-CONTROL.
171     114.000      *------------
172     115.000           CLEAR I-O FOR ENTRY-SCREEN.
173     116.000           PERFORM 200-GET-FUNCTION-AND-ACCOUNT THRU 299-EXIT.
```

```
174     117.000          IF  FUNCTION = "ADD"
175     118.000              PERFORM 300-ADD-FUNCTION THRU 399-END-ADD
176     119.000              GO TO 199-END-CONTROL.
177     120.000          IF  FUNCTION = "DIS"
178     121.000              PERFORM 400-DISPLAY-FUNCTION.
179     122.000      199-END-CONTROL.
180     123.000     *----------------
181     124.000          EXIT.
182     125.000
183     126.000      200-GET-FUNCTION-AND-ACCOUNT.
184     127.000     *----------------------------
185     128.000          ENABLE INPUT F-FUNC, F-ACCT.
186     129.000          READ ENTRY-SCREEN.
187     130.000          VERIFY FUNCTION.
188     131.000          DISABLE INPUT F-FUNC.
189     132.000          IF  FUNCTION = "END"
190     133.000              GO TO 299-EXIT.
191     134.000
192     135.000      210-VERIFY-ACCOUNT.
193     136.000     *-------------------
194     137.000          IF  F-ACCT IS AVAILABLE
195     138.000              VERIFY ACCOUNT-NUMBER
196     139.000              DISABLE INPUT F-ACCT
197     140.000          ELSE
198     141.000              ACCEPT F-ACCT
199     142.000              GO TO 210-VERIFY-ACCOUNT.
200     143.000      299-EXIT.
201     144.000     *---------
202     145.000          EXIT.
203     146.000
204     147.000      300-ADD-FUNCTION.
205     148.000     *-----------------
206     149.000          MOVE "CHK" TO FUNCTION.
207     150.000          WRITE TRANSACTION-RECORD.
208     151.000          READ XACTION.
209     152.000          IF  ACCOUNT-EXISTS
210     153.000              STOP "That account already exists!"
211     154.000              GO TO 399-END-ADD.
212     155.000          MOVE "ADD" TO FUNCTION.
213     156.000          ENABLE INPUT F-CUSTOMER-RECORD.
214     157.000          READ ENTRY-SCREEN; INVALID GO TO 399-END-ADD.
215     158.000          VERIFY TRANSACTION-RECORD.
216     159.000          DISABLE INPUT FOR F-CUSTOMER-RECORD.
217     160.000          WRITE TRANSACTION-RECORD.
218     161.000          READ XACTION.
219     162.000          IF  ADD-WAS-SUCCESSFUL
220     163.000              STOP "The account has been added"
221     164.000          ELSE
222     165.000              STOP "Sorry, that account exists!".
223     166.000      399-END-ADD.
224     167.000     *------------
225     168.000          EXIT.
226     169.000
227     170.000      400-DISPLAY-FUNCTION.
228     171.000     *---------------------
229     172.000          WRITE TRANSACTION-RECORD.
230     173.000          READ XACTION.
231     174.000          IF  ACCOUNT-EXISTS
232     175.000              PRESENT TRANSACTION-RECORD
233     176.000              WRITE F-CUSTOMER-RECORD
234     177.000              STOP "Hit TRANSMIT to continue"
235     178.000          ELSE
236     179.000              STOP "That account does not exist".

Errors:         1
Procedure size: 03B6
Data size:      019D
```

In the above compilation, a single error has been detected by the FPL compiler. To correct the error, reinvoke the EDIT processor for the source file (FORM:S). The diagnostic message from the compiler indicates a missing period, which can be added using the RR command to reread the record that contains the error.

```
!EDIT FORM:S
EDIT B03 HERE
*TY9
    9.000         SPECIAL-NAMES
*RR9
    9.000         SPECIAL-NAMES.
*END
!
```

At some sites, policy may require that all compilations be performed in batch mode. To do this, invoke EDIT and BUILD a job file. This file requests the FPL compilation; it also invokes FEPLINK to process the object unit file (temporary file *OU) and to produce the executable interpretation unit called FORM:IU. The interpretation unit can be loaded into a front-end processor and executed under control of the Forms Interpreter.

```
*E
!EDIT
EDIT B03 HERE
*BUILD FPL_JOB
    1.000 !JOB
    2.000 !FPL FORM:S OVER *OU
    3.000 !FEPLINK *OU OVER FORM:IU
    4.000
*END
!BATCH FPL_JOB
27429 .RPACCT running 0:00/0:00
!
```

Some time later, check the status of the batch compilation for completion using the CHECK command.

```
!CHECK 27429
27429-1 .RPACCT completed successfully at 13:08 09/08/83
```

This message indicates that both the FPL compilation and FEPLINK occurred without error.

The testing of the Forms Program must be coordinated with the application program operating on the CP-6 host. The administrator responsible for operation of the Forms Program in production mode must prepare for testing (as described in the TP Administrator Guide, CE50). Assuming that these preparations are made, the Forms Program can be loaded into the front-end processor on request from an authorized user logged on to a separate terminal. Typically the programmer operates the two terminals used in debugging.

At the terminal that is to communicate with the Forms Program, log on and enter the command:

```
DEBUG CDA
```

where CDA is the type associated with the Forms Program FORM:IU. Meanwhile via the programmer's terminal, invoke the debugger for Forms Programs, FOX, on behalf of the station that called the Forms Program.

```
!FOX
FOX B02
>DEBUG RLP ON TP/RLPINST USING FORM:IU
FOX B02 HERE IC = FORM:CDA :157 [ENTRY]
```

FOX identifies itself, the Forms Program by PROGRAM-ID (FORM:CDA), and the entry point into the program according to the debug schema (at 157 which is source line number 100, the start of the PROCEDURE DIVISION). FOX then waits for entry of a directive from the programmer.

Using the AT directive, set a breakpoint at 174, the number assigned in the debug schema to the source line number 117.

```
>AT 174
```

Request program execution by entering the directive GO. (At the terminal running the Forms Program, enter the data requested. Initially, enter DIS and an account number to request display of that account information.)

```
>GO
01 BRK @ FORM:CDA :174 [IF]
```

Execution stops at the specified breakpoint, allowing the programmer to enter the DISPLAY directive to examine the contents of the item called FUNCTION.

```
>DISPLAY FUNCTION
FUNCTION = 'DIS'
```

To request execution of statements in the program, statement-by-statement, enter the symbol ].

```
>]
FORM:CDA :177 [IF]
>]
FORM:CDA :178 [PERFORM]
>]
FORM:CDA :229 [I/O]
>]
FORM:CDA :230 [I/O]
>]
FORM:CDA :231 [IF]
```

Instead of continuing step by step execution, display the item called CUSTOMER-NAME, then change the contents of that field via the LET directive.

```
>DIS CUSTOMER-NAME
CUSTOMER-NAME = 'MIKE ROESECOND                '
>LET CUSTOMER-NAME 'RICHARD PETKIEWICZ'
>DIS CUSTOMER-NAME
CUSTOMER-NAME = 'RICHARD PETKIEWICZ            '
```

Resume program execution by entering the GO directive.

```
>GO
```

To interrupt execution, press the BREAK key (or press ESC, then B). This action stops execution and identifies the point of interruption.

```
BREAK @ FORM:CDA :220 [STOP]
```

Display any currently assigned breakpoints via the SHOW directive.

```
>SHOW ATS
01 FORM:CDA :174 [IF]
```

The only breakpoint (at 174) is deleted via KILL ATS and replaced by a breakpoint at the same point that also includes an attachment (the DISPLAY directive). Then resume execution by entering G for GO. (From the terminal running the Forms Program, request the function ADD and an account number.)

```
KILL ATS
>AT 174;DIS FUNCTION
>G
01 BRK @ FORM:CDA :174 [IF]
FUNCTION = 'ADD'
```

At the breakpoint, the contents of FUNCTION is displayed as requested by the attachment. Then ] is entered to request statement by statement execution.

```
>]
FORM:CDA :175 [PERFORM]
>]
FORM:CDA :206 [ASSIGNMENT]
>]
FORM:CDA :207 [I/O]
>]
FORM:CDA :208 [I/O]
>]
FORM:CDA :209 [IF]
>]
FORM:CDA :210 [STOP]
>]
FORM:CDA :211 [GOTO]
```

Cancel via the KILL directive any breakpoints, then use the SHOW directive to verify that the KILL took effect. Proceed with execution by entering GO.

```
>KILL ATS
>SHOW
  No such breakpoint(s)
>GO
```

Press the BREAK key and display FUNCTION; then exit FOX by entering the END directive.

```
BREAK @ FORM:CDA :214 [I/O]
>DIS FUNCTION
FUNCTION = 'ADD'
>END
!
```

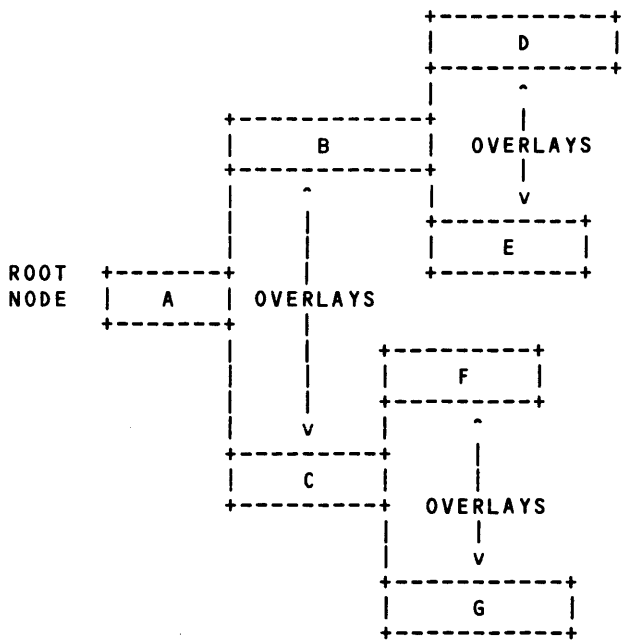
# MODULE 4-4

# Using LINK Overlays

This module discusses LINK overlay programs and provides several examples of using overlay programs in the CP-6 system.

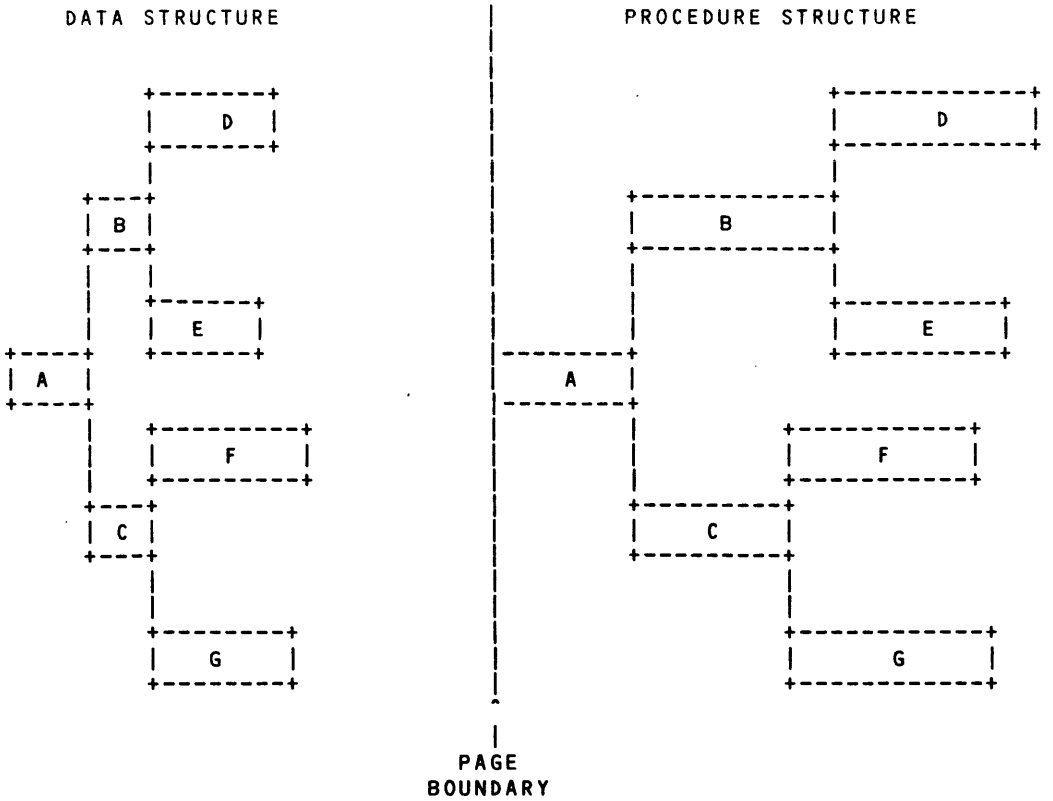
Overlay programs are used to reduce memory requirements for a program by constructing the run unit so that the program can execute with only a part of the program image in memory. Some of the nonresident portions are defined to overlay other parts of the program image.

The following figure shows the main memory layout of an overlay program:




Each box represents a portion or node of the program. Each node is stored as a separate entity within the run unit file. Memory allocation extends horizontally from left to right. In the sample overlay program, B and C overlay each other, D and E overlay each other, and F and G overlay each other.

For program overlays, the LINK processor constructs two different structures: one for data (with read/write/execute access), and one for procedure (with read/execute access) as illustrated below:




The data structure is allocated first; the procedure structure is allocated starting at the page boundary following the maximum path of the data structure, unless modified by the PBIAS LINK option.

Overlay programs are specified using the complex form of the LINK command. The specification of overlay programs is described in detail in the CP-6 Programmer Reference Manual, CE40.

## Resolving Differences and Ambiguities

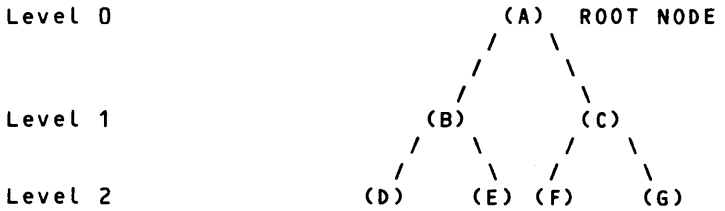
Overlay programs can be quite complex. If the programs contains multiple definitions which are 'ambiguous', the LINK processor will attempt to resolve these references in accordance with rules specified in CE40, the Programmer Reference Manual. If references cannot be resolved, they are collected by the LINK processor and listed at the end of the linking process.

For a discussion of overlay structures containing ambiguities and the rules for resolution, see the Programmer Reference Manual, CE40.

Programs with more than one level of overlay cannot be shared. See the CP-6 System Programmer Guide, CE62, for a discussion of this restriction.

## Program Trees

The following figure depicts a sample overlay program as a tree structure.




The linkage forms the paths of the tree and determines the resolution of references. The four paths in the sample program are the four connecting lines which proceed from the root node (A), to the four farthest nodes (D, E, F, and G).

Node relationships are defined in terms of levels. The root node is at level 0. Nodes connected to the root node are at level 1, and so on through the last level of the tree.

All nodes except the root node have a parent node, a node at the previous level to which the node is directly connected. The level 1 nodes are 'direct descendents' of the root node. Level 1 nodes, in turn, are the parents of the level 2 nodes connected to them.

The descendents of a node are all nodes to the nth level of the tree that descends in an connecting line from the node. (In the sample tree structure, the descendents of B are D and E, and the descendents of A are all the remaining nodes in the structure.)

The ancestors of a node are all parent nodes in a backward path to the root node. (In the sample tree structure, A and B are the ancestors of D).

## Specifying an Overlay

The complex form of the LINK, LOAD, LYNX or RUN command is used to define and describe a program overlay structure. Member nodes are defined by specifying their names, and their relationship is described by identifying their levels through balanced or nested parentheses. Consider the following tree structure:

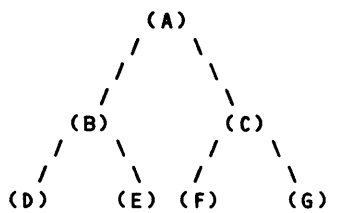

This overlay structure is defined as follows:

```
!LINK A(B)(C) ON...
```

The root node A is not enclosed in parentheses. Each of its level 1 descendents is defined as a tree, and that definition is specified by enclosing each level 1 node in a separate set of parentheses.

Both the member nodes and the tree order is established through the command. B is now defined as the leftmost path. This order is important in that it establishes the order in which the LINK processor searches descendent nodes to resolve ambiguities.

Now, consider again the tree structure:




This overlay structure is defined as follows:

```
!LINK A(B(D)(E))(C(F)(G)) ON...
```

The above example illustrates the following features of overlay specifications:

1. The root node is specified immediately following the keyword LINK and is not enclosed in parentheses as follows:

```
     Root (R)
      |
      v
!LINK A
```

2. A left parenthesis is entered to indicate that a definition of a direct descendent at level 1 follows. The file id (or ids) that make up the level 1 node are specified:

```
      R
      |
      v
!LINK A(B
       ^
       |
      1st level (1st)
```

3. Additional left hand parentheses/file ids are specified to define a complete path to the last level:

```
      R  2nd level (2nd)
      |  |
      v  v
!LINK A(B(D
       ^
       |
      1st
```

4. The end of the last level is signalled by closing the deepest nested parenthesis:

```
      R  2nd
      |  |
      v  v
!LINK A(B(D)
       ^  ^
       |  |
      1st  End of Path (EOP)
```

5. The backward path from the last level is followed looking for the first branch to open a new path:

```
      R 2nd 2nd
      |  |  |
      v  v  v
!LINK A(B(D)(E
       ^  ^
       |  |
      1st  EOP
```

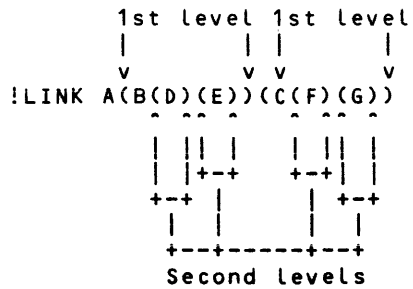
6. The new path is followed to completion. The end of the last level is signalled by closing the deepest nested parenthesis:

```
      R 2nd 2nd
      |  |  |
      v  v  v
!LINK A(B(D)(E)
       ^  ^  ^
       |  |  |
      1st EOP EOP
```

7. This backward/branch forward procedure continues until all the descendents of the direct descendent of the root node have been defined and their relationships described through nested parentheses.

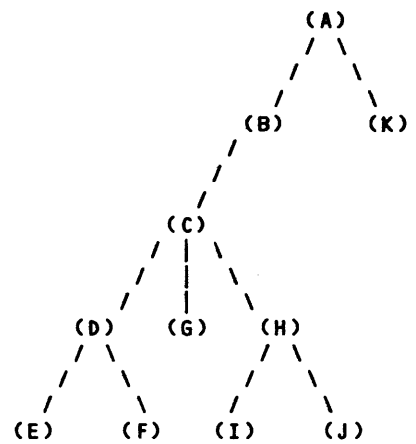
```
       1st level
       +-------+
       |       |
       |2nd 2nd|
      R| |  |  End of Descendent (EOD)
       | |  |  |
       v v  v  v
!LINK A(B(D)(E))
       ^  ^  ^
       |  |  |
      1st EOP EOP
```

8. The procedure is repeated with the next direct descendent of the root node.

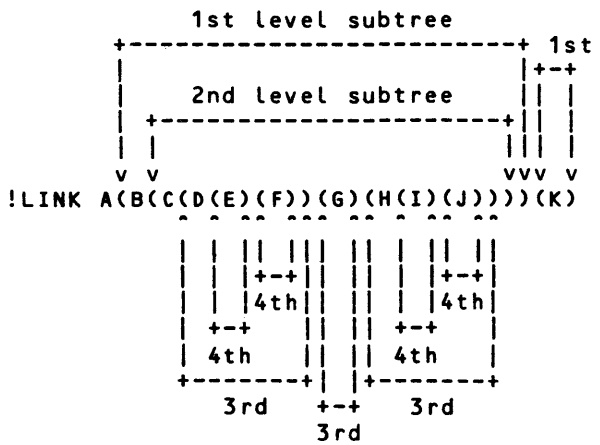



This procedure continues until every direct descendent of the root node and all its descendents are defined in this manner.

Consider the following tree structure:




This overlay structure is defined as follows:




Notes:

1. For each node at each level a new parenthesis is open. These parentheses are nested along the path from the 1st level ancestor to the farthest descendent.

2. When the farthest point on the leftmost path is reached, the first closed parenthesis is entered.

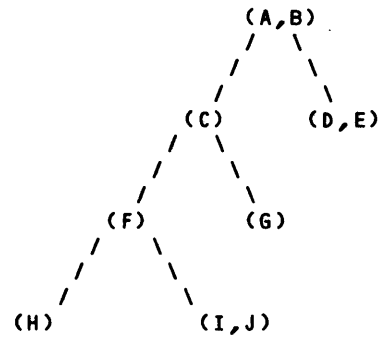
3. Definition continues by taking the backward path:

If the next (backward) node has no descendents, a closed parenthesis is entered.

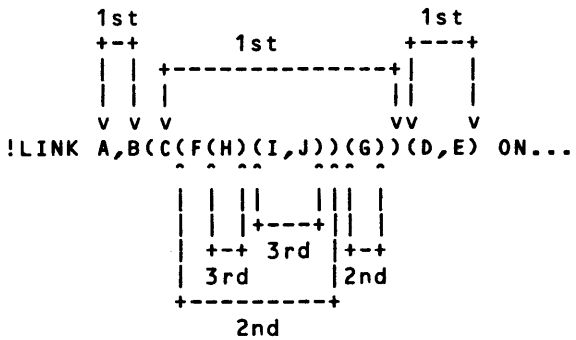
If the next (backward) node has a descendent, definition branches forward down the leftmost path of the new subtree.

4. This process repeats itself until the entire tree is defined, and all left-hand parentheses are balanced with right-hand parentheses.

Finally, consider the following example of a LINK command that defines and describes the overlay structure of a program in which some nodes contain more than a single file. Assume the following structure:




The following LINK command defines and describes the above overlay structure:



## Using LINK to Build a Run Unit with Overlays

LINK's BREF option should be invoked when you create a program with overlays. Unless the BREF option is used, you will have to load and transfer control to overlays via explicit calls to monitor services M$OLAY. (When the BREF option is used, the linker supplies all calls to M$OLAY.) The M$OLAY service calls or releases a specified overlay.

A rule-of-thumb to determine what value to use for BREF= is to use the number of nodes minus one.

Example:

Following is an example of linking an overlaid program:

```
!LINK ;
!     ROOT:OU;
!     (NODE1:OU,SUB1:OU,SUB2:OU,SUB3:OU);
!     (NODE2:OU,SUB4:OU,SUB5:OU);
! OVER  PGM:RU (BREF=2)
```

The next example illustrates an overlaid FORTRAN program:

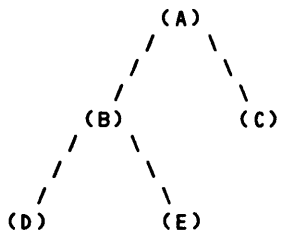
```
!FORTRAN *ROOT OVER *ROOT_OU,ME
FORTRAN 77 VERSION C00  SEP 06 '83
*    1.000>     1:        PROGRAM ROOT
     2.000>     2:        CALL NODE1
     3.000>     3:        CALL NODE2
     4.000>     4:        END
 ERRORS FOUND       : 0        TOTAL ERRORS FOUND: 0

!FORTRAN *NODE1 OVER *NODE1_OU,ME
FORTRAN 77 VERSION C00  SEP 06 '83
*    1.000>     1:        SUBROUTINE NODE1
     2.000>     2:        PRINT*,'Entering Node1'
     3.000>     3:        END
 ERRORS FOUND       : 0        TOTAL ERRORS FOUND: 0

!FORTRAN *NODE2 OVER *NODE2_OU,ME
FORTRAN 77 VERSION C00  SEP 06 '83
*    1.000>     1:        SUBROUTINE NODE2
     2.000>     2:        PRINT*,'Entering Node2'
     3.000>     3:        END
 ERRORS FOUND       : 0        TOTAL ERRORS FOUND: 0

!LINK *ROOT_OU(*NODE1_OU)(*NODE2_OU) OVER *L(BREF=2)
*  :SHARED_COMMON.:SYS (Shared Library) associated.
*  Number of branch reference instances = 2.
*  Number of unique branch reference targets = 2.
*  No linking errors.
*  Total program size = 3K.
!*L

Entering Node1
Entering Node2
 *STOP*
```

## Using LINK's PROMOTE_BLANK and PROMOTE_LABEL Option

The PROMOTE_BLANK and PROMOTE_LABEL LINK options are used to specify how blank common and labeled common storage will be used.

- PROMOTE_BLANK specifies that all instances of blank common must refer to the same storage allocation.
- PROMOTE_LABEL specifies that all instances of labeled common with the same name must refer to the same storage allocation.
- PROMOTE Specifies that both the PROMOTE_BLANK and PROMOTE_LABEL options are to be in effect. The default is not to PROMOTE.

If PROMOTE_BLANK is specified, the LINK processor may promote the allocation of blank common to accomplish this request. Assume the following structure:



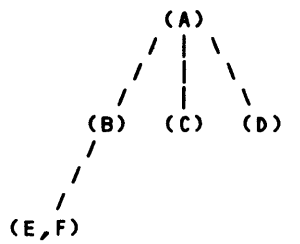

If instances of blank common are in nodes D and E, and the PROMOTE_BLANK option is not specified, allocations for blank common will be made in both nodes D and E.

However, if the PROMOTE_BLANK option is specified, the LINK processor will allocate only one area of blank common so that all instances of blank common refer to the allocated storage. This may be accomplished by promoting allocation of blank common to one of its ancestors. In the case above, blank common will be allocated in node B.

Since PROMOTE_LABEL specifies that all instances of labeled common with the same name must refer to the same storage allocation, the LINK processor may promote the allocation of each and every uniquely named label common when PROMOTE_LABEL is specified.

Example

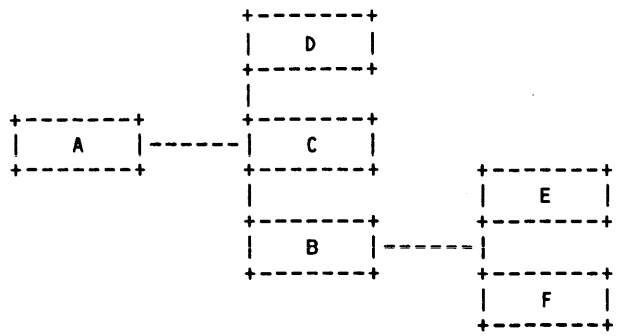
Assume that you have six object modules named A, B, C, D, E, and F which you want to link into an overlaid program. Also assume that all modules but A have references to a labeled common section named BLAZE. The overlaid program can be represented as follows:




The LINK command to be used is:

```
!LINK A(B(E,F))(C)(D) OVER *RU (BREF=3)
```

This command generates the following overlay structure:




When PROMOTE_LABEL is specified, all labeled common sections are forced to the root node A.

IF PROMOTE_LABEL is not specified, references to a labeled common section called BLAZE are unique in nodes B, C and D. In other words, nodes B, E and F share a common area of memory tagged with the label BLAZE, node C has a unique area of memory tagged with the label BLAZE, and node D also has a unique area of memory tagged with the label BLAZE.

If node A has a reference to BLAZE, PROMOTE_LABEL will have no effect. PROMOTE_BLANK works analogously to PROMOTE_LABEL. The default is to not PROMOTE_BLANK and not PROMOTE_LABEL.

If the target RU is not to be overlaid, the PROMOTE options have no effect.

## Using HELP

Enter

```
!HELP (LINK) TOPICS
```

to obtain a display of LINK topics available through the HELP facility.

# MODULE 5-0

## Section 5 - Practical Applications

This section contains examples of CP-6 system practical applications.

# MODULE 5-1

# Creating Sorted Indexes on CP-6

The following procedure for sorting document indexes will accept any EDIT-built file for input, sort it into alphabetic order (regardless of case), and place it back into your original file. A few restrictions apply:

1. Do not place any CP-6 TEXT commands in the file; they will simply be sorted at the front of the output file.

2. Maximum record length is 100 characters.

3. Every line must have an index entry to sort in it; that is, continuation lines are not allowed. If you must continue a line, repeat the index entry on the continuation line.

   Example:

   ```
   DELTA commands,3-6,4-7,4-10,4-12,
       5,19-5,19-7,20-1
   ```

   should be entered as

   ```
   DELTA commands,3-6,4-7,4-10,4-12
   DELTA commands,5,19-5,19-7,20-1
   ```

4. The index entries may begin in any column, but you must be consistent in a single file; that is, if you start entering index entries in column 1, then all succeeding entries must begin in column 1 also.

5. The separator between the index entry and references to it is immaterial; in the above example, a comma was used. Spaces may also be used. The list of references may begin in any column after the index entry.

You may call the index by any filename you desire. To sort the file into order, the command is

```
!XEQ SORTINDEX[.account] INDEX='filename'
```

The procedure will output a considerable amount of text on your terminal while it is working. Check this carefully for comments enclosed in double quotes ("); they will tell you whether the process was successful or not. When completed successfully, your original file will be sorted into alphabetic order and re-keyed for editing.

This is what SORTINDEX looks like:

```
!PCL
SEVERITY ABORT
COPY INDEX OVER INDEX(NLN,RECS=100)
!IF STEPCC>0 THEN GOTO NOFILE
!SET F$SORTIN INDEX
!SET F$SORTOUT INDEX
!SORT
REC INLEN=100,MEMORY=32
KEY START=1,LEN=100,XLATE=UC
XLATE UC,START=97,CHAR='ABCDEFGHIJKLMNOPQRSTUVWXYZ',;
 START=225,CHAR='ABCDEFGHIJKLMNOPQRSTUVWXYZ',;
```

```
 START=353,CHAR='ABCDEFGHIJKLMNOPQRSTUVWXYZ',;
 START=481,CHAR='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
END
!IF STEPCC=0 THEN GOTO GOODSORT
!"SORT step failed; index is not in order."
!GOODSORT: COPY INDEX OVER INDEX(LN,NBLANKS)
!RESET F$SORTIN
!RESET F$SORTOUT
!GOTO DONE
!NOFILE: "File name missing: say !XEQ SORTINDEX.acct INDEX=filename"
!DONE: "Processing terminated"
```

A sample run follows:

```
!EDIT
EDIT B03 HERE
*BUILD IND
    1.000 FOUR
    2.000 SCORE
    3.000 AND
    4.000 SEVEN
    5.000 YEARS
    6.000 ago
    7.000 our
    8.000 forefathers
    9.000 brought
   10.000 forth
   11.000 upon
   12.000 this
   13.000 continent
   14.000 A
   15.000 new
   16.000 nation
   17.000 conceived
   18.000 in
   19.000 liberty
   20.000 and
   21.000 DEDICATED
   22.000 TO
   23.000 the
   24.000 PROPOSITION
   25.000 that
   26.000 all
   27.000 MEN
   28.000 are
   29.000 created
   30.000 EQUAL
   31.000
*END
!XEQ SORTINDEX INDEX=IND
$PCL
PCL B03 here
<SEVERITY ABORT
<COPY IND OVER IND(NLN,RECS=100)
 ..COPYing
$IF STEPCC>0 THEN GOTO NOFILE
$SET F$SORTIN IND
$SET F$SORTOUT IND
$SORT

* SORT STARTS B06    OCT 20 '83  14:45:04.37

****  NO COLLATION FILES; WILL ATTEMP MEMORY CONTAINED SORT.
```

```
    * SORT STOPS  B06    OCT 20 '83  14:45:10.02
    $IF STEPCC=0 THEN GOTO GOODSORT
    $                 GOTO GOODSORT
    $GOODSORT: COPY IND OVER IND(LN,NBLANKS)
     ..COPYing
    $RESET F$SORTIN
    $RESET F$SORTOUT
    $GOTO DONE
    $DONE: "Processing terminated"
```

A listing of the sorted file appears below:

```
    !EDIT IND
    EDIT B03 HERE
    *TY
        1.000 A
        2.000 ago
        3.000 all
        4.000 AND
        5.000 and
        6.000 are
        7.000 brought
        8.000 conceived
        9.000 continent
       10.000 created
       11.000 DEDICATED
       12.000 EQUAL
       13.000 forefathers
       14.000 forth
       15.000 FOUR
       16.000 in
       17.000 liberty
       18.000 MEN
       19.000 nation
       20.000 new
       21.000 our
       22.000 PROPOSITION
       23.000 SCORE
       24.000 SEVEN
       25.000 that
       26.000 the
       27.000 this
       28.000 TO
       29.000 upon
       30.000 YEARS
    *   EOF fit after 30.000
    *END
```

# MODULE 5-2

## How to Perform Compilations in Batch Mode

The following method will enable you to run your compilation in batch mode, deliver the compilation diagnostics to a file in your account, and deliver a line printer listing only if the compilation is successful. Although this example uses a PL-6 compilation, a similar approach may be used with other CP-6 system compilers. Also note that LISTER.X and EDGEMARK.X may not exist at all sites.

In this example, several options may be selected by the user batching the job on the BATCH command. These are:

| | |
|---|---|
| $LO | If $LO is equal to 1, the job will make a PL-6 compilation with LO, and, if the compilation is successful, print the compilation listing (source & LO) on the line printer, edgemarked appropriately. If $LO is equal to 0 (the default), the job will perform a standard LS compilation and, if successful, print the LS listing, edgemarked. |
| $NOLS | If $NOLS is equal to 1, no listing will be printed if the compilation was successful (even the JCL and banner will be suppressed). If $NOLS is equal to 0 (the default), standard, edgemarked listings will be produced if the compilation was successful. |
| $PRERR | If $PRERR is equal to 1, the erroring compilation will print a compilation listing on the line printer. If $PRERR is equal to 0 (the default), the erroring compilation will not generate a line printer listing of the compilation. In either case, a file called D:modulename will be created in the user's account. |

As presented below, the job contains several DEFAULT commands that are used to set up the accounts and conditions of the compilation. Once established, these conditions require infrequent modification. For example, this job looks at :B01SI and :B010U to obtain source files and include files.

Finally, the IBEX command you would use to batch a compilation of TES$EXAMPLE would be:

```
!BATCH $JOB Z=TES$EXAMPLE
```

Here is a listing of the file $JOB

```
!JOB RERUN,WSN=W00,D=DFR,NAME=Z
!RESOURCE TIME=TIM,MEM=150
!DEFAULT TIM=10,'&'='',W00=LOCAL,$LO=0,$NOLS=0,$PRERR=0,VERS=B01C
!DEFAULT SIACCT=:B01SI,UIACCT=:CGUI,OUACCT=:CGOU,LSACCT=DLRHOST
!DEFAULT DFR=00:01
!LIMIT LO=2000
!LET STEPCC=0
!SET M$LO *&Z,CTG,ORG=UR,FUN=CREATE
!SET M$DO *TEMP,ORG=UR,CTG,FUN=CREATE
!PL6:
!IF $LO = 0 THEN GOTO NOLOCOMP
!PL6 Z.SIACCT,Z.UIACCT OVER Z.OUACCT(NSYS,SR(.OUACCT,.:B010U,.:B01X0U),LS,LO,SC)
!GOTO COMPDUN
!NOLOCOMP:
!PL6 Z.SIACCT,Z.UIACCT OVER Z.OUACCT(NSYS,SR(.OUACCT,.:B010U,.:B01X0U);
                                                          ,LS,SC)
!COMPDUN:
!R
!LET HOLDCC=STEPCC
!IF STEPCC = 0 THEN GOTO COMPOK
!ERR:
!C *TEMP OVER D:&Z.LSACCT
!IF $PRERR = 0 THEN GOTO END
!LISTER.X *&Z INTO,*TEMP (BA)
!GOTO EDG
!COMPOK:
!DEL D:&Z.LSACCT
!C ME INTO *TEMP
        ********************************
        *                              *
        *         NO  ERRORS           *
        *                              *
        ********************************
!IF $NOLS = 1 THEN GOTO END
!IF $LO = 0 THEN GOTO NOLO
!LISTER.X *&Z INTO,*TEMP (BA,LO)
!GOTO EDG
!NOLO:
!LISTER.X *&Z INTO,*TEMP (BA)
!EDG:
!EDGEMARK.X *TEMP ON LP@&W00 (L='Z',R='VERS',NA)
!END:
!SET M$LL NO
!LDEV LP01,ERASE,REMOVE
!LET STEPCC=HOLDCC
```

# MODULE 5-3

## A CP-6 System Program with its Own 'HELP'

This module presents an example of a CP-6 IBEX program that incorporates its own "HELP" facility. When the program is executed, the user is presented with a number of choices. In response to these program queries, the user can enter either the data required, or 'HELP'.

Example:

```
TEXT Options>HELP
```

The program has asked for TEXT options. The user responds by asking for HELP.

```
Enter TEXT options, separated by commas
Valid Options are:
  CB                        Change Bars
  CK                        Check
  CRPT=octalstring          Encryption mode
   .                         .
   .                         .
   .                         .
  WT                        Wait
```

In response to the users 'HELP', the program lists the TEXT options which may be entered. Then, the program again asks for options:

```
TEXT Options>  CK
```

The user responds with 'CK', indicating the check option.

This is how the feature looks in the IBEX command file:

```
           .
           .
           .
 81.000 !IF OPT~='HELP' THEN GOTO OPTPLACE1
 82.000 !OUTPUT 'Enter TEXT options, separated by commas'
 83.000 !OUTPUT 'Valid Options are:'
 84.000 !OUTPUT '  CB                          Change-Bars'
 85.000 !OUTPUT '  CK                          Check'
 86.000 !OUTPUT '  CRPT=octalstring            Encryption-Mode'
 87.000 !OUTPUT '  FN=name                     Form-Name'
 88.000 !OUTPUT '  FROM=name                   From-Page'
 89.000 !OUTPUT '  IN=n                        Indentation'
 90.000 !OUTPUT '  LS=n                        Line-Spacing'
 91.000 !OUTPUT '  NOF                         No-Fill-Mode'
 92.000 !OUTPUT '  NB                          Number-Printing'
```

```
 93.000 !OUTPUT '  NBB                          Number-Printing-Brief'
 94.000 !OUTPUT '  PGS=(n,n-n)                  Page-Selection'
 95.000 !OUTPUT '  PM=string                    Parameter'
 96.000 !OUTPUT '  SP                           Stop'
 97.000 !OUTPUT '  TO=n                         To-Page'
 98.000 !OUTPUT '  WT                           Wait'
100.000 !GOTO OPTQUER1
          .
          .
          .
```

# MODULE 6-0

# Section 6 - Use of Magnetic Tape in the CP-6 System

The use of magnetic tape in the CP-6 system involves cooperative effort between the user and the system operator. The user issues commands to allocate resources (tape drives), mount tapes for reading, writing, or copying, etc. The system operator mounts and dismounts tapes as requested, and also performs functions involved with tape security, use of the LABEL processor, automatic volume recognition (AVR), etc.

This section concentrates on tape usage from the user's point of view, but will often refer to the operator and system functions which make it possible for the user to perform his tasks using magnetic tape. The operator tasks and keyins related to magnetic tape use are described in detail in CE34, the Operations Reference Manual.

The modules in this section cover the following topics:

Module 6-1 Rules for Tape Usage
Module 6-2 Tape Commands, Options, and Calls
Module 6-3 CP-6 System Tape Processing
Module 6-4 How to Make Tapes That Other Machines Can Read
Module 6-5 Converting Imported Tapes for CP-6 Use
Module 6-6 How To Copy Tapes
Module 6-7 Multi-reel Tapes
Module 6-8 Tape Formats
Module 6-9 Tape Errors

# MODULE 6-1

# Rules for Tape Usage

## Tape Management

For the user, tape management is achieved:

- via PCL commands and PCL input and output options
- via IBEX commands and options of the IBEX SET command
- via monitor service calls

These PCL, IBEX, and monitor service features are summarized in Module 6-2, Tape Commands, Options, and Calls.

### PCL Commands

The MOUNT, REMOVE, and RELEASE commands control the physical mounting and removing of tapes. The REWIND, SPR, SPF, and SPE commands control the positioning of tapes. The SCAN command searches free tape. The WEOF command writes an end-of-file marker to a device.

In addition, many of the PCL commands related to files apply to files on tape as well as files on disk, i.e., COPY, COPYALL, etc.

A subset of the PCL commands can be issued directly from IBEX. These include: COPY, LIST, MOUNT, REMOVE, and REWIND. PCL commands are described in detail in the Programmer Reference manual, CE40, and in the PCL HELP facility.

## Basic Types of Magnetic Tapes

There are two categories of tape types in the CP-6 system: labeled (ANS) and free. The labeled category includes ANS, CP-6 Labeled, and EBCDIC. The free category includes free and managed free. The characteristics of these types are listed in the following table:

Types of Tape

| Type | Description |
|---|---|
| ANS | ASCII labels, ASCII data, filenames of 17 characters or less. Files are ORG= UNDEF, FIXED, or VARIABLE. |
| CP-6 Labeled | Superset of ANS which allows filenames up to 31 characters, includes File Information Table (FIT) for each file; files of any CP-6 ORG which includes KEYED and INDEXED files. Data written in binary. |
| EBCDIC Labeled | Like ANS except labels and data are written in EBCDIC. ORG=FIXED or VARIABLE, EBCDIC=YES. |
| FREE | Completely the user's responsibility to write and retrieve the data, non-standard labels, etc. Tape file management (TFM) doesn't care what is on these tapes. Data written in ASCII, binary, or EBCDIC. |
| Managed FREE | Like FREE but the records may be blocked. Files are ORG=UNDEF, FIXED, or VARIABLE. Data written in ASCII (or EBCDIC if ORG=UNDEF or FIXED). |

For data written in ASCII, 9-bit bytes in memory correspond to 8-bit bytes on tape (the high order bit of each byte is ignored). For data written in binary, eight 9-bit bytes in memory correspond to nine 8-bit bytes on tape.

## ANS Levels of Protection

The manner in which tapes are handled in any one CP-6 system is determined by the ANS protection level of that particular system. There are three levels of ANS volume protection: unprotected, semi-protected, and fully-protected. The protection level is set by the system manager. The characteristics of each level of protection are shown in the following table:

ANS Levels of Protection

| | POSSIBLE USER FUNCTIONS | | | | |
|---|---|---|---|---|---|
| TYPE OF ANS VOLUME PROTECTION | CHANGING VOLUME TYPE: FREE-> ANS ANS-> FREE | CONNECT FREE TAPE OR EXPIRED ANS VOLUME TO USER | CHANGE VOLUME SERIAL NUMBER | MAKE SCRATCH REQUESTS | WRITE ON UNEXPIRED ANS VOLUME |
| UNPROTECTED SYSTEM | Yes, without operator verification. | Yes - operator must specify serial no. and use MOUNT keyin for free tape.<br><br>For expired ANS tape MOUNT is optional.<br><br>(Applies to all three levels of protection) | Operator may use MOUNT keyin to change vol# until user is connected. | Yes - for free tapes and ANS volumes. | Yes |
| SEMI-PROTECTED SYSTEM | Yes, but operator must issue OVER keyin. | | No.<br><br>Yes - with FMSEC priv. | Yes - for free tapes and ANS volumes. | Only if operator issues an OVER keyin |
| FULLY PROTECTED SYSTEM | No.<br><br>Yes - with FMSEC priv. | | No.<br><br>Yes - with FMSEC priv. | No - not for any type of tape.<br><br>Yes - with FMSEC priv. | No.<br><br>Yes - with FMSEC priv |

## Free and Managed Free Tapes

The function of free and managed free tapes is to permit tapes with non-ANS format to be read and written on in the CP-6 system.

### Free Tapes

For free tapes, positioning, content, and data are completely user controlled. The user decides whether to read or write in ASCII or binary using the BIN option.

Note that data written in binary may be read in ASCII and vice versa. ASCII to EBCDIC translation on ASCII write and EBCDIC to ASCII translation on ASCII read may be requested when a free tape is opened using the EBCDIC and CNVRT options, and overridden on read or write using the TRANS option.

The user may also rewind the tape (M$REW), skip records in either direction (M$PRECORD), skip a tape mark in either direction (M$PFIL), write a tape mark (M$WEOF), and cause the free tape to be dismounted (unloaded) after the tape is closed (M$REM, M$CLOSE REM=YES).

### Managed Free Tapes

Managed free tape files are treated similarly to ANS labeled tape files with like organizations, and have the same restrictions (e.g., no record keys). Managed tape file records are blocked on output and deblocked on input according to the user's instructions (e.g. BLKL, RECL, ORG). Managed tapes may have EBCDIC data if ORG=FIXED or UNDEF.

Note that since there are no labels to describe the required deblocking of input files, the user's program must have an intimate knowledge of the input tape.

The primary application of managed free tape is in transferring character files of FIXED organization; the advantage of managed free over free tape being the automatic blocking/deblocking of records into physical tape blocks and subsequent saving of tape footage.

## Comparisons - Free and Managed Free

Both free and managed free tape are specified with ASN=DEVICE and a tape RES such as MT or FT. The difference is that for "real" free tape ORG=FREE while for managed free tape ORG is VARIABLE, FIXED or UNDEF. A subtle difference between free and managed tape on input is that when a free tape read hits a file mark, the end-of-file error is returned and tape position is left after the file mark. For managed tape, tape position is left before the file mark until the file is closed.
Labeled tapes are specified with ASN=TAPE. ORG defines the format of the tape file.

The function of labeled tapes in the CP-6 system depends on the file organization. CP-6 labeled tape files are intended to be used to backup or save CP-6 disk files. Any disk file can be saved on tape and restored to its previous condition. All CP-6 labeled tape file data is written in binary; no information is lost. CP-6 labeled tape files have standard ANS labels; they are always blocked and spanned except for RANDOM and IDS organizations. The block length is always a multiple of 4096 bytes.

## ANS Labeled Format

ANS format tape files are intended to be used to transfer character data to and from other systems with ANS tape capabilities. They may also be used to backup and save non-keyed character files.

All ANS format tape file data is written in ASCII; the high order bit of each CP-6 byte is lost. The user decides, where valid, whether ANS format tape files are to be created blocked and/or spanned and specifies a block length.

When saving character files for which the keys are unimportant, an ANS S-format tape file (ORG=VARIABLE, BLOCKED=YES, SPANNED=YES) with a large block size will use the least amount of tape.

ANS volumes have expiration dates coded on the tape itself. This date defines when the volume "expires" and may be re-used (written on). In the CP-6 system, the expiration date for the first volume on a tape is assumed as the expiration date for all volumes on that tape. Depending upon this date, the CP-6 system requires certain operator keyins for volumes, and limits access to volumes.

### EBCDIC Labeled Format

EBCDIC labeled format tape files are intended to be used to transfer data to and from EBCDIC systems. They have no particular use within a CP-6 shop. All EBCDIC files are written in ASCII mode causing the high order bit of each CP-6 byte to be lost. The user decides, where valid, whether EBCDIC format tape files are to be created blocked and/or spanned and specifies a block length.

### Mixing of CP-6, ANS, and EBCDIC Labeled

CP-6 labeled and ANS labeled tape files may be freely intermixed on the same volume set; such volume sets are called ASCII volume sets. EBCDIC labeled tape files may only reside on EBCDIC volume sets.

## Tape Fids

There are important differences between disk fids and tape fids. For ANS tapes, tape fids must follow the ANS restrictions for such names.

For full information about tape fids, see the Programmer Reference manual, CE40, or use HELP by inserting !HELP(IBEX)TAPE_FIDS.

### ANS Tape Fids

If the options used designate the tape fid as an ANS name, the name can only include those characters that are legal in an ANS name. A standard format ANS tape filename must be 1 to 17 characters: upper case, numeric, !"%$'()*+,./?;:<>+-_

Example:

```
!PCL
PCL COO here
<COPY A1811_DISK TO LT#0008/A1811&TEST (ORG=V, RECL=400, BL=3200)
```

The above commands result in an error message because '&' is not a legal character for an ANS file name. The use of ORG=V option means that the tape will be an ANS tape; as such the name must be a valid ANS filename. This is also true for the options ORG=F and ORG=UN.

### Tape Fids May Need Quotes

Tape fids that include unusual characters must be enclosed in quotes.

Example:

A user has an ANS labeled tape from another installation; he wants to use it as input to a FORTRAN run unit. He issues the following !SET command:

```
!SET F$1 LT#ABCD/DA395.DATA,BLK=25500,REC=255,ORG=FIXED,EB=YES
```

However, the job aborts. When DA395.DATA is enclosed in single quotes, as shown below, the job works.

```
!SET F$1 LT#ABCD/'DA395.DATA',BLK=25500,REC=255,ORG=FIXED,EB=YES
```

The problem here is that the 'period' (.) is a legal character in a standard ANS tape label, whereas in the CP-6 system a period is used to separate name from account. There is no account in a standard ANS tape label, so the .DATA is ignored unless the entire name is enclosed in quotes.

### Multi-reel Tape Fids

A valid labeled tape fid is LT#S1#S2#S3. PCL treats this as a 'unit' (called a volume set) rather than three separate tapes. See Module 6-7, Multi-reel Tapes.

## Acquiring Tape Drives

A tape drive is considered to be a CP-6 resource (such as memory, time, etc.). There are a number of ways in which the user can acquire tape drives (provided he has authorization):

- through the use of the IBEX RESOURCE, ORESOURCE, or ACQUIRE commands, which enable the user to directly acquire resources from the system.
- for online users, on the first reference to a tape via commands such as PCL MOUNT or COPY which result in the acquisition of a drive through operator keyin.

The CP-6 system supports tape drives that operate at 800, 1600, and 6250 BPI, and any one system may have a mixture of these capabilities. Individual dual-density drives operate at 800/1600, or 1600/6250 BPI. This is discussed in detail under single-density and mixed-density systems in this module.

Generally speaking, when you want to use a scratch tape or to mount a specific tape by serial number, you need not specify BPI and can leave this to be assigned by the system. However when you are concerned with acquiring a drive or drives with specific density capability or mixed-density capability, you are best advised to acquire those drives through the use of the RESOURCE, ORESOURCE, or ACQUIRE commands.

You can check to see what tape drives are assigned to you by using the IBEX DISPLAY command with the RESOURCES option, i.e., DISPLAY RESOURCES.

## MOUNT Command

Before using the MOUNT command, you should alert the operator by using the IBEX MESSAGE command:

```
!M PLEASE MOUNT A SCRATCH TAPE WITH RING AS LT#ABLE
```

This sends a message to the operator. RING tells the operator to place a ring in the reel of the tape so that it can be written on.

```
!MOUNT LT#ABLE RING
!COPY STUFF TO LT#ABLE/STUFF
```

You next use the MOUNT and COPY commands to initiate copying the file STUFF to labeled tape ABLE. When the operator has taken appropriate action to mount the tape, and the AVR process is completed, a bang (!) prompts you to begin. Since you have not specified drive or density, you have been assigned the system default density and drive.

## RESOURCE, ORESOURCE, and ACQUIRE Commands

The IBEX ORESOURCE, RESOURCE, and ACQUIRE commands are used to request tape drives: ORESOURCE for the online user, RESOURCE for the batch user, and ACQUIRE to request additional resources for the online user. When you use these commands to request a tape drive, you may either request a drive by logical name (such as MT01, MT02, etc.) or you may allow the system to assign the resource name by default. It is very important to use resource names when using tapes of different densities.

### Default Tape Drive Assignments

MTDFLT is an option of the MON command in the TIGR deck that describes the attributes of the default tape drive. If nothing is specified for MTDFLT in the TIGR deck, the system will assign to MTDFLT the attributes of the first drive that is declared in the deck. For example, if the first declared drive is a 1600/6250 device, the default drive you get for a simple online tape copy is a 1600/6250 device; the system won't allow you to use a 800/1600 device unless it is specifically named. However, if MTDFLT had been set to 1600, the system would allow you to use either a 1600/6250 device or an 800/1600 device as the default drive.

A drive may only be used at the density or densities logically acquired, regardless of the densities the physical drive is capable of. This is true both for default tape drive assignments and directly acquired drives.

For example, for a default assignment, if MTDFLT has been set to 1600, the drive acquired through default can only be used at 1600 BPI even though it is physically capable of 6250 BPI.

If a device has been acquired directly, for example by

```
!RES MT01(1600)
```

that device is logically acquired at 1600 BPI and can only be used at that density. However, if the device had been acquired by

```
!RES MT01(1600,6250)
```

it can then be used at either 1600 BPI or 6250 BPI.

## Single-density and Mixed-density Systems

The CP-6 system currently supports two different types of tape drives (from a resource management point of view): 800/1600 BPI drives, and 1600/6250 BPI drives. A "single-density" system contains only one type of drive. A "mixed-density" system contains a combination of 800/1600 and 1600/6250 drives.

Problem:

In a single-density system, a user specifies the following:

```
!JOB
!RES MT(1600)=1,MT(800)=1
!PCL
COPY LT#4839/DATA TO LT#XX31#XX32/DATA(EB,DE=800)
```

This results in an error message. However, with the following change the job is accepted:

```
!JOB
!RES MT01(1600),MT02(800)
!PCL
COPY LT01#4839/DATA TO LT02#XX31#XX32/DATA(EB,DE=800)
```

If several tape devices are acquired that have different attributes (densities), they should be referenced by their full name (i.e. MT05 not just MT) to assure getting the desired association between density and drive.

Example:

```
!JOB
!RES MT01(800),MT02(1600,6250),MT03(6250)
!PCL
COPY MT01#XXXX/AFILE TO MT02#YYYY/BFILE(DEN=1600)
REM #YYYY
COPYALL MT02#ZZZZ TO MT03#AAAA
```

In the above example, the RESource command acquires three drives. MT01 is an 800BPI only drive. MT02 is a 1600/6250 BPI dual density drive. MT03 is a 6250 BPI only drive. The COPY command copies the file AFILE from the 800 BPI tape #XXXX. The COPYALL command copies all the files from the tape ZZZZ to the 6250 BPI tape #AAAA. The #ZZZZ tape may have been written at either 1600 or 6250 BPI.

Density specification is also honored during FUN=IN or FUN=UPDATE tape opens if the tape's density can not be determined during AVR. For a description of AVR, see Module 6-3. Once the density is set, it can not be changed.

## Examples of Online Use

```
!M    PLEASE MOUNT A SCRATCH TAPE WITH RING AS   LT#ABCD
!M   PLEASE MOUNT ANOTHER SCRATCH TAPE WITH RING AS FT#1234
!PCL
PCL COO here
<MOUNT FT#1234 RING
<MOUNT LT#ABCD RING
```

Via the IBEX command message, you send two messages to the operator requesting that two scratch tapes be mounted. One is free tape #1234; the other is labeled tape #ABCD. You then enter PCL and formally request the mount of the tapes via the MOUNT command. Note that if the tapes had been initialized prior to the session, the messages to the operator would have been unnecessary.

```
<REW FT#1234
<COPY ME TO FT#1234
.
```

You rewind the free tape only (to make certain that tape is at beginning after other activity) and use the COPY ME command to prepare for online input. Input can be placed in the file once you have been prompted with a period (.).

```
.This is the first record on the free tape.
.This is the second record on the free tape.
.This is the third record on the free tape.
.<F>
<WEOF FT#1234
```

In this example, you have written three records. You use <ESC><F> to indicate the end of file input, then use the WEOF command after the last file on the free tape to mark the end of the volume.

```
<REW LT#ABCD
<COPY LESLIE TO LT#ABCD/LESLIE
..COPYING
<COPY HELEN TO LT#ABCD/HELEN
..COPYING
```

You rewind the labeled tape and copy files LESLIE and HELEN to it.

```
<LIST LT#ABCD
LESLIE          HELEN
..   2 files listed
<SPF LT#ABCD/HELEN
<COPY MARTA TO LT#ABCD/MARTA
..COPYING
<LIST LT#ABCD
LESLIE          MARTA
..   2 files listed
```

You list the files on the labeled tape, then issue a SPF command that positions the tape to the beginning of file HELEN. You then copy file MARTA to the labeled tape and relist the tape. Note that file MARTA has been written over file HELEN.

```
<REW FT#1234
<COPY FT#1234 TO ME
This is the first record on the free tape.
This is the second record on the free tape.
This is the third record on the free tape.
<REW FT#1234
<SPR FT#1234 1
<COPY FT#1234 TO ME
This is the second record on the free tape.
This is the third record on the free tape.
```

You now rewind the free tape and issue a COPY command that displays the records on that tape. You rewind the tape, issue an SPR command that positions the tape one record forward, and then use another COPY command which results in the display of only records 2 and 3.

```
<REW FT#1234
<SCAN FT#1234
FT#1234     FILE     RECS     MAXLEN     1600 BPI
               1        3         42
```

You rewind the free tape and issue a SCAN command. The resulting display provides the tape number, file information, and density.

```
<RELEASE LT#ABCD
```

You request that the labeled tape be removed from the tape drive and release the tape drive so that other users can allocate it for their tasks.

## File Sequence Numbers

The names of files on a particular tape volume do not need to be unique. The File Sequence Number (FSN) may be used to identify a specific file which has a non-unique name.

The following example shows how you can list tape files to obtain the FSN.

Example:

```
!MOUNT LT#FROG RING
!COPY CE33_TEST TO LT#FROG/FILE1
```

The tape is automatically labeled (if it was not previously), as a file (FILE1) has been created.

```
!REWIND LT#FROG
!L LT#FROG
FILE1
```

You rewind the tape, request a listing of the files in LT#FROG, and are given FILE1.

```
!COPY CE32_TEST TO LT#FROG/FILE2
..COPYING
!L LT#FROG
FILE1        FILE2
..     2 files listed
```

You copy a file to FROG to be named FILE2. Once completed, a listing of the file names and their sequence numbers can be obtained. Copies to tape always begin at the current tape position, regardless of tape content, unless certain options are specified.

```
!L (A) LT#FROG
14:36 AUG 21 '82 LT#FROG.
ORG TY  BLKL  RECL     REC VOL       FSN NAME
CON     4096     0       2 FROG        1 FILE1
KEY     4096     0    1115 FROG        2 FILE2
..    2 files listed
```

The files and their attributes are listed.

```
!COPY CE32_03 OVER LT#FROG/FILE2
..COPYING
!L (A) LT#FROG
14:37 AUG 21 '82 LT#FROG.
ORG TY  BLKL  RECL     REC VOL       FSN NAME
CON     4096     0       2 FROG        1 FILE1
KEY     4096     0    1115 FROG        2 FILE2
KEY     4096     0    1092 FROG        3 FILE2
..    3 files listed
```

Another file has been copied to FROG as FILE2. Note that in this example two distinct files have the same file name. The File Sequence Number (FSN) distinguishes them.

The first file on the tape is not copied over only because listing tape contents causes position to be left after the last file on the tape; copies to tape always begin at the current tape position (regardless of tape content) unless certain options (e.g., XTEND or PHYS) are used.

This example assumes that the site is running in ANS unprotected mode. If protected mode is used, you cannot write on a tape unless it has been labeled by the system manager/operator prior to use, or is expired.

Due to the label and tape mark structure imposed by the ANS standards, opening a file by its sequence number is much faster than opening the same file by its name. For name opens, all labels must be inspected to find the name. For sequence number opens, the arrangement of tape marks gives a precise determination of file location.

A tape file is opened by file sequence number by means of the PCL FSN input option, the !SET command FSN option, or the monitor services M$DCB and M$OPEN FSN options.

The following example illustrates the retrieval of a file using the PCL FSN input option:

```
!PCL
PCL C00 here
<MOUNT LT#FROG
<COPY LT#FROG (FSN=(2-6)) TO NORMA
```

Specifies that files with FSNs 2 through 6 are to be read sequentially from the tape LT#FROG and copied to the disk file NORMA. FSN may not be specified if a filename or range is included in the source parameter. The MOUNT command is not needed if the tape is already mounted.

The following example demonstrates the importance of being able to retrieve a file by FSN when there is more than one file on the tape with the same name.

```
<MOUNT LT#FUZZY RING,REEL=EDWARD
<C ME TO LT#FUZZY/BEAR
.CE54 Status Summary
.
<C ME TO LT#FUZZY/BEAR
.    Datafair Trip Report
.    (March 21-25 1983)
.
```

You mount tape LT#FUZZY and build two files with the same name (BEAR) on the tape.

```
<L LT#FUZZY(A)
10:49 AUG 17 '83 LT#FUZZY
ORG TY  BLKL  RECL     REC VOL           FSN NAME
CON     4096     0       1 FUZZY           1 BEAR
CON     4096     0       2 FUZZY           2 BEAR
..    2 files listed
```

Then you use the PCL LIST command to list tape LT#FUZZY with file attributes; the listing shows the two files that have the same name.

```
<REW LT#FUZZY
<C LT#FUZZY/BEAR TO ME
CE54 Status Summary
<C LT#FUZZY/BEAR TO ME
    Datafair Trip Report
    (March 21-25 1983)
```

You rewind LT#FUZZY and attempt to retrieve the file BEAR by NAME, but discover that you may get either file, depending on your current position on the tape.

```
<C LT#FUZZY(FSN=1) TO ME
CE54 Status Summary
```

By specifying the file by FSN, you can get the desired file no matter where it is located on the tape.

```
<REM LT#FUZZY
```

Since you're done with the tape now, you use the REMOVE command to dismount tape LT#FUZZY.

## Hardware Limitations

The following hardware limitations apply to the use of magnetic tapes in the CP-6 system.

### Minimum Record Size

When writing a free tape, there is a limitation as to the minimum size a record must be before it can be successfully written. This limit is four bytes. The four-byte lower limit is a hardware restriction which is detected by software. There is no similar problem with labeled tapes (LTs) since records are blocked and (if necessary) padded to obtain a data block which is acceptable.

### Lost Data

Tape hardware will not inform the user of lost data conditions on a read. For this reason, when reading a free tape it is important to know how much data resides on the tape, or to use a buffer size larger than the largest expected record.

### Number of Bytes Stored on Tape

Information on tape is stored in 'frames'. On a 9-track tape, each frame contains nine bits. Eight of these bits are used for data; one is used to check parity.

Honeywell machines use a word size of 36 bits. Each word contains four 9-bit bytes. The hardware automatically rearranges each eight 9-bit bytes into nine 8-bit data frames (plus one parity bit each) for a convenient fit on a 9-track tape during binary write operations. Thus when read back in, data is in multiples of eight bytes.

In addition, the tape drives will write only integral numbers of words. Thus when four bytes (one word) are written, five frames (or 40 bits) are placed on tape. Four bits of that last frame are padding, not actual data.

## File Management Buffers

When using magnetic tape, you should give special consideration to the size of the file management buffers (FPOOL) as set by the IBEX LIMIT command. This is particularly true when reading or writing tapes with large blocks. For example, a block size of 28672 requires a seven-page file buffer (one page = 4096 bytes) or FPOOL=7. If you were both reading and writing tapes with this block size, a minimum of FPOOL=14 should be used. To this number should be added the number of buffers required to perform miscellaneous functions (at least two to three).

While the default FPOOL buffer allocations supplied by the CP-6 system will be adequate for many tasks, you should be aware that this allocation may not be sufficient for circumstances involving large blocks or very large records.

## DCBs

The DCB assignments and options values assigned by the SET command or by monitor services remain effective throughout the job or session unless modified

- at a subsequent job step by another SET command or monitor services call,
- at a subsequent job step by a RESET command which resets some or all DCB parameter values to the default assignments by deleting the existing assignments,
- during the job step by the ADJUST command. ADJUST assigns the specified option values to the specified DCB, but the assignment does not survive the job step.

See the discussion of DCBs in CE40, the Programmer Reference Manual.

# MODULE 6-2

# Tape Commands, Options, and Calls

## Introduction

This module contains summary tables of the IBEX and PCL commands and options that are related to the use of magnetic tape. Also included is a summary of monitor service calls that are used in the management of magnetic tape in the CP-6 system.

## PCL Tape Control Commands

A subset of the PCL commands can be issued directly from IBEX. These include: COPY, LIST, MOUNT, REMOVE, and REWIND. PCL commands are described in detail in the Programmer Reference manual, CE40, and in the PCL HELP facility.

The PCL tape control commands are summarized in the following table.

PCL Tape Commands

| Command | Description |
|---|---|
| MOUNT | Requests that a certain tape be mounted on a system tape drive. When a MOUNT command is issued, the operator is informed of the request. No further commands can be issued until the requested mount has taken place. If the BREAK or <CNTRL>Y key is struck while a MOUNT request is pending, the request is terminated, and a cancellation message is sent to the operator. <CNTRL>Y cancels the MOUNT command and gives control to the command processor (IBEX). If the user issues a GO command to IBEX, PCL will resume execution and reissue the MOUNT command. |
| REWIND | Rewinds a tape to its beginning. If it is a multi-volume set positioned at any reel other than the first, the current reel is dismounted. |
| SPE | Positions a tape just beyond the end of its last file. If the file specifies a free tape, the tape is positioned forwards until two adjacent tape marks are encountered and is then positioned between them. If the fid specifies a labeled tape, the tape is positioned after the last file. |
| SPF | Positions a tape to a specified file. If the fid specifies a filename on a labeled tape, the specified file becomes the current file, and positioning takes place forwards or backwards from that file. |

| Command | Description |
|---|---|
| SPR | Positions a free tape forwards or backwards a specified number of records. It does not allow positioning beyond the tape marks. If the positioning operation encounters a tape mark, the user is informed and the tape is positioned just inside the offending tape mark. |
| SCAN | Searches a free tape from present position to the first encountered double tape mark, and reports the recording density of the tape, the number of records accessed, and the length of the longest record in each file. |
| REMOVE | Requests that a certain tape be removed (dismounted) from the system tape drive. |
| RELEASE | Releases a tape drive back to the system. If a tape is currently mounted, it is removed. |

## PCL and !SET Output Options

A subset of PCL options are of particular importance to the user of magnetic tapes. A similar subset of the IBEX SET command options perform many of the same functions, but the PCL and SET command options do have differences. These options are summarized below.

### !SET Command Options

SET command options related to the use of tape are shown in the following tables. For more information, see CE40, Programmer Reference manual, or use HELP:

```
HELP (IBEX) SET_OPS option
```

For example,

```
HELP (IBEX) SET_OPS SPANNED
```

| Option | Description |
|---|---|
| ACCESS | Establishes associated accounts and their permissions. |
| ACS | Defines how file access will occur: sequentially, or in journal mode. |
| ACSVEH | Defines a vehicle control list that allows the file creator to control access by limiting it to processors. |
| BLKL | Specifies the maximum physical tape block size (in bytes) that will be read from or written to a labeled tape. |
| BLOCKED | Controls packing of logical records into physical tape records. |
| CVOL | Specifies that the user is to be notified of end-of-volume conditions. |
| DENSITY | Specifies the recording density at which a tape is to be written. |
| EBCDIC | Controls conversion of tape file labels, user labels, and the data itself from ASCII to EBCDIC. |
| EXPIRE | Determines the expiration date of the volume set for ANS tapes. |
| FSN | Specifies the file sequence number for labeled tapes. |
| FUN | Specifies whether a file can be read and/or modified. |
| KEYL | Specifies the length of a key for an indexed file. |
| KEYX | Specifies the key starting position for an indexed file. |
| MAXVOL | Specifies the maximum number of tapes to be used as scratch tapes to supplement the explicit volumes requested by serial number. |
| NRECS | Specifies the number of records in a relative file. |
| ORG | Defines the file organization; distinguishes between ANS, EBCDIC, managed free, and free tape. |
| READ | Specifies the accounts that may read but not write the file. |
| RECL | Specifies maximum record length. |
| SN | Specifies a list of tape serial numbers. |
| SPANNED | Determines whether logical records may be divided between physical tape records. |
| TYPE | Specifies the file type. |
| VOLACCESS | Controls access to read and write on the volume. |
| WRITE | Specifies the accounts that may have write access to a volume. |

!SET Options (Tape) (cont.)

| Option | Description |
|---|---|
| XTEND | Determines the position for CREATE opens of labeled tape. |

## PCL Input Options

The following PCL input options apply to tapes. Additionally, options that apply to CP-6 files will also apply to files on tape when appropriate. For more information, see CE40, Programmer Reference manual, or use HELP:

```
HELP (PCL) INOPS option
```

For example,

```
HELP (PCL) INOPS DEOD
```

PCL Input Tape Options

| Option | Description |
|---|---|
| BINARY | Specifies that any read operation from a free tape (FT) is to interpret data as binary. The default is NBINARY (see the NBINARY option). |
| BLOCK | Specifies the block length for input from free tape. |
| DEOD | Specifies that all files in a free tape (FT) are to be read as if there were only one file until two consecutive tape marks (EODs) are encountered. |
| EBCDIC | Specifies that tape file labels and records read from free tape (FT) be translated from EBCDIC to ASCII. The default is no translation. |
| FSN=(n-m) | Specifies that files n through m are to be read sequentially from a tape. The value of m must exceed the value of n, and both must be between 1 and 9999. FSN may not be specified if a filename or range is included in the source parameter. If m is omitted, only file n is assumed. |
| FTORGANIZATION | Specifies the input file organization for free tape. Organization may be FREE, UNDEFINED, FIXED, or VARIABLE. The default is FREE. |
| NBINARY | Specifies that any read operation from a free tape (FT) is to interpret data as character rather than as binary. The default is NBINARY. (See the BINARY option.) |
| NBLOCKED | Specifies that the input free tape is not blocked. The default is blocked. |

PCL Input Tape Options (cont.)

| Option | Description |
|---|---|
| NCNVRT | Specifies EBCDIC data read from EBCDIC tapes is not to be translated into ASCII. The default is translation. |
| NSPANNED | Specifies that the input free tape is not spanned. If unspecified, it is assumed to be spanned. |
| ORGANIZATION | Specifies that only files of certain organizations are to be selected. |
| PHYSICAL | Determines how the range specification of file names will be interpreted. Specifies that a range of file names in a source parameter is to be interpreted physically; this only applies to labeled tape. PCL positions to the first file in the range specification and then consecutively accesses files until the second file in the the range specification is located and accessed.<br><br>If PHYSICAL is not specified, all files whose filenames are alphabetically between the two filenames in the range specification are accessed. |
| RECLENGTH | Specifies the logical record length for input from managed free tape. |
| TRANSPARENT | Specifies input without translation. |
| VOLUME | Specifies which volume of a tape set is to be accessed initially. The value can range from 1 to 511. |

## PCL Output Options

The following PCL output options apply to tapes. Additionally, options that apply to CP-6 files will also apply to files on tape when appropriate. For more information, see CE40, Programmer Reference manual, or use HELP:

```
HELP (PCL) OUTOPS option
```

For example,

```
HELP (PCL) OUTOPS DENSITY
```

PCL Output Tape Options

| Option | Description |
|---|---|
| BINARY | Specifies that all records are to be written as binary data. The default is for records to be written in the same form as they were input. |
| BLOCK | Sets the maximum block size for files on tape. The size is expressed in number of bytes per block. If BLOCK is not specified, a format dependent default will be supplied. The maximum is also format dependent. |
| DENSITY | Specifies the tape recording density. Valid densities are: 800, 1600, 6250, or 556 (valid for 7T only). |
| EBCDIC | Specifies that the labels on labeled tape are to be written in EBCDIC; data will also be written in EBCDIC unless NCNVRT is included as one of the output options. Valid only for labeled tape, managed file types F and U, and free tape (FT). |
| MAXVOL | Specifies the maximum number of additional tape volumes that can be accumulated as part of the volume set. (When you run out of space writing on a labeled tape, the system can allocate scratch volumes to be added to the set.) The value must be in the range of 1 to 511. The default is 511. Valid only for labeled tape. |
| NBLOCKED | Specifies that tape output is not to be blocked. See the BLOCK option. |
| NCNVRT | Specifies that data is not to be translated into EBCDIC format n the EBCDIC labeled tape. Valid only for labeled tape and only when the EBCID option is included among the output options. |
| NSPANNED | Specifies that records are not to be spanned. Spanned records may be divided between physical tape records and may exceed the record size specified in the RECLENGTH option. Valid only for labeled tapes. |
| ORGANIZATION | Specifies the organization of the output file. For free tapes, ORG=FREE; for managed free tapes and for ANS tapes, ORG=UNDEFINED, FIXED, or VARIABLE. For CP-6 Labeled tapes, all values of ORGANIZATION are acceptable. For EBCDIC tapes, ORG cannot be VARIABLE. |
| RECLENGTH | For fixed record length formats such as RELATIVE and FIXED, the RECL option is the maximum record length in bytes. For variable record length TAPE formats such as CONSEC, KEYED, and VARIABLE, the RECL option is the maximum record segment length in bytes. For unspanned files, at most one record segment is used per record thus limiting each record. Records exceeding this length are truncated. All variable record length TAPE formats include control information bytes with the data record. |
| TRANSPARENT | Specifies input without translation. |
| VOLACCESS | Specifies volume access mode. |

PCL Output Tape Options (cont.)

| Option | Description |
|---|---|
| VOLUME | Specifies which reel in a multi-volume tape set is to be accessed initially. Valid only for labeled tape. |
| XTEND | Prevents overwrite of other files on the tape by writing the new output file at the end of the labeled tape set. If unspecified, the write occurs at the current tape position. Valid only for labeled tape. |

### PCL vs SET

As can be seen from the above tables, PCL and SET options are sometimes similar, but not identical. For example, block length in !SET is BLKL; in PCL it is BLOCK. The following examples show the same options in !SET and PCL:

```
SET:  !SET INFILE FT#1234, ORG=FIXED, RECL=20, BLKL=60

PCL:  !COPY FT#1234(FTORG=FIXED, RECL=20, BL=60)
```

## Monitor Service Calls

Monitor service calls related to magnetic tapes are summarized here. For more complete information about monitor service calls, see the Monitor Services Reference manual, CE33, or use HELP:

```
    HELP (MONSER) call
```

for example,

```
    HELP (MONSER) M$OPEN
```

## M$DCB

The DCB is the communication data block between the user and the monitor concerning the attributes of a disk file, labeled tape file, or device. The following M$DCB options are of particular importance to users of magnetic tapes. The options in the table are also options used with M$OPEN.

M$DCB and M$OPEN Tape Options

| Option | Description |
|---|---|
| ASN | Indicates whether the DCB parameters describe a DISK file (FILE), a labeled tape (TAPE), a specific device (DEVICE), or a communication group (COMGROUP). |
| BLKL | Specifies the maximum physical tape record size (in bytes) that will be read from or written to a labeled tape. |
| BLOCKED | YES specifies that logical records and record segments are to be packed into physical tape records. NO results in, at most, one record or record segment per tape record. BLOCKED applies only to FIXED and VARIABLE labeled and managed tape files. BLOCKED should be used in conjunction with SPANNED for optimum tape record utilization. |
| CNVRT | YES specifies that for EBCDIC tape files, data is to be translated from EBCDIC to ASCII after reading, or ASCII to EBCDIC before writing. Translation is done while data is being moved between user and monitor buffer for all file formats except UNDEF, for which translation is done in the user's buffer. (This destroys original contents when writing.) CNVRT applies only to FIXED, VARIABLE, and UNDEF formats.<br><br>For nonbinary writes to free tape, if CNVRT is set and EBCDIC is not, the user's buffer is converted to assure that no high order bits are set (which will cause an I/O error). If both CNVRT and EBCDIC are set for free or managed free tapes, normal translation takes place. The default is YES. |
| CVOL | YES specifies that the user desires to be notified of end-of-volume conditions. CVOL should only be specified for SPANNED=NO tape files since spanned tape files may have records which cross volume boundaries. It is not possible to process these records if the user takes CVOL control.<br><br>For free tapes, end-of-tape will be reported to permit the user to M$CVOL to the next output tape or M$WRITE which automatically mounts the next output tape. For read operations, two file marks are interpreted to indicate end-of-volume. A read encountering two file marks will 1) if CVOL was specified, return an end-of-file error and leave tape position after the first file mark or 2) if CVOL was not specified, cause the next volume to be automatically mounted and read.<br><br>The default is NO in which case volume changes are done automatically. |

| Option | Description |
|---|---|
| DENSITY | The recording density at which a tape is to be written. DENSITY may be specified only when creating the first file of a volume set. All following files are created at the same density as the first. For IN or UPDATE opens, DENSITY is determined from the tape volume containing the opened file. |
| EBCDIC | YES specifies that tape file labels (including user labels) are to be translated from EBCDIC to ASCII on input and from ASCII to EBCDIC on output. Data is subject to translation (see DCB.CNVRT). EBCDIC applies only to VARIABLE, FIXED, and UNDEF files on labeled and managed tape, and to free tape. EBCDIC must be specified on output and input for free and managed tape files. |
| EXPIRE | For labeled tape volumes in protected systems, EXPIRE specifies the number of days to protect the volume against content changing operations (UPDATE or CREATE). In semi-protected systems, unexpired volumes require an OVER keyin to UPDATE or CREATE open. EXPIRE has no effect in unprotected systems. The expiration date of the entire volume set is that of the first file of the volume set. |
| FSN | The file sequence number (FSN) for labeled tape indicates the position of the tape file relative to the beginning of the set, with first file of volume set numbered 1. |
| FUN | Applies to tape files used to read, update, or create a file. Unless otherwise specified, record positioning is to the first data record in the file.<br><br>IN - Used to read records from a file. Specifies the read only mode.<br><br>UPDATE - Used to extend an existing tape file. Specifies the read and write mode. On opening an existing tape file for UPDATE, the file will be positioned at the end.<br><br>CREATE - The type of file open that CREATE performs will depend on the parameters of the M$OPEN option EXIST. Specifies the write and read mode. If the file is cataloged at M$OPEN time (CTG=YES), the function effectively changes to UPDATE (FUN=UPDATE). |
| MAXVOL | Specifies the number of extra tape volumes which will be requested as scratch tapes after the volumes in the serial number list are used up. |
| NAME | Specifies the name of the file to which the DCB is to be assigned. For UNDEF, FIXED, or VARIABLE tape file names, NAME must be a 17-character legal ANS name. For other tape files, the name may consist of up to 31 alphanumeric characters from the following character set: A-Z,a-z,0-9,:,$,_,-. |

| Option | Description |
|---|---|
| ORG | File organization. For ASN=TAPE, ORG is meaningful for FUN=CREATE opens only.<br><br>CONSEC - specifies that the records in the file are consecutively organized and each record will be processed sequentially.<br><br>FIXED - specifies ANS or EBCDIC tape format F - fixed-length (RECL) records with no control information. (See EBCDIC, CNVRT, BLOCKED.)<br><br>FREE - specifies free tape (i.e., not managed free tape for which FIXED, UNDEF, or VARIABLE must be specified).<br><br>UNDEF - specifies ANS or EBCDIC tape format U - undefined-length records with no control information. (See EBCDIC, CNVRT.)<br><br>VARIABLE - specifies ANS tape format D for unspanned files, ANS tape format S for spanned files, and format V for EBCDIC files - variable length records. (See EBCDIC, CNVRT, BLOCKED, SPANNED.)<br><br>The default for disk and tape files is CONSEC. The default for free tape is FREE. |
| RECL | For fixed record length formats such as RELATIVE and FIXED, RECL is the maximum data record length in bytes. For VARIABLE formats, RECL is the maximum record segment length in bytes. For unspanned files, at most one record segment is used per record, thus limiting each record to RECL bytes which includes 4 bytes of record information. For spanned files, RECL is the maximum length of the data record not including record information bytes. RECL may be zero for spanned files indicating that there is no maximum.<br><br>RECL is determined for IN or UPDATE opens from the file opened, except for managed tape files for which RECL must be specified. |
| SEQ | YES specifies that sequencing is to occur on each output record. |
| SPANNED | YES specifies whether logical records may be divided between physical tape records. Spanned records may exceed the record limit imposed by RECL, since RECL applies to record segment size. SPANNED applies only to VARIABLE labeled and managed tape files. SPANNED is determined for IN or UPDATE opens from the tape file opened. Spanned and unspanned files may be freely intermixed on a volume.<br><br>SPANNED should be used in conjunction with BLOCKED for optimum tape record utilization. (SPANNED must be specified for INPUT or UPDATE managed tape files.) |

| Option | Description |
|---|---|
| | The default is YES which permits records of any length and results in the most efficient usage of tape records. |
| VOL | On open, VOL specifies which volume of the volume set (as specified by the serial number list) is to be initially mounted. VOL is used in conjunction with XTEND to determine which volume of the volume set to Any processor can determine current volume by using VOL as an index to the serial number table. See M$OPEN SN. |
| VOLACCESS | Indicates labeled tape volume set access limitations. |
| XTEND | For CREATE opens of labeled tape files, YES causes the volume set to be positioned after the last file if VOL=0. If VOL is non-zero, the volume selected is positioned after its last file. XTEND = NO causes the next file to be created at current volume position. The default is NO. |

---

Note: For additional M$OPEN tape options, see next table.

## M$OPEN - OPEN DCB

The M$OPEN monitor service performs the functions necessary to give a user access to the I/O medium through a DCB. M$OPEN and M$DCB share the tape options shown in the preceding table. The additional M$OPEN options in the following table are also used with magnetic tape processing:

Additional M$OPEN Tape Options

| Option | Description |
|---|---|
| MNTONLY | YES specifies that the initial tape volume is to be mounted without opening any files. The tape volume is positioned to its beginning. The file sequence number (FSN) and file section number (FSECT) of the first file (section) are returned in the DCB for labeled tapes. DCB.ASN is set to TAPE for labeled tapes, and to DEVICE for free or managed tapes. |
| SN | Locates an area containing a list of tape serial numbers or a pack set name. This list must be in the same order in which the volume set was created and cannot contain duplicate or blank serial numbers. Blank serial numbers occurring at the end of the list will be counted in MAXVOL. Serial numbers must conform to ANSI standards. |
| UHL | Locates the VLP_ULBL into which User Header Labels (UHLs) are to be read during an IN or UPDATE open of a labeled tape file, or from which UHLs are to be written during a CREATE open. The label number and contents must conform to ANSI standards. |

## M$CLOSE - CLOSE DCB

The M$CLOSE service terminates and inhibits I/O through a specified DCB, until the DCB is again opened. For tape updating, the M$CLOSE service performs these additional functions:

- Performs end-of-file processing appropriate to tapes:

  For free or managed tape, if the last operation performed was a write, two file marks are written and the tape positioned between them. If the last operation was a write-end-of-file, one file mark is written and the tape positioned before it.

  For labeled tape, User Trailing Labels (UTLs) may be specified to be written following the End Of File label group for CREATed or UPDATEd (if modified) files. A buffer may be specified for receipt of any UTLs present for IN or UPDATE file if the file was not modified. Tape position is left following the file mark which follows the end of file label group (including UTLs) based upon the specification of the POS parameter.

- Positions the tape following end-of-file processing, if the POS parameter is specified.

## M$CVOL - CLOSE VOLUME

The M$CVOL service causes the monitor to terminate the reading or writing of data in the magnetic tape reel currently associated with a specific DCB, and to advance to the next reel of the volume set.

For free and managed tapes, the current reel is terminated and the next reel positioned to its beginning.

For input files on labeled tape, the current file section is positioned to its end and determination made (by reading the following label group) of the existence of a subsequent file section. If no such file section exists, M$CVOL will ALTRET with an end-of-file error and leave the current file positioned to its end. If a subsequent file section exists, the current volume is dismounted, and the next mounted and positioned to the first record of the next file section. Any UTLs following the EOV label group are returned if UTL is specified. Any UHLs following the HDR label group of the next file section are returned if UHL is specified. Both UTLs and UHLs are returned in VLP_ULBL format.

For output files on labeled and managed tape, the current file section is truncated after the last whole record.

In addition, for output files on labeled tape, an EOV label is written followed by any UTLs specified by UTL. The current volume is dismounted, and the next mounted and positioned ready to write the next record of the file (first of the next file section) after writing necessary VOL1, HDR and UHL labels as required or specified by UHL.

Note: On input, UTLs and UHLs are returned to their separate buffers contiguously packed. The first word of the user label areas will contain the number of labels of that type returned. On output, UTLs and UHLs are written from the user's respective label buffers which are contiguously packed. The first word of the user label areas will contain the number of labels of that type to write. Any labels not containing the proper first three characters will cause termination of user label writing.

CVOL control should only be used with unspanned tape files, since spanned tape files may have records which cross volume boundaries. It is not possible to process these records if the user takes CVOL control.

## M$READ - READ RECORD

The M$READ service causes a specified data record to be read into a user buffer in memory. The M$READ service is used for all types of files and devices for which input is appropriate.

The user normally provides a buffer that is large enough to contain the maximum length record. In the normal case if a record exceeds buffer size, the record is truncated and this condition is reported as an error. If a record is smaller than buffer size the remainder of the buffer is unchanged from its previous contents. The M$READ service also provides the option (CONT) to issue several calls to read successive portions of a single large record.

## M$WRITE - WRITE RECORD

The M$WRITE service causes a data record stored in a buffer in memory to be written. The M$WRITE service is used for all types of files and devices for which output is appropriate.

The user normally provides a buffer to accommodate the maximum length record. To write records of varying sizes, the user adjusts the FPT field BUF_ which contains a vector: BUF_.BUF$ points to the start of the buffer and BUF_.BOUND specifies the record length minus 1. The user typically adjusts the BOUND field before writing each variable-length record to the record length minus 1. M$WRITE provides options comparable to the options available on the M$READ service.

## M$PRECORD - POSITION TO RECORD

The M$PRECORD service permits the user to change position within a disk or tape file. Based on the parameters supplied, the monitor positions by key or positions forward or backward by a specified number of records.

Positioning by key applies to all file organizations except consecutive and unit record. To position by key the user specifies these parameters: DCB, KEY, KEYS=YES, and optionally KEYR=YES and N. If KEYR=YES the monitor returns the key of the record found, or if the requested key does not exist the next larger key. If N is also specified, the file is positioned by key and then forward or backward by the number of records specified by N.

The M$PRECORD service positions the file so that the next record read will be the record with the specified key, if a record with the specified key exists. The alternate return is taken with an error reported (E$NOKEY), if a record with the specified key does not exist; the file remains positioned to read the record with the next higher key.

NOTE: For keyed and indexed labeled tape files opened with ACS=SEQUEN, the file is searched in the forward direction. The search is terminated if a key of equal value is found or if end-of-file is reached.

The user may request relative positioning by specifying KEYS=NO and N as the number of records to skip from the current position. The number of records to skip may be a positive number to move forward or a negative number to move backward. At the time the M$PRECORD is called, current position is considered to be (1) the next record to be read if the previous operation was a read, (2) end-of-file if the previous operation was a write for a consecutive disk file or tape file, (3) the next record if the previous operation was a write for other disk files, and (4) the record which would be read next if the previous operation was a call to M$PRECORD or M$PFIL. If the N parameter specifies a value which would cause positioning beyond the limits of the file, the alternate return is taken; the file remains positioned before the first record or after the last record depending on the value of N.

NOTE: If the same FPT is used to position by key at times and relatively by number of records at other times, N must be cleared before positioning by key.

After the relative positioning operation, the number of records actually skipped is returned in the ARS field of the DCB.

## M$REW - REWIND

For labeled and managed tape files, M$REW positions to the beginning-of-file (just the same as M$PFIL with BOF=YES). For free tapes, M$REW rewinds to beginning-of-tape (BOT). If the user calls M$REW with a closed DCB, the DCB is opened automatically using the information currently in the DCB. If no DCB is specified, the serial number parameter specifies which tape volume to rewind. The volume must currently belong to the user and cannot be open.

## M$PFIL - POSITION FILE

The M$PFIL service causes positioning of the medium to the beginning or the end of the current file. For labeled and managed tape files, the position is set before the first record or beyond the last record in the file. Free tapes are positioned beyond the next file mark in either the forward or reverse direction. If the user calls M$PFIL with a closed DCB, the DCB is opened automatically using the information currently in the DCB.

## M$WEOF - WRITE END-OF-FILE

The M$WEOF service is appropriate for only certain devices that require special end-of-file procedures. The M$WEOF service causes a file mark to be written on device (free or managed free) tape. (For managed tape, the current buffer is truncated before the file mark is written.) If the user calls M$WEOF with a closed DCB, the DCB is opened automatically using the information currently in the DCB.

## M$REM - REMOVE OR RELEASE VOLUME

The M$REM service permits a tape volume to be dismounted and optionally permits its respective resource to be released by its volume number. M$REM also rewinds the tape.

There is no DCB associated with an M$REM call. The specified tape volume must belong to the user and cannot currently be open.

## M$TRUNC - TRUNCATE BUFFERS

The M$TRUNC service releases any blocking (POOL) buffers associated with a DCB after completion of any outstanding I/O operations. The M$TRUNC service applies to consecutive, keyed, indexed, relative, and unit record files on labeled tape; labeled tapes in V format; and labeled and managed tapes in D, S, and F formats. For any subsequent read or write operations for the DCB, the system assigns blocking buffers automatically as needed.

# MODULE 6-3

# CP-6 System Tape Processing

This module describes CP-6 system functions involved in the use of magnetic tapes.

This module sometimes refers to tape format information which is detailed in Module 6-8, Tape Formats.

## Automatic Volume Recognition

Before you can use a tape, the tape must undergo the automatic volume recognition (AVR) process. The purpose of AVR is to determine:

1. Tape type - either free tape, unexpired ANS/EBCDIC, or expired ANS/EBCDIC
2. Tape density
3. Volume access limitations
4. Volume owner
5. Volume serial number

The AVR process begins when a tape reel has been successfully loaded and the drive is in a ready state. The AVR routine first determines if the reel has a ring inserted. A rewind, set density, read sequence is then performed to determine whether:

1. the tape can be read at one of the drive's densities
2. the tape can not be read (either the drive can not read at the density the tape was written at, or the first physical record of the tape contains a bad area), or
3. the tape is blank.

Blank and unreadable tapes are considered to be free scratch (unlabeled) tapes.

For tapes successfully read, the contents of the tape's first record are checked to determine if the tape is an ANS volume, a free tape, a managed free tape, or a CP-6 PO tape.

For ANS volumes not created by CP-6, the volume access is ALL. For ANS volumes created by CP-6, the volume access is whatever VOLACCESS the volume creator specified.

As authentication continues, labels are skipped until:

1. an I/O error occurs which causes the volume to be considered a free tape
2. a blank tape, a tape mark, or a label less than 80 bytes long is read which causes the volume to be considered an ANSscratch volume, or
3. an 80-byte HDR1 label is read, which causes the expiration date in the HDR1 label to be compared against the current date to determine if the volume is an expired or unexpired ANS volume.

The AVR routine now rewinds the tape and reports the AVR to tape-concerned operators' consoles and tape file management (TFM), which determines how the volume may be accessed by a user as a function of the volume's serial number and its type:

- either free, unexpired -- for unexpired ANS volumes, or
- expired -- for expired ANS volumes, CP-6 PO tapes, and ANScratch tapes).

The AVR process, referred to as "AVRing a tape", is now complete and the volume is ready to be acquired by you.

## Tape Resource Management

Tape resource management (TRM) is the next stage of tape verification. It is TRM that actually permits a user to access a tape. TRM remembers which volumes are mounted on which drives, which volumes belong to which users, and which volumes are currently in use (i.e. have a DCB open to them). In addition, TRM records tape drive utilization for accounting purposes.

TRM becomes involved when you attempt to open a file on a tape volume. Volumes requests may be divided into two types:

- those requesting a specific volume for input or output
- those requesting "any" volume for output by specifying a blank serial number (this is known as a "scratch" mount when a free tape is desired and an "ANSscratch" mount when an ANS volume is desired).

These differences mainly affect which operator keyins are required for giving a volume to a particular user the first time a user gains access to a volume. Following accesses require no operator intervention since TRM already knows which volumes belong to which users and by what serial numbers the user will refer to them as.

Note that in any type of system the operator must specify the serial number for non-scratch mounts of free tapes.

The LABEL processor may be used to label ANS volumes in fully and semi-protected shops. This processor is described in CE34, Operations Reference Manual, and in CE41, System Support Reference Manual.

When two or more scratch tapes are being used by the same user (different tapes, or tapes with the same name), the user should, on successive opens, fully qualify the resource name which is returned in the DCB after the first (and succeeding) opens (DCB.RES) e.g. by specifying 'MT01' instead of simply 'MT'. This eliminates confusion about which scratch tape is desired.

## CP-6 Tape File Management

CP-6 Tape File Management (TFM) is almost functionally equivalent to Disk File Management to permit programs to work with either a disk file or tape file without special considerations. However, it must be remembered that tapes are definitely not random access devices. Due to the sequential nature of tape, writing a record in the middle of a tape file during an update operation destroys the rest of the file and destroys all following files on that tape volume.

TFM behaves differently according to the level of ANS volume protection selected by the system manager. (See ANS Levels of Protection, Module 6-1.)

Tape files are especially different from disk files when a tape volume set contains more than one volume and has files which cross volume boundaries. TFM will cause proper volumes to be mounted when a file crosses a volume boundaries even when processing involves backing up to a previous volume (by issuing an M$REW or M$PFIL to position to beginning of file or an M$PRECORD to skip backwards). Changing volumes is a time-consuming operation for the operator since reels must be physically mounted. It is time-consuming to the program unless reels have been already pre-mounted by the operator.

A special monitor service, M$CVOL, exists to position a tape file to the beginning of the next section of the file (the part of the file on the next volume). This may be used on input to either skip records in the current file section and go on to the next or to cause the next volume to be mounted if the user has requested volume change (CVOL) control (see M$DCB CVOL). TFM notifies the user of volume end conditions with either of the two abnormal returns E$EOVOL or E$EOVOLS (see Tape Errors module).

E$EOVOL on output means that the M$WRITE failed and must be reissued after the volume change. E$EOVOLS only occurs on input and means that the current record segment is divided across volumes.

M$CVOL may be used on output to cause the next volume to be mounted and the current file section ended. The primary purpose of M$CVOL is to provide user label handling capabilities. It permits the user to read user trailing labels (UTLs) after the current file section, and user header labels (UHLs) preceding the next on input and to write such UTLs and UHLs on output. There is no reason to request CVOL control unless user label processing is desired.

User label processing is the user's responsibility; CP-6 TFM simply makes these labels accessible to the user. TFM verifies that successive volumes indeed contain the proper sections of the tape file. There is no reason for the user to do this. User labels preceeding the file (written or read during M$OPEN) and following the file (written or read during M$CLOSE) are useful only to special applications.

CP-6 TFM insures that file sequence numbers properly ascend as each succeeding file is created.

TFM supplies required ANS labels (see Module 6-8, Tape Formats) and supports the following ANS labeled volumes:

- single-file multi-volume
- single-file single-volume
- multi-file single-volume
- multi-file multi-volume

## Anomalies and Errors

For a complete listing of tape error messages and a discussion of how they are handled see Module 6-9, Tape Errors.

# MODULE 6-4

# How to Make Tapes That Other Machines Can Read

The CP-6 system is capable of producing tapes for use on other systems such as CP-V, or systems which follow ANS standards. This module presents summary information to assist the production of these tapes.

## Tape Types in CP-V and CP-6

The following tables may be used to compare the types of tape formats available on CP-6 and CP-V:

CP-6 TAPE FORMATS

| | |
|---|---|
| FREE | Completely the user's responsibility to retrieve the data, non-standard labels, etc. Files are ORG=FREE. |
| Managed FREE | Like FREE but the records may be blocked. Files are ORG=UNDEF, FIXED, or VARIABLE. |
| ANS | ASCII labels, ASCII data, filenames of 17 characters or less. Files are ORG= UNDEF, FIXED, or VARIABLE. |
| CP-6 Labeled | Superset of ANS which allows filenames up to 31 characters, includes File Information Table (FIT) for each file; files of any CP-6 ORG which includes KEYED and INDEXED files. |
| EBCDIC | Like ANS except labels and data are written in EBCDIC. ORG=FIXED or VARIABLE, EBCDIC=YES. |

CP-V TAPE FORMATS

| | |
|---|---|
| FREE | Completely the user's responsibility to retrieve the data, non-standard labels, etc. |
| ANS | ASCII labels, ASCII data, filenames of 17 characters or less. Files are UNDEF, FIXED, or VARIABLE. Equivalent to 'ANS' on CP-6 as listed above; available only on systems with tape drives which contain the optional ASCII conversion option, such as the X560 CP-V machines. |
| CP-V ANS(IBM) | Like ANS except labels and data are written in EBCDIC. This format is equivalent to 'EBCDIC' on CP-6, as listed above. |

| | |
|---|---|
| CP-V Labeled | EBCDIC labels, EBCDIC data, filenames up to 31 characters, files of any CP-V supported ORG. This format is similar in content to 'CP-6 labeled' listed above, but differs greatly in how the information is laid out. |

## Creating Tapes

The following are some examples of creating ANS or IBM/CP-V tapes for use on systems other than CP-6 (i.e., CP-V). The pertinent SET and PCL options needed to create these tapes are described. Note: The CP-V of IBM/CP-V indicates a CP-V ANS tape, and NOT a CP-V labeled tape.

Examples:

The following examples are with options fully abbreviated. (These all create EBCDIC files.)

- To create a blocked file with variable length logical records:

```
PCL: COPY srcefid TO LT#reelno/ansfid(OR=V,EB,NS)
SET: SET dcb LT#reelno/ansfid,OR=V,EB(CN),SPAN=NO
```

- To create a blocked file with fixed length records:

```
PCL: COPY srcefid TO LT#reelno/ansfid(OR=F,BL=4096,EB, RECL=80)
SET: SET dcb LT#reelno/ansfid,OR=FI,BLK=4096,EB(CN), REC=80
```

- To create an UNDEFined format file:

```
PCL: COPY srcefid TO LT#reelno/ansfid(OR=UN,RECL=80,EB)
SET: SET dcb LT#reelno/ansfid,OR=UN,REC=80,EB(CN)
```

NOTES:

1. Before using ANS tapes for transporting files containing non-textual data (i.e., binary, packed decimal, etc.), to a non CP-6 machine, keep in mind the conversion will probably NOT be what you want and you may have to use some other means for transporting.

2. If files capable of non-sequential access (i.e., KEYED, INDEXED, etc.) are copied to ANS tape, the files are copied as if they were CONSECutive files, and any keys or similar information is lost.

3. If files with variable length records are blocked and the destination machine is CP-V, be advised that PCL on the CP-V machine will probably need patch #32847 in order to successfully read the CP-6 ANS tape. The alternative is:

   - on CP-6, create blocked fixed length records (ORG=FI,...) (see the second example above)

   - then, on CP-V, specify the LN option on the output fid such as:

```
COPY AT#ANS/ansfid TO outfid(LN)
```

   This will create an EDIT keyed file on CP-V. If you just wanted variable length records (not EDIT keyed), then:

```
COPY outfid OVER outfid(NLN)
```

## ANS Tape Options at a Glance

```
    PCL OPTIONS                  SET OPTIONS
    -----------                  -----------

    OR[GANIZATION] =             OR[G] =
       { F[IXED]     }              { F[IXED      }
       { UN[DEF]     }              { UN[DEF]     }
       { V[ARIABLE] }               { V[ARIABLE] }
    BL[OCK] = block-size         BLK[L] = block-size
    DE[NSITY] = tape-density     DE[NSITY] = tape-density
    EB[CDIC]                     EB[CDIC] {[=YES] | =NO}
    NCN[VERT]                    [(CN[VRT] {[=YES] | =NO})]
    R[ECLENGTH] = rec-length     REC[L] = rec-length
    NS[PANNED]                   SPAN[NED] {[=YES] | =NO}
    NB[LOCKED]                   BLO[CKED] {[=YES] | =NO}
```

Note: In PCL, there are no counterparts to CNVRT=YES and SPANNED=YES because PCL defaults to CONVERT and SPANNED.

## ANS Tape Options - Complete Description

ORG[ANIZATION]

F[IXED] Records are fixed length with no control information

UN[DEFINED] Records are an undefined length with no control information

V[ARIABLE] Variable length records

BL[OCK] / BLK[L]

Specifies the maximum physical tape record size, in bytes (1-32764), that will be written to the ANS tape. If this option is omitted, ORGs of FIXED and VARIABLE have a default of 4096 via a SET command, and 2048 in PCL. This option is not applicable to files with ORGANIZATION of UNDEF.

DE[NSITY]

Specifies the density (800,1600 or 6250 BPI) at which to write the tape.

NOTE: This option is not REQUIRED to create an ANS tape, but if not specified, care must be taken to insure that a tape written at the default density can be read on the target machine.

EB[CDIC],NCN[VERT] / EB[CDIC] ... (CN[VRT] ... )

Controls the translation (ASCII to EBCDIC) of tape file labels and data.

In PCL: EBCDIC specifies that the tape labels are to be written in EBCDIC; data will also be written in EBCDIC unless the NCNVERT option is included.

In SET: EBCDIC option specifies whether to write tape labels in EBCDIC (EBCDIC) or ASCII (EBCDIC=NO). The data will be written in ASCII unless (CNVRT) follows the EBCDIC options.

RECL[ENGTH] / REC[L]

In PCL: For files with FIXED organization, RECLENGTH specifies the maximum record length, in bytes. For files with VARIABLE organization, RECLENGTH specifies the maximum record length, in bytes. If this option is omitted or 0 is specified, 80 is used.

In SET: For files with FIXED organization, RECL specifies the actual record length. For files with VARIABLE organization, RECL specifies the maximum record length including the four byte count field.

NS[PANNED] / SPAN[NED] {[=YES] | =NO}
Specifies whether logical records are to be spanned between physical records (blocks). This option is only applicable to ORG=VARIABLE tape files. ORG=UNDEF or FIXED files can not be spanned. CP-6 ORG tape files are always spanned (except ORG=RANDOM or IDS). It distinguishes between ANS formats D and S and EBCDIC (IBM) formats B, S, and R.

## Block Sizes for ANS Tapes

The CP-6 system writes multiples of four bytes when creating blocks for ANS tapes. This is in accordance with the ANS standard, which states that all data blocks and labels may be padded out to a multiple of the word length. This padding is included in the block length, but does not affect reading the file.

Example:

A user creates an ANS labeled tape using PCL with the following command:

```
COPY fid TO LT#sn/fid(ORG=F,BL=1330,RECL=133,NSPANNED)
```

However, the system changes the block size to 1332 because the I/O system writes multiples of four bytes.

# MODULE 6-5

# Converting Imported Tapes For CP-6 Use

## Introduction

This module contains suggestions concerning the use of tapes made on other systems for use on the CP-6 system. The information is accurate as of the date of publication of this manual; however the user should be aware that the accuracy of commands and specifications pertaining to systems other than the CP-6 system cannot be guaranteed.

## How to Make ANS Tapes on CP-V for CP-6 Use

Options allowed when creating an ANS tape with CP-V PCL that the CP-6 system can read are:

FMT(U) - Unblocked - One record per block
FMT(F) - Fixed-length records, blocked
REC(n) - Record size. The value n specifies the size of records for FMT(F) only, where 1 <= n <= 32767 bytes. Records will be truncated or padded to conform, but padding with blanks will extend only 140 bytes. The BLK value must be a multiple of the record size (n). The default for n is 128 bytes.
BLK(n) - Block size. The value n specifies the maximum block size to be built for FMT(F), where 1 <= n <= 32767 bytes. The default is 2048 and if n is less than 18, 18 will be used.

Examples:

To create a blocked file with fixed length records:

```
COPY srcefid TO AT#reelno/ansfid(FMT(F))
```

This would use the defaults: REC(128),BLK(2048)

To create an unblocked file:

```
COPY srcefid TO AT#reelno/ansfid
```

Transporting either an EDIT keyed file with an initial key of 1.000 and increments of 1.000, or a CONSECutive file with variable length records can be done by doing the following:

On CP-V System:
---

```
COPY srcefid TO AT#reelno/ansfid(BLK(b),REC(n))
```

where b is a multiple of n, and n is >= the longest record

On CP-6 System:

If the file was EDIT keyed on CP-V system:

```
COPY LT#reelno/ansfid TO outfid(LN,NB)
```

If not:

```
COPY LT#reelno/ansfid TO outfid(NB)
```

Notes:

1. Before using ANS tapes for transporting files containing non-textual data (ie. binary, packed decimal, etc.), keep in mind the conversion will probably NOT be what you want and you may have to use some other means for transporting.

2. If files capable of non-sequential access (ie. keyed, random, etc.) are copied to ANS tape, the files are copied as CONSECutive files and any keys or similar information will be lost.

## Making ANS Tapes on Multics for CP-6 Use

The following Multics command copies one file to ANS label tape for transport to the CP-6 system. Key in the command exactly as shown except for the variable information indicated by CAPITAL LETTERS.

```
cpf -ids "record_stream_ -target vfile_ SEGID" -ods "tape_ansi_
    TAPESN -nm FILENM -cr -nb NO -retain none -mode ascii"
```

Note that the above is a single command. Substitute appropriate values for SEGID, TAPESN, FILENM and NO as follows:

SEGID is the Multics segid of the file to be copied to the tape (the file name for the CP-6 system).

TAPESN is the tape serial number to be used (1-6 characters).

FILENM is the name to be used on the tape. Enter it in ALL CAPS and restrict it to letters and numbers; also, it should be no more than 17 characters long for ANS compatibility.

NO is the file sequence number on the tape. Use 1 for the first file, 2 for the second, and so on.

The above command can be placed in an EC file, substituting &1, &2, &3, and &4 for SEGID, TAPESN, FILENM, and NO, respectively. This will simplify the file copy operation if it is used repeatedly.

## ANS Tapes Made On Other Systems

Generally speaking, ANS tapes made on other systems should be readable on the CP-6 system, as the purpose of ANS standards is to enable such transfer of information. These other-system ANS tapes may be written in either ASCII or EBCDIC; the CP-6 automatic volume recognition (AVR) process determines the format of the tape and ANS EBCDIC tapes are converted to ASCII when read by the CP-6 system.

For such tapes, the CP-6 COPYALL command can be used:

```
COPYALL LT#GNORF TO .MYACCT
```

The CP-6 system will automatically recognize the "EBCDICness" of the tape and perform the necessary conversions when reading the files.

ANS tapes made on other systems may have ANS names which require quotes when used in the CP-6 system. For example:

```
!COPY LT#46424/DKS.DK090HEC TO FILE
```

will not work since the ANS name includes a period (.). The name must be enclosed in quotes:

```
!COPY LT#46424/'DKS.DK090HEC' TO FILE
```

The above command is accepted by the CP-6 system.

# MODULE 6-6

# How to Copy Tapes

## Making Tape Copies

The PCL COPY and COPYALL command can be used to make copies of tapes. The COPY command copies selected files, and the COPYALL command (which can be used only for labeled tapes) copies multiple files (or complete tapes).

### Making Copies of Labeled Tapes

CP-6 labeled tapes and ANS tapes may be copied with the COPYALL command. For example:

```
<COPYALL LT#source TO LT#dest
```

results in a copy of the source tape. To make a copy of selected files, the COPY command can be used:

```
<COPY LT#source/A - H TO LT#dest
```

This results in all the files whose names are in the alphabetic range from A to H inclusive being copied. The order on output will be the same as on input (i.e., they will not be sorted by name.)

The PHYSICAL PCL input option can be used to copy a consecutive range of files. For example:

```
<C LT#source/B>H(PHYS) TO SQUISH
```

tells PCL to copy only those files consecutively located between B and H (including B and H).

A labeled tape cannot be copied as a free tape unless the FMSEC privilege is used.

## Copying Managed Free Tapes

It is possible for a user to copy selected files from one managed free tape to another tape. If the requirement exists to copy a large number of files on a managed free tape, or to copy an entire tape, it is possible to SCAN the tape to determine how many files need to be moved, build an XEQ file that says 'COPY FT#source TO FT#dest', once for each file to be moved (after an initial !PCL) and run the XEQ file. File sequence number (FSN) may also be used to select files for copying.

At the end of any free tape operation (with the exception of SPR) the tape is positioned past the end of the file. Thus, after you copy one file, the tape is positioned after your file and you are at the beginning of your next file.

## Copying Free Tapes

A free tape may be copied as one large file by using the DEOD option; however, if there are intervening file marks, these will be stripped out by PCL. If the free tape has been structured to include individual files, these can be selected for copying using the PCL COPY command in the same manner that files or a range of files are selected for copying managed tapes.

The SCAN command can be used to examine the free tape and select the file or files desired. Files may be selected by filename or FSN.

A COPY from FT to FT with no options specified will apply all the tape defaults.

When reading tapes made on other systems, a knowledge of the maximum record size is useful in determining the initial buffer size required. In the absence of this information, some experimentation is necessary, as CP-6 hardware does not return a lost data error.

## Extension of Tape Files

In order to extend a tape file, the file must be opened FUN=UPDATE and positioned to its end (M$PRECORD BOF=NO) before any WRITEs take place.

## ASCII TO EBCDIC Conversion

The EBCDIC option works with any ORG except ORG=V.

Example:

A user enters the following commands:

```
!PCL
PCL B03 here
<COPY FID TO FT#1823(EB,OR=V,NS,RECL=3000)
<END
```

in the expectation of creating an EBCDIC tape. However, the tape is created in ASCII.

In order for the conversion to work, the tape file organization must not be variable. If ORG=V is selected, the EDCDIC option is ignored.

## Making a Tape Copy for Export

It is possible to create a copy of a tape for export by using PCL:

Example:

```
!ORES MT(1600BPI)=1
!M Please get reel number 12 (LT#XFER) from library.
!PCL
PCL B03 here
<MOUNT LT#XFER RING,REEL=12
<COPYALL fid OVER LT#XFER(DE=1600,ORG=V,NFA,NBIN)
<REM LT#XFER
<END
!M Please send reel 12 (LT#XFER) to 'No.24, Royce Bldg.'
```

## Copying Binary Data to Tape

Copying binary data to tape is easy to do, even though the CP-6 system must do some special work in order to place the ninth bit on the tape. Simply use PCL to say:

```
!COPY file TO LT#sn/file
```

Do NOT specify ORG on the destination tape, as this will tell PCL to create a transportable ANS tape format which only records eight bits out of nine. The ORG of the disk file tells PCL how to write the tape. Don't bother to specify RECL either, as it is meaningless for a CP-6 tape. You may, however, specify a BLKL to make PCL write longer tape records. See the discussion of block size below.

## Copying ANS Tapes Made on Other Systems

See Module 6-5 under ANS Tapes Made on Other Systems.

## Block Size

The following considerations enter into specifying block size when copying tapes.

### Blocked Tapes and FPOOL Buffers

When copying a tape with large blocks, the size of the file management buffers (FPOOLS) as set by the IBEX LIMIT command will have an appreciable effect upon the time required to perform the copy.

For example, consider the following command:

```
COPYALL LT#0023/?.CP91ACC OVER LT#2110/?.CP91ACC (BLOCK=28672)
```

The block size specified requires a seven-page file buffer (one page = 4096 bytes) for the input tape, and another for the output tape. This block size will require at least 7 FPOOL pages to work at all, and will require at least 14 FPOOL pages to operate efficiently.

To the number of buffers mentioned above, the user must add those buffers required to perform miscellaneous functions (two or three).

The default number of buffers supplied by the CP-6 system will be sufficient in the performance of ordinary tasks, and need be given special consideration only when large blocks are specified.

### Block Size Versus Efficiency

Increasing block size results in a gain in tape efficiency (that is, more data on less tape) but also increases the probability of error. The marginal gain in tape efficiency using 7k blocks over 4k or 5k should be balanced against this increased probability; the tradeoff may not necessarily be a good buy.

Tape efficiency may be expressed as:

$$\text{efficiency} = \frac{\text{block size (inches)}}{\text{block size (inches)} + \text{gap}}$$

where

$$\text{block size (inches)} = \frac{\text{block size (bytes)}}{\text{BPI}}$$

and

$$\text{gap} = 1/2 \text{ inch}$$

The following table illustrates results calculated for both 1600 BPI and 6250 BPI comparing block sizes of 4K and 7K.

| Density | Efficiency @4K(16kb) | Efficiency @7K(28kb) |
|---|---|---|
| 1600 BPI | .95 | .97 |
| 6250 BPI | .84 | .90 |

# MODULE 6-7

# Multi-reel Tapes

## Creating Multi-volume Tape Sets

Consider the following situation: You want to create a multi-volume tape set. To do so, you issue the PCL command

```
!COPY ANYFILE TO LT#SN1#SN2#SN3/ANYFILE
```

You don't know how many reels the file will occupy, so the tape mounted could be any of the three serial numbers. How do you remove or rewind the tape?

The answer is to issue the PCL command

```
!REM LT#SN1#SN2#SN3
```

or

```
!REW LT#SN1#SN2#SN3
```

PCL remembers reels that are used together. So if you use a volume set called LT#SN1#SN2#SN3, PCL remembers that the three are related and you are positioned on (for example) #SN2. Then when you say "REW LT#SN1#SN2#SN3", PCL removes #SN2 and sets things up so that the next use of that set gets #SN1.

User programs and other processors do not have access to PCL's memory of what is going on, so they won't handle tapes in the same way. If you use, for example, a COBOL program to create a multi-volume tape set and then use PCL to rewind it, PCL tries to do the "right thing": It tries to rewind all the reels and complains for each one that isn't mounted. Thereafter, PCL remembers which reels belong to that set and that the set is positioned to the first reel.

## Using Volume Sets

A valid labeled tape fid is LT#s1#s2#s3. PCL treats this as a volume set or 'unit' rather than as separate tapes. Hence

```
REW LT#R1#R2#R3
```

(if R2 is the current one mounted) unloads R2, asks for R1 and positions it at BOT (even if the tape set was left in this state as the result of, for example, output from an executing COBOL program).

```
REW LT#R1,LT#R2,LT#R3
```

attempts to perform a rewind on three different single-reel tape sets and requests that the ones not mounted be mounted.

```
REM LT#R1#R2#R3
```

attempts to remove all reels of the set.

For a three-reel set which ends on reel two,

```
SPE LT#R1#R2#R3
```

leaves the tape positioned in #R2 -- it puts the file at current position on the tape set (on #R2 where the SPE left it).

Once PCL treats #R1#R2#R3 as a volume set, if you then use the partial tape set #R2#R3, you may cause some difficulty. PCL will now remember that there is a set called #R2#R3 and store any subsequent volume transitions under that name. Later use of the full set would (once again) put you on #R1.

The first time #R1#R2 is used PCL starts with the first volume, #R1. If the set was used before, the volume last used with that set is opened. Note if either of the volumes is manipulated by itself (with or without PCL), you will encounter difficulties.

There is no reason why volumes cannot be pre-mounted or even used before being used as a set as long as you make certain that they're at BOT. Note, however, that there is no way for PCL to tell anything about which reels are mounted, as PCL uses a private system of remembering what's going on without use of monitor support.

In order to read a tape in PCL that was set up by another processor, VOL=n should be used to tell PCL where the last processor left the tape. It is possible to say VOL=n on output.

Problem:

A program outputs to LT#S1#S2#S3 and leaves it positioned at end of file (which may be on any of the reels). You then want to rewind the tape and then use it as input to something else (perhaps PCL). How do you know what 'n' to use on VOL=n?

If you say:

```
!REW LT#S1#S2#S3
!IF STEPCC>1 THEN ABORT
```

is PCL going to give an error message about non-mounted reels and set the STEPCC?

The answer is that yes, PCL will complain about non-mounted reels in this example. However, the STEPCC is set to a low number (one) to indicate that this was only a warning message.

Does PCL 'remember' the volume set only for the particular PCL invocation, or can a series of commands given to IBEX be processed correctly? The answer is that the memory lasts for the the entire session. Therefore, invocations from IBEX and departures from PCL are possible.

You cannot get COBOL to position to BOT on #S1 when closing the output file, since COBOL doesn't report volume transitions. However, you can say

```
!ACCEPT ANNOUNCE
```

This will tell you when you hit a MOUNT request. However this does not apply when running in batch.

# MODULE 6-8

# Tape Formats

## CP-6 Tape Formats

The purpose of this module is to describe and explain the various tape formats and tape file organizations, to enable a user to make an informed choice when determining optimum parameters for creating tape files.

The formats described in this module reflect standards of the American National Standards Institute (ANSI). Further information about American National Standards (ANS) tape labels and file structures is available in the American National Standard for Magnetic Tape Labels and File Structure for Information Interchange (ANSI X3.27-1978), which defines a tape labeling technique to enable the transfer of character data between two dissimilar operating systems.

Tape formats used in the CP-6 system are an extension of the ANSI standard to permit backup of non-character CP-6 disk files. However, both CP-6 format and ANS format tape files have the same label structure; they only differ in data record representations and the inclusion of a File Information Table (FIT) with data records for CP-6 format tape files.

The ANS labeling method uses 80-byte labels and tape marks (records distinguishable from data records) in a way which allows a number of files of different formats to be stored on a single tape volume. The primary ANS labels, named after the contents of their first four bytes, are:

Primary ANS Tape Labels

| LABEL | MEANING |
|---|---|
| VOL1 | First label on tape; contains volume serial number and volume owner information. |
| HDR1 | Indicates the beginning of a file "section" (files which cross volumes are composed of separate sections); contains basic file identification information such as file name, creation and expiration dates, and section number. |
| HDR2 | Follows HDR1 label and continues file information with data record structure information. |
| EOV1 | Indicates the end of a file section; implies that the file continues on the next volume of the set. |
| EOV2 | Mirrors HDR2 which began the file section except contains the block count for the file section. |
| EOF1 | Indicates the end of a file section and file. |
| EOF2 | Mirrors HDR2 which began the file section except contains the block count. |

## User Labels

In addition to these required labels which are supplied by CP-6 Tape File Management (TFM), the ANS standards make provisions for user labels to supplement HDR label information. The first three characters of each user label are determined by the standard; the fourth and remaining characters are supplied by the user. User labels are described in the following table:

User Labels

| LABEL | MEANING |
|---|---|
| UHL | User Header Label immediately follows HDR labels. |
| UTL | User Trailer Label immediately follows EOV or EOF labels. |

## Additional ANS Labels

The ANS standard provides for other labels:

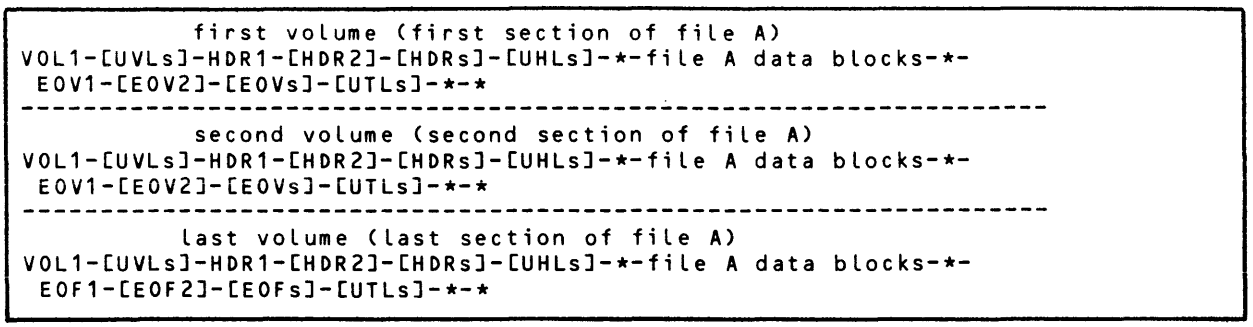
Additional ANS Labels

| LABEL | MEANING |
|---|---|
| HDR3 through HDR9 | Used in addition to HDR1 and HDR2. |
| EOV3 through EOV9 | Used in addition to EOV1 and EOV2. |
| EOF3 through EOF9 | Used in addition to EOF1 and EOF2. |
| UVL1 through UVL9 | User volume labels; immediately follow the VOL1 label. |

A tape volume containing this last group of labels can not be created by CP-6 TFM, but if these labels are present, they will be ignored. Options on the M$OPEN, M$CVOL, and M$CLOSE service calls permit the creation and acquisition of UHL and UTL label groups.

## ANS Labeled Structure

The overall structure of a single-file multi-volume ANS labeled volume set is diagrammed in the following figure. Note that CP-6 TFM also supports single-file single-volume, multi-file single-volume, and multi-file, multi-volume volume sets.




Tape Structure

In the above figure, the square brackets denote optional labels. The asterisks represent tape marks. Dashes represent inter-record gaps. The structure of data blocks depends on the format of the tape file. The double tape mark (represented by *-*) indicates end of volume. HDR2, EOF2, and EOV2 labels are always present for CP-6 created tape files. If a HDR2 is not present on an input volume created by another system, file format is assumed to be UNDEF.

For more detailed information about ANS tape labels and file structure, see the ANSI standard referenced at the beginning of this module.

# MODULE 6-9
# TAPE ERRORS

## Introduction

This module describes the kinds of errors that may be encountered by the user in performing tasks using magnetic tape. A summary table of these errors in alphabetical order is included at the end of the module.

While a wide variety of errors are possible, most mean basically the same thing: your tape can not be trusted for reliable information. Luckily, most serious tape errors are unlikely to occur, but if one does, it should be handled by reporting the error and giving up.

### I/O Errors

The I/O system retries all errors a number of times. When an I/O error is returned to the user it means that at the current point on this tape, nothing can be read or written properly. I/O errors are shown in the following table:

I/O Errors

| Error Message | Meaning/Comment |
|---|---|
| E$IOERR | Can't read or write tape. Tape file management at this point "locks" an ANS volume (but not a free tape) against all further I/O attempts. Subsequent I/O attempts will cause E$CANTIO error. |
| E$CANTIO | No I/O is currently allowed. The volume is unlocked when it is rewound (M$REW or M$CLOSE PTV) which enables tape file management to verify the actual position. |
| E$SMALLRECL | Record is too small to be written; records less than four bytes long, which are considered to be noise records, are not permitted to be written. |

| Error Message | Meaning/Comment |
|---|---|
| E$POSERR | Position error; tape position has been lost. This occurs when a supposedly ANS labeled volume is discovered to have non-standard or unrecognizable labels where standard labels are expected. After this error has been received on an open tape file, the only reasonable action is to close the file. No more I/O is permitted until position is found; attempts to do I/O will result in E$CANTIO. The user can restore position in one of two ways, either by rewinding the tape (as for free tape) or by attempting to open the next "findable" file on the tape with an M$OPEN specifying NXTF (next file) and FINDPOS (find position). Note that a number of files may be skipped trying to find the next "openable" file and that it is possible, if the tape is near the end of the reel, for the tape to run off the end of the reel. |

## Free Tape Errors

Errors which are peculiar to free tape are:

Free Tape Errors

| Error | Meaning/Comment |
|---|---|
| E$EOT | End of tape; detected during writes; record successfully written, though. |
| E$BOT | Beginning of tape; detected during M$PFIL and M$PRECORD |
| E$BLNKTP | Blank tape; detected during read. |
| E$BOF | Beginning of file; tape mark hit during M$PRECORD. (Can also occur for ANS tape.) |
| E$EOF | End of file; tape mark hit during M$READ or M$PRECORD. (Can also occur for ANS tape.) |

Free Tape Errors (cont.)

| Error | Meaning/Comment |
|---|---|
| E$OPER | Fatal device error; tape drive went out of ready. |

## Volume End Errors

A special service, M$CVOL, exists to position a tape file to the beginning of the next section of the file (the part of the file on the next volume). This may be used on input to either skip records in the current file section and go on to the next or to cause the next volume to be mounted if the user has requested volume change (CVOL) control (see M$DCB CVOL). The user is notified of volume end conditions with either of the two abnormal returns:

Volume End Errors

| Error | Meaning/Comments |
|---|---|
| E$EOVOL | End of volume. E$EOVOL on output means that the M$WRITE failed and must be reissued after the volume change. |
| E$EOVOLS | End of volume but current record crosses volumes. E$EOVOLS only occurs on input and means that the current record segment is divided across volumes. |

## Volume Change Errors

Another condition occurs during volume change when the subsequent volume can not be mounted for some reason or the volume set is malformed. The DCB may stay open even though there is no tape volume associated if the previous volume has been already dismounted. I/O attempts will result in abnormal returns. The DCB must be closed by the user. The operation which caused the error to be detected will result in one of the following errors:

Volume Change Error Messages

| Error | Meaning/Comments |
|---|---|
| E$SECTERR | The next or previous file section is missing. |
| E$VOLOUT | No next volume to CVOL to. |
| E$VOLORDER | Next or previous volume is missing or misplaced. The serial number list (volume set) is incomplete or out of order. |

## Data Record Structure Errors

Errors in which non-standard or conflicting data record structure are detected during a read are reported to the user. Usually record segments can be salvaged under such conditions; thus, CP-6 TFM returns partial record segments even though the abnormal return is taken. This class of errors includes:

Data Record Structure Errors

| Error | Meaning/Comments |
|---|---|
| E$BADBLKL | Actual data block is smaller than indicated in the block. |
| E$BADRECL | Actual record is smaller than indicated in the block. |
| E$BADSPAN | Spanning information is inconsistent. |
| E$PARTIALKEY | Part of the key returned is missing. |
| E$BLKCNT | The system accumulated block count differed from the block count specified in the EOV1 or EOF1 label. The DCB will contain the proper ARS for the segment returned. In some cases the data record structure is sufficiently destroyed to render the data inaccessible. |
| E$NONDECRCW | Record control word (for VARIABLE (D,S) ANS format file records) is nondecimal. This error can not be passed by; subsequent attempts to read the same record will result in the same error. |

## Break Error Messages

Since operator intervention is an important part of system tape handling, some operations require that a program must be temporarily suspended. Hitting break or control-Y during such moments aborts the current operation. Abnormal returns result in:

Break Error Messages

| Error | Message/Comments |
|---|---|
| E$MNTBRK | Control-Y or break while waiting for volume to be mounted. |
| E$TAPBRK | Control-Y or break while waiting for RING/OVER keyin, or during file search. This message is generated by hitting break or control-Y after the volume has been mounted but during a period when Tape File Management is searching for a requested file. |

If any of these errors occur during a tape M$OPEN, it should probably be reported and considered fatal. It is possible to "recover" from these errors by re-executing the M$OPEN.

## Operator-generated Errors

The CP-6 operator has the ability to deny access to a tape for a number of reasons. These result in errors:

Operator-generated Errors

| Error | Meaning/Comments |
|---|---|
| E$CANT | The operator can't mount the specified volume. |
| E$OPROT | The operator is protecting the tape by not issuing a RING keyin and not putting a ring in the reel. |

## Errors and Protection Level

The following errors are related to the level of ANS volume protection.

Protection Level Errors

| Error | Meaning/Comments |
|---|---|
| E$OPROT | The operator is protecting the tape by not issuing an OVER keyin. |
| E$UNEXPIRED | That volume is unexpired. This error is peculiar to a fully-protected shop as the result of attempting to write on an unexpired volume. |
| E$NOTANS | Mounted volume is not an ANS volume. This error occurs in a semi-protected shop on an attempt to open an ANS tape DCB to a free tape for output, or in any shop on an attempt attempt to open an ANS tape DCB to a free tape for input. |
| E$NOTDEV | Mounted volume is not a free (device) tape; occurs in a semi-protected shop on an attempt to open a free tape DCB to an ANS tape for output. |

## Access Limitation Errors

Errors which occur due to the volume owner specifying an access limitation are:

Access Limitation Errors

| Error | Meaning/Comment |
|---|---|
| E$NOTOWNR | Access to this volume is limited to its owner. |
| E$NOTOWNRW | Write access to this volume is limited to its owner. |

## Tape Type and Tape Format Errors

Errors concerned with tape type and tape format are:

Tape Type/Format Errors

| Error | Meaning/Comments |
|---|---|
| E$DENSBAD | Density specified on M$OPEN is unavailable on the acquired tape drive. The user may specify a volume density change when opening the first file of a volume set for output. Density changes elsewhere in the volume set are not permitted. |
| E$ASCIITAP | An EBCDIC file may not be created on an ASCII tape. The user may specify a change from an EBCDIC volume to an ASCII volume or vice versa only when creating the first file of the volume set. |
| E$EBCDICTAP | An ASCII file may not be created on an EBCDIC tape. |
| E$EOSET | End of volume set. The FSN of the last file of a volume set is returned when an open results in the E$EOSET error. |

## Miscellaneous Errors

Remaining errors are concerned with protocol involved in acquiring volumes, creating files, and specifying volume sets:

Miscellaneous Tape Error Messages

| Error | Meaning/Comments |
|---|---|
| E$BRDUPSN | There is a blank or duplicate serial number imbedded in the specified serial number list (blank serial numbers may occur at the end of the list or be indicated by specifying a non-zero MAXVOL (see M$DCB). |
| E$NOSN | No serial number or blank serial number is specified for an input volume. |
| E$NOFIDTHIS | No file name or file sequence number is specified for this input tape file. |
| E$BADFSN | The volume set contains 9999 files which is the maximum for an ANS labeled volume set. A new file can not be created. |
| E$FSNERR | The current volume contains adjacent files with file sequence numbers that are either out of order or not in sequence. |
| E$SCRORDER | An ANSscratch or free tape which is not the first volume of a set is specified as the volume to create a labeled file on; since its file sequence number can not be determined, this is not allowed. |
| E$VOLERR | The serial number list does not contain the volume specified by DCB.VOL |
| E$FUNNYPOS | Due to an error in a previous operation, this volume was left with a funny position; it must be rewound to be accessed again. |

Miscellaneous Tape Error Messages (cont.)

| Error | Meaning/Comments |
|---|---|
| E$PARTIALSN | The serial number list returned during an M$CLOSE operation is incomplete because the user's buffer is not large enough. |

There are, of course, a number of errors which are not limited to tape files, but also occur with disk files. Errors of this type are in CE40, Programmer Reference manual.

## Tape Error Message Summary

The following table presents a summary of tape error messages in alphabetical order for easy reference.

Tape Error Message Summary

| Error Message | Meaning |
|---|---|
| E$BADBLKL | Actual data block is smaller than indicated in the block. |
| E$BADFSN | The volume set contains 9999 files which is the maximum for an ANS labeled volume set. |
| E$BADRECL | Actual record is smaller than indicated in the block. |
| E$BADSPAN | Spanning information is inconsistent. |
| E$BLKCNT | The system accumulated block count differed from the block count specified in the EOV1 OR EOF1 LABEL. |
| E$BLNKTP | Blank tape; detected during read. |
| E$BOF | Beginning of file; tape mark hit during M$PRECORD. |
| E$BOT | Beginning of tape; detected during M$PFIL and M$PRECORD. |
| E$BRDUPSN | There is a blank or duplicate serial number imbedded in the specified serial number list (blank serial numbers may occur at the end of the list or be indicated by specifying a non-zero MAXVOL (see M$DCB). |
| E$CANT | The operator can't mount the specified volume. |
| E$CANTIO | No I/O is currently allowed. |
| E$DENSBAD | Specified density is unavailable on the acquired tape drive. |
| E$EBCDICTAP | An ASCII file may not be created on an EBCDIC tape. |
| E$EOF | End of file; tape mark hit during M$READ or M$PRECORD. |
| E$EOSET | End of volume set. |

| Error Message | Meaning |
|---|---|
| E$EOT | End of tape detected during writes although record successfully written. |
| E$EOVOL | End of volume. |
| E$EOVOLS | End of volume but current record crosses volumes. |
| E$FSNERR | The current volume contains adjacent files with file sequence numbers that are either out of order or not in sequence. |
| E$FUNNYPOS | Due to an error in a previous operation, this volume was left with a funny position; it must be rewound to be accessed again. |
| E$IOERR | Tape data can not be read or written. |
| E$MNTBRK | Control-Y or break while waiting for volume to be mounted. |
| E$NOFIDTHIS | No file name or file sequence number is specified for this input tape file. |
| E$NONDECRCW | Record control word (for VARIABLE (D,S) ANS format file records) is non decimal. |
| E$NOSN | No serial number or blank serial number is specified for an input volume. |
| E$NOTANS | Mounted volume is not an ANS volume. |
| E$NOTDEV | Mounted volume is not a free (device) tape. |
| E$NOTOWNR | Access to this volume is limited to its owner. |
| E$NOTOWNRW | Write access to this volume is limited to its owner. |
| E$OPER | Fatal device error; tape drive went out of ready. |
| E$OPROT | The operator is protecting the tape by not issuing a RING keyin and not putting a ring in the reel. |
| E$OPROT | The operator is protecting the tape by not issuing an OVER keyin. |
| E$PARTIALKEY | Part of the key returned is missing. |
| E$PARTIALSN | The serial number list returned during an M$CLOSE operation is incomplete because the user's buffer is not large enough. |
| E$POSERR | Position error; tape position has been lost. |
| E$SCRORDER | An ANSscratch or free tape which is not the first volume of a set is specified as the volume to create a labeled file on; since its file sequence number can not be determined, this is not allowed. |
| E$SECTERR | The next or previous file section is missing. |
| E$SMALLRECL | Record is too small to be written. |

Tape Error Message Summary (cont.)

| Error Message | Meaning |
|---|---|
| E$TAPBRK | Control-Y or break while waiting for RING/OVER keyin, or during file search. |
| E$UNEXPIRED | That volume is unexpired. |
| E$VOLERR | The serial number list does not contain the volume specified by DCB.VOL. |
| E$VOLORDER | Next or previous volume is missing or misplaced; the serial number list (volume set) is incomplete or out of order). |
| E$VOLOUT | No next volume to CVOL to. |

# Index

Note: Index references indicate the page on which the paragraph containing the index term actually ends. Should the paragraph straddle two pages, the actual indexed term might be on the first page, while the index reference is to the second page.

## A


Abbreviating Account References Through Wildcarding - 53
ACCESS - 50
Access Limitation Errors - 175
account access - 49 53
account references - 53
accounts, file - 54
ACQUIRE - 66
Acquiring Tape Drives - 120
ACSVEHICLES - 50
Additional ANS Labels - 166
Allocating Resources and Establishing Service Limits - 66
Anomalies and Errors - 148
ANS Labeled Format - 118
ANS Labeled Structure - 166
ANS labels - 164 166
ANS Levels of Protection - 116
ANS Tape Fids - 119
ANS Tape Options - Complete Description - 151
ANS Tape Options at a Glance - 151
ANS Tapes Made On Other Systems - 155
ANS unprotected mode - 127
ASCII - 3
ASCII TO EBCDIC Conversion - 158
attributes - 43
  file - 43
  file access - 49
  FUN - 49
  SHARE - 49
Automatic Volume Recognition - 145

## B

backspacing - 13
backup file - 46
BASIC - 85
Basic Types of Magnetic Tapes - 115
BATCH - 6 68 69
Batch accounting report - 87
Batch Jobs - 67
BATCH/XEQ Substitution - 75
Block Size - 159
Block Size Versus Efficiency - 159
Block Sizes for ANS Tapes - 152
Blocked Tapes and FPOOL Buffers - 159
Break Error Messages - 173
BREF option (LINK) - 102
BUILD command (EDIT) - 29 48

## C

CANCEL - 68
CHECK and NOTIFY - Comments - 7
CHECK command - 6 7
command file (EDIT) - 40
Command File Logic - Conditional Execution - 71
Command File That Interrogates User - 77
Command File to Read Tape - 76
Command Files - 69
  EDIT - 31 40
  IBEX - 69 76
command labels (IBEX) - 71
Comparisons - Free and Managed Free - 118
compilations in batch mode - 109
conditional execution -
  EDIT - 39
  IBEX - 71
consecutive files - 47 48
Conserving Disk Space - 57
CONTROL H - Backspacing - 13
CONTROL R - Forward Positioning - 13
copies, lineprinter - 60
COPY command (PCL) - 60
COPY command -
  EDIT - 35
  LEMUR - 81
  PCL - 51
COPY ME command - 30 48
Copying ANS Tapes Made on Other Systems - 159
Copying Binary Data to Tape - 158
Copying Free Tapes - 157
Copying Managed Free Tapes - 157
copying on the lineprinter - 60
CP-6 Tape File Management - 147
CP-6 Tape Formats - 164
CP-V - 149 153
Creating Multi-volume Tape Sets - 161
Creating Tapes - 150

## D


Data Record Structure Errors - 171
Data Replacement - 69
data replacement parameters - 69
DCBs - 128
debugging FPL - 88
DEFAULT - 69
Default Tape Drive Assignments - 121
Defining New Escape Sequences - 21
disk - 49
disk space - 57 58
DISPLAY PROFILE command - 9
DISPLAY USER command - 79
Displaying DRIBBLE Files at Your Terminal - 27
DISPLAYS - 6
DONT ACCEPT command - 79
dribble files - 26


## E


EBCDIC Labeled Format - 119
EDIT - 31
EDIT Command Files - 40
EDIT COPY command - 47
EDIT -
  command file - 31 40
  conditional execution - 39
  selection criteria - 32 39
Errors and Protection Level - 174
errors, tape - 168
ESCAPE <CR> - 18
ESCAPE <RET> - Position to Beginning of Record - 14
ESCAPE A - 3
ESCAPE A - Setting Pagination Mode - 16
ESCAPE D - Retrieving the Last Input Line - 17
ESCAPE I - 13
ESCAPE J - 18 19
ESCAPE J, ESCAPE <CR>, ESCAPE O, ESCAPE M - 18
ESCAPE K - Deleting From Current Edit Point - 16
ESCAPE M - 18
ESCAPE N - 11
ESCAPE O - 18
ESCAPE R - Retyping the Current Input Line - 17
escape sequences - 11 15 18
  defining new - 21
ESCAPE V - 19
ESCAPE V - Moving to Character 'N' - 12
ESCAPE X - Deleting Current Input Line - 15
ETMF - 7
Examples of IBEX Command Files - 76
Examples of Online Use - 123
Execute Files - 67
Executing Programs - 64
EXPRESSION COMPONENT - 75
Expression Component - Precedence of IBEX Operators - 74
expressions, IBEX - 73
extended attributes (file) - 44
Extension of Tape Files - 157

## F


fid - 49
file access attributes - 49
file attributes - 43
File Management Buffers - 128
file organization - 51
file recovery - 46
File Sequence Numbers - 125
File Types - 54
file, XEQ - 67
files - 29 35 51
  access attributes - 49
  accounts - 54
  attributes - 43
  backup - 46
  consecutive - 47 48
  copying - 35
  dribble - 26
  EDIT command - 31
  extended attributes - 44
  in other accounts - 59
  indexed - 47
  keyed - 47 48
  keyed data - 58
  listing and reviewing - 43
  merging - 35 48
  organization - 47
  recovery - 46
  reviewing - 43
  scratch - 56
  selecting - 51
  star - 35 56
  types - 54
formats, tape - 164
Forms Program - 92
forward positioning - 13
FPL - 88
FPOOLS - 128
Free and Managed Free Tapes - 117
Free Tape Errors - 169
Free Tapes - 117
FUN attribute - 49


## G


GLOBAL - 69


## H


Hardware Limitations - 127
HELP - 2 11 111
How to Make ANS Tapes on CP-V for CP-6 Use - 153


## I


I/O Errors - 168
IBEX Command Labels - 71
IBEX Expressions - 73
IBEX Programming Conventions - 63
IBEX -
  command files - 76
  operators (precedence) - 74
  preprocessor - 75

programming - 63
SET command - 49
IF command (EDIT) - 39
IMP - 21
IMP command file - 25
indexed files - 47
indexes - 106
input manipulation processor (IMP) - 21
inserting characters - 18
inserting/replacing/overstriking - 18
Interrupt Processing - 65
Introduction - 129 153 168
Invoking Language Processors - 65

## J

JOB - 67

## K

keyboard, redefining - 21
Keyed Data Files - 58
keyed files - 47 48

## L

labels (IBEX) - 71
labels -
ANS - 164
ANS additional - 166
ANS user - 165
LDEV command (IBEX) - 60
LEMUR processor - 81
LIMIT - 66
linefeed function - 32
lineprinter - 60
LINK -
ambiguities in references - 97
ancestor node - 97
BREF option - 102
descendent node - 97
HELP - 104
nodes - 97
overlay programs - 95
overlays - 95
program trees - 97
PROMOTE options - 103
root node - 97
sharing overlayed programs - 97
tree structure - 98
LIST (IBEX) - 43
LIST command (IBEX) - 44
LIST command (LEMUR) - 81
LIST command (PCL) - 44
listing and reviewing files - 43
Listing File Attributes - 43
Logical Density - 122
Lost Data - 127
lost records restored - 46

## M


M$CLOSE - CLOSE DCB - 140
M$CVOL - CLOSE VOLUME - 141
M$DCB - 136
M$OPEN - OPEN DCB - 139
M$PFIL - POSITION FILE - 143
M$PRECORD - POSITION TO RECORD - 142
M$READ - READ RECORD - 142
M$REM - REMOVE OR RELEASE VOLUME - 144
M$REW - REWIND - 143
M$TRUNC - TRUNCATE BUFFERS - 144
M$WEOF - WRITE END-OF-FILE - 144
M$WRITE - WRITE RECORD - 142
mag tape -
  access limitation errors - 175
  acquiring drives - 120
  acquiring resources - 121
  ANS labeled structure - 166
  ANS labels - 164
  ANS levels of protection - 116
  ANS options - 151 151
  ANS user labels - 165
  ASCII to EBCDIC conversion - 158
  automatic volume recognition (AVR) - 145
  basic types - 115
  block size - 152 159
  block size versus efficiency - 159
  blocked tapes and FPOOL buffers - 159
  break error messages - 173
  copying binary da ta to tape - 158
  CP-6 formats - 149
  CP-V - 149
  CP-V formats - 149
  data record structure errors - 171
  DCBs - 128
  default drives - 121
  drives - 120
  EBCDIC - 119
  error message summary - 177
  errors - 168
  extension of tape files - 157
  file management buffers - 128
  file sequence numbers (FSNs) - 125
  formats - 164
  FPOOL buffers - 159
  FPOOLS - 128
  free - 117
  free tape errors - 169
  hardware limitations - 127
  I/O errors - 168
  labeled tapes - 118
  labels - ANS - 164
  labels - ANS additional - 166
  labels - ANS user - 165
  logical density - 122
  lost data - 127
  managed - 117
  miscellaneous errors - 176
  mixed-density - 122
  monitor service calls - 135
  multi-reel - 120
  multi-volume tape sets - 161
  number of bytes stored - 127
  operator-generated errors - 173
  PCL commands - 114
  PCL MOUNT command - 121

protection level errors - 174
record size - 127
resources - 121
single-density - 122
tape copies - 156
tape fids - 119
tape file management (TFM) - 147
tape format errors - 175
tape resource management (TRM) - 146
tape sets - 161
tape type errors - 175
tapes for other machines - 149
volume change errors - 171
volume end errors - 170
volume sets - 162
Making a Tape Copy for Export - 158
Making ANS Tapes on Multics for CP-6 Use - 154
Making Copies of Labeled Tapes - 156
Making Tape Copies - 156
Managed Free Tapes - 117
MD(Move/Delete) command - 37
MERGE command - 36
merging files - 35 48
Minimum Record Size - 127
Miscellaneous Errors - 176
Mixing of CP-6, ANS, and EBCDIC Labeled - 119
Monitor Service Calls - 135
More About ESCAPE J - 19
MOUNT Command - 121
Moving the Cursor - 11
Multi-reel Tape Fids - 120
multi-volume tape sets - 161
MULTICS - 154

## N

NODEBUG option - 58
NOTIFY command - 7
Number of Bytes Stored on Tape - 127

## O

Object Units - 57
!OFF FULL report - 87
Operator-generated Errors - 173
ORESOURCE - 66
Other Files - 58
overlay programs - 95
overstriking characters - 18

## P

pagination mode - 16
password - 51
PCL - 36
PCL (and star files) - 56
PCL (wildcarding) - 52
PCL and !SET Output Options - 130
PCL commands - 48 114
PCL Input Options - 132
PCL output options - 48 133
PCL processor - 51
PCL Tape Control Commands - 129
PCL vs SET - 135

Peak memory - 87
platen control - 20
Preprocessing of Commands - 75
preprocessing of commands (IBEX) - 75
Preprocessor Substitution - 75
preprocessor substitution (IBEX) - 75
printing files - 60
PRIORITY - 68
PROFILE command - 9
Program Exit Method - 72
program tree structure - 98
Program Trees - 97
programs, source - 57
PROMOTE options (LINK) - 103

## R

range source component - 51
READ - 50
READ command (EDIT) - 40
record ranges (EDIT) - 32
Recovering Files - 46 46
Redefining the Keyboard - 24
RELEASE - 66
replacing characters - 18
RESET command - 49
Resetting Special Function Keys - 23
Resolving Differences and Ambiguities - 97
RESOURCE - 66
Resource memory required - 87
RESOURCE, ORESOURCE, and ACQUIRE Commands - 121
RESPONSE - 7
retrieving the last input line - 17
REVIEW Command (PCL) - 45
RN (renumber) command - 38
RUN command - 83 85
Run Units - 58
Running Out Of Space - 57

## S

sample CP-6 session - 79
scratch files - 56
SE range (EDIT) - 32
selecting files - 51
selection criteria (EDIT) - 39
selection range (EDIT) - 32
service limits - 66
SET command - 49 80
SET command options - 49
!SET Command Options - 130
Setting Up Special Function Keys - 22
SETUP command - 6 9 9
setup file - 6 78 78
SHARE and FUN attributes - 49
sharing users - 49
Single-density and Mixed-density Systems - 122
SORT processor - 86 106

sorted indexes - 106
Source Programs - 57
special function keys - 22
Specifying an Overlay - 98
star files - 35 56
STATUS command - 7
STEPCC - 72
string selection (EDIT) - 39
substitutions (IBEX) - 75
sysid - 6 7

## T

Tab Stops - 12
tape drives - 120
  default - 121
Tape Error Message Summary - 177
tape errors - 168
Tape Fids - 119
Tape Fids May Need Quotes - 120
Tape File Management (TFM) - 147
tape formats - 164
Tape Management - 114
tape options - 151
Tape Resource Management - 146
tape sets, multi-volume - 161
Tape Type and Tape Format Errors - 175
Tape Types in CP-V and CP-6 - 149
Terminal Profiles - 9
TEXT - 86
TFM (tape file management) - 147
tree structure (program) - 98
TRM (tape resource management) - 146
TRUTH VALUE operator - 74

## U

UNDER DELTA command - 80
underlining - 18
User Labels - 165
Using Files in Other Accounts - 59
Using HELP - 104
Using LINK to Build a Run Unit with Overlays - 102
Using LINK's PROMOTE_BLANK and PROMOTE_LABEL Options - 103
Using Volume Sets - 162

## V

Volume Change Errors - 171
Volume End Errors - 170
volume sets - 162

## W

wildcarding - 52
WRITE - 50

## X

XEQ command - 67 69
XEQ files - 67 69
XMIT - 69

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: CP-6 APPLICATION PROGRAMMER HANDBOOK

ORDER NO. CE55-01

DATED JANUARY 1984

**ERRORS IN PUBLICATION**

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

---

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**
**200 SMITH STREET**
**WALTHAM, MA 02154**

**ATTN: PUBLICATIONS, MS486**

---

**Honeywell**

**Together, we can find the answers.**

# Honeywell

**Honeywell Information Systems**
**U.S.A.**: 200 Smith St., MS 486, Waltham, MA 02154
**Canada**: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
**U.K.**: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
**Mexico**: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho Chiyoda-ku, Tokyo
**Australia**: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

39518, 5C184, Printed in U.S.A.

CE55-01