**Xerox Data Systems**

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

# Xerox Universal Time-Sharing System (UTS)

## Sigma 6/7/9 Computers

## Time-Sharing

## User's Guide

FIRST EDITION

90 16 92A

April 1971

Price: $3.75



Printed in U.S.A.

# NOTICE

This document applies to the A00 (initial) release of UTS.

# RELATED PUBLICATIONS

| Title | Publication No. |
|---|---|
| Xerox Sigma 6 Computer/Reference Manual | 90 17 13 |
| Xerox Sigma 7 Computer/Reference Manual | 90 09 50 |
| Xerox Sigma 9 Computer/Reference Manual | 90 17 33 |
| Xerox Universal Time-Sharing System (UTS)/TS Reference Manual | 90 09 07 |
| Xerox Universal Time-Sharing System (UTS)/SM Reference Manual | 90 16 74 |
| Xerox Universal Time-Sharing System (UTS)/OPS Reference Manual | 90 16 75 |
| Xerox Batch Processing Monitor (BPM)/BP, RT Reference Manual | 90 09 54 |
| Xerox Symbol and Meta-Symbol/LN, OPS Reference Manual | 90 09 52 |
| Xerox Edit (for UTS)/Reference Manual | 90 16 33 |
| Xerox Delta (for UTS)/Reference Manual | 90 16 34 |
| Xerox BASIC/LN, OPS Reference Manual | 90 15 46 |
| Xerox Sort-Merge/Reference Manual | 90 11 99 |
| Xerox Manage/Reference Manual | 90 16 10 |
| Xerox FORTRAN Debug Package (FDP)/Reference Manual | 90 16 77 |
| Xerox Extended FORTRAN IV/LN Reference Manual | 90 09 56 |
| Xerox Extended FORTRAN IV/OPS Reference Manual | 90 11 43 |
| Xerox ANS COBOL/LN Reference Manual | 90 15 00 |
| Xerox 1400 Series Simulator/Reference Manual | 90 15 02 |

Manual Type Codes: BP - batch processing, LN - language, OPS - operations, RBP - remote batch processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities.

# CONTENTS

## APPENDIXES



## FIGURES



## TABLES



## EXAMPLES

# 1. INTRODUCTION

## UNIVERSAL TIME-SHARING SYSTEM

The Universal Time-Sharing System (UTS) is a general purpose time-sharing system that operates on a Sigma 6, 7, or 9 computer, a variety of peripheral devices, and a network of remote terminals linked to the computer by telephone lines. To gain access to the system through a remote terminal, you simply dial the number of the computer on a telephone attached to your terminal. A wide variety of services is provided by the UTS terminal executive and its subsystems after the terminal is connected to the system.

In addition to on-line (i.e., time-sharing) terminal services, UTS provides a full set of batch processing services. Jobs may be submitted to the batch job stream through a card reader at the central site or through an on-line terminal.

Since UTS can serve as many as 64 users simultaneously, the individual user obtains the results of his processing much sooner than he would under a conventional system. Under a normal load, the response to most on-line requests occurs in less than two seconds.

## TERMINAL EXECUTIVE LANGUAGE

The Terminal Executive Language (TEL) is the on-line command language for UTS, a concise natural language for performing on-line functions and calling on-line subsystems. It also provides information services, such as accounting charges and status of available system resources.

Functions performed directly by TEL commands include

- Building a file.
- Initiating a processor.
- Loading and executing a program.
- Quitting or continuing an interrupted processor.
- Copying a file.
- Deleting a file.
- Controlling output.
- Setting DCB assignments.
- Submitting batch jobs.
- Checking the status of batch jobs.
- Saving and restoring files.
- Queuing output for symbiont devices.
- Setting tab stops for terminal I/O.
- Controlling the terminal interface (e.g., page width and length).
- Setting the log-on password.
- Communicating with the operator.

On-line subsystems and facilities available through TEL include

| | |
|---|---|
| FORT4 | Extended version of FORTRAN IV. |
| META | Assembler with powerful procedure (macro) capability: Meta-Symbol. |
| BASIC | Subsystem for creating, executing, and maintaining programs written in a simple mathematical language. |
| EDIT | Line/text editor. |
| PCL | Language for copying and deleting files, listing directories, and manipulating tapes. |

| | |
|---|---|
| DELTA | Debugging subsystem used primarily for assembly-language programs. |
| FDP | Debugging package for FORTRAN programs. |
| LINK | Subsystem that constructs an executable program (load module) from object-program modules. |
| BATCH | Subsystem that submits a batch job file to the batch job stream. |
| CONTROL | Subsystem used to display performance measurements and change critical system parameters (authorized users only). |
| SUPER | Subsystem used to determine who may log on and with what privileges and limits (authorized users only). |

Subsystems are usually called explicitly by name but may also be called implicitly by the following TEL commands:

| | |
|---|---|
| BUILD | Calls Edit to build a file. |
| COPY | Uses PCL to copy files. |
| DELETE | Uses PCL to delete files. |
| RUN | Uses LINK to link a program and causes the program to be loaded and executed. |

## SCOPE OF THIS MANUAL

This manual is designed as a simple guide for using UTS in time-sharing mode only. It is not intended as guide to "sophisticated" usage, nor as a complete reference to TEL and other commands. Please refer to Xerox UTS/TS Reference Manual, Publication 90 09 07 and applicable language reference manuals for complete command forms and descriptions. However, Appendix A of this manual presents a summary of TEL commands in reference format.

The command formats shown in the text are not necessarily complete, as for example in the case of PCL COPY. Only the more commonly used forms are given and explained. Also, knowledge of at least one of the programming languages available under UTS is required for full understanding of this manual.

## NOTATION CONVENTIONS USED IN THIS MANUAL

The following conventions are observed throughout this manual:

1. All characters typed by the system are shown underlined.

2. Special-purpose terminal control keys (Figure 1) used in examples are shown circled. These keys are as follows:

   (ESC) Escape

   (BRK) Break

   (RUB) Rubout

   (RET) Carriage return

   (LF) Line feed (may normally be used instead of carriage return)

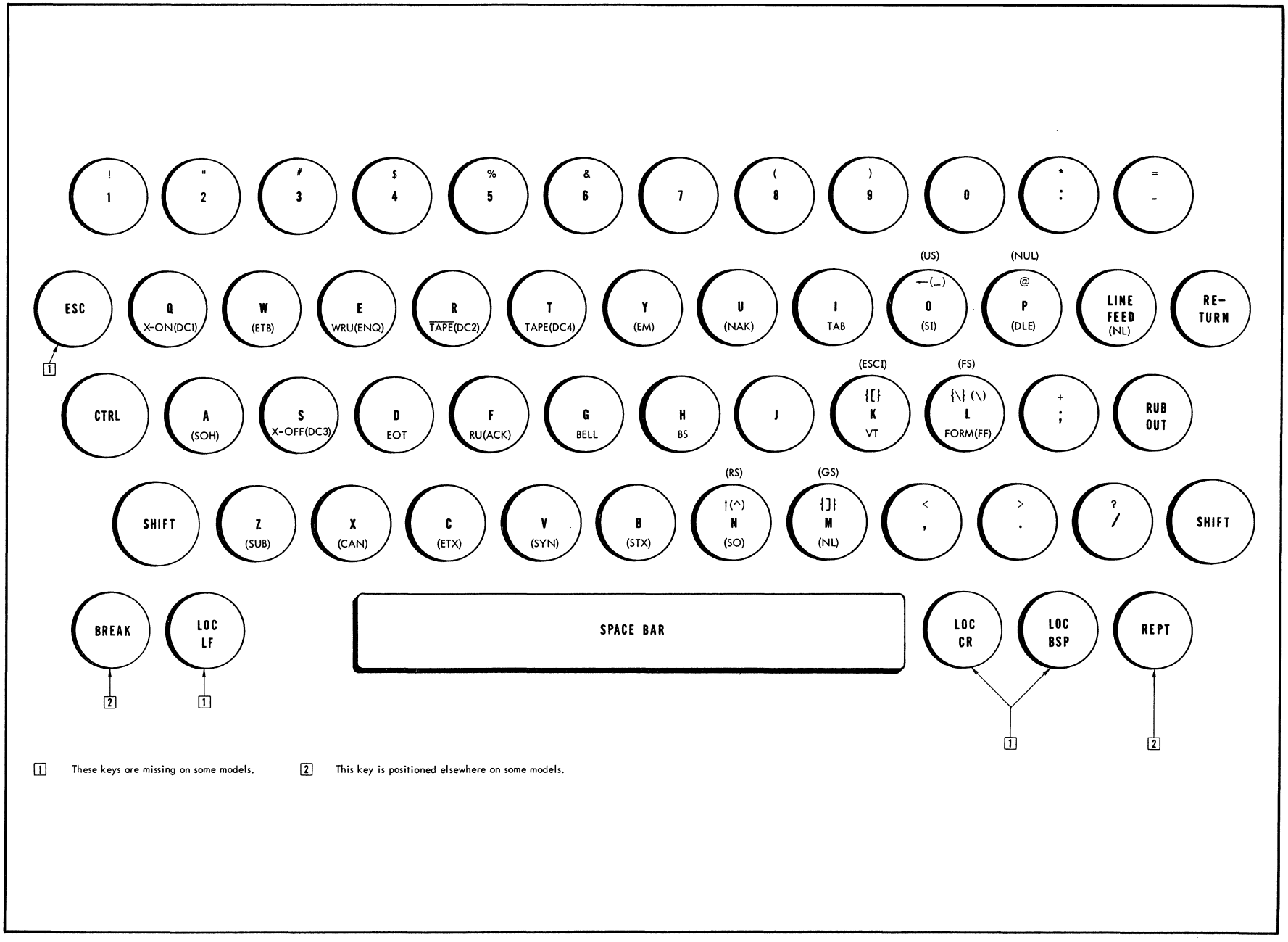
3. Combinations of keys depressed simultaneously are indicated as follows:

   (TAB) Notation for $I^C$ or (ESC)I, the "tab character".

   $\alpha^C$ Some alphabetic key (symbolized by $\alpha$) and the CTRL (control) key pressed simultaneously (e.g., $X^C$).

   Note that the encircled control keys normally do not result in the printing of any character except for Rubout (\) and $X^C$ (← or _).

4. In command formats, square brackets are used to indicate optional parameters, and braces are used to indicate a required choice. (These characters are not an actual part of the command. Also, lowercase letters are employed to indicate where in a command to substitute a name, symbol, numerical value, etc. For example, EDIT file-name indicates that the name of a file may be specified along with the verb EDIT (separated by one or more blanks).




1 These keys are missing on some models.

2 This key is positioned elsewhere on some models.

Figure 1. UTS Terminal Keyboard

# 2. LOGGING ON AND OFF

## DIALING THE COMPUTER

To establish connection with the computer, proceed as follows:

1. Turn on power switch for terminal and for acoustical coupler (or "modem"), as necessary.
2. Pick up telephone handset, wait for dial tone, and dial computer. A high-pitched tone will be heard if a communication line is available.
3. Place handset on acoustic coupler (see Figure 2).

TEL (Terminal Executive Language) now responds with the following message:

UTS AT YOUR SERVICE

ON AT (time and date)

LOGON PLEASE:

You can now log onto UTS (provided you have been enrolled on the system by the system manager).

## LOGGING ON AND OFF

To log on, you must have an account number, a user-ID, and possibly a password. The account is your billing number and the user-ID is your personal or group identification. Both are assigned by the system manager. Password is an account-protection feature that is assigned either by the system manager or by yourself (see PASSWORD Command, below). It can be modified periodically for security purposes.

Upon receipt of the message LOGON PLEASE:, enter your account, ID, and password, in that order, separated by commas. The password and preceding comma are omitted if no password was assigned.

Figure 2. Typical Dialing Unit

For terminals operated in full-duplex mode, character echoing by the system is normally on but can be turned off (e.g., to suppress printing of passwords or other security-related information) by striking the (ESC)E keys. Striking the (ESC)E keys a second time turns echoing back on. For terminal units operated in half-duplex mode, character echoing by the system must be turned off, as above, to suppress duplicate printing of characters.

It may not always be possible to log on. If an error prevents the reading of the logon file, the message UNRECOVERABLE I/O ON RAD, or ABNORMAL ERROR ON LOGON FILE will be typed. Whenever you are unable to log on, start over by striking the BREAK key and trying again. The system tries five times to log you on before dismissing you.

If a MAILBOX file (a message file) exists at log-on time, the message CHECK DC/MAILBOX will appear. You may examine this MAILBOX file by copying it to your terminal as follows:

```
! COPY MAILBOX TO ME
```

(The underscored exclamation mark is the "prompt character" issued by TEL.)

The allowable characters for ID's, accounts, and passwords are

A-Z a-z 0-9 ← $ * % : # @ -

The graphic representation of certain special characters, such as the left arrow, is terminal-device dependent, as is the availability of the lowercase alphabetics. The character set shown above should be regarded as representative only in this respect.

Account-number and password may each be from one to eight characters in length. The user-ID may consist of one to 12 characters.

Example 1. Logging On and Off.

| | |
|---|---|
| UTS AT YOUR SERVICE | The user dials the computer. |
| ON AT 12:30 MAR 12, '71 | The system identifies itself, states the time and date, and requests that the user log on. |
| LOGON PLEASE: 2232,HALL (RET) | In response, the user types in his account number (2232) and user ID (HALL). He does not use a password because the system manager has not assigned him one. |
| 12:30 03/12/71 2232 HALL 15-9[1] | A page heading is printed by the system; the items of information in the heading are, in order: time, date, account number, user-ID, two internal identifiers, and page number (enclosed in square brackets). |
| !OFF (RET) | The Terminal Executive types its prompt character (!) indicating that the system is ready to process a TEL command. Since this was just an experiment for the user, he logs off. |
| CPU = .0124 CON = :01 INT = 2 CHG = 10 | Summary of accounting information for session. |

The user has used .0124 minutes of Central Processor Time (CPU = .0124); he has been connected to the terminal (from dialing up to end of accounting summary) .01 hour (CON = :01); he has interacted with the system twice (INT = 2), logging on and the OFF command. His charge is e.g., 10 charge units, an installation-dependent value.

Example 2. Logging On with a Wrong Account Number

```
UTS AT YOUR SERVICE                          The user dials the computer.
ON AT 02:30 MAR 12, '71

LOGON PLEASE: 223L,HALL (RET)                He types in the right ID but the wrong account number.

ACCOUNT/ID 223L/HALL ?                       The system questions the incorrect account number, and
                                             asks the user to log on again, which he does.
LOGON PLEASE: 2232,HALL (RET)

-page heading-

!OFF (RET)                                   He then logs off.

-accounting summary-
```

Example 3. Inability to Log On Due to Error in Logon File

```
UTS AT YOUR SERVICE
ON AT 12:42 MAR 17, '71
LOGON PLEASE: C37-105,HALL (RET)
      The user tries to log on.

ABNORMAL ERROR ON LOGON FILE
SORRY,UNABLE TO LOG YOU ON
      The system cannot log him on and so informs him.

CPU = .0024 CON = :01 INT = 1 CHG = 10
      The accounting summary is presented.

UTS AT YOUR SERVICE
      The system repeats its logon sequence.

ON AT 12:43 MAR 17, '71
LOGON PLEASE: C37-105,HALL (RET)
      This time the user's logon is accepted.
-page heading-
!OFF (RET)
      He now logs off.
CPU = .0024 CON = :01 INT = 2 CHG = 15
```

## PASSWORD COMMAND

The purpose of the logon password is to protect your resources and files by preventing illicit use of your ID and account number. The PASSWORD command allows you to change your password frequently to make it difficult for anyone else to know what it is. You can also use the command to cancel your password if you wish.

It is important to remember your password because only the system manager is able to recover it for you if you do not remember it.

Example 4. Setting a Password

```
UTS AT YOUR SERVICE
ON AT 12:49 MAR 17, '71
LOGON PLEASE: 2232,HALL (RET)
```

The user logs on, with no password set.

```
!PASSWORD SECRET (RET)
PASSWORD CHANGE SUCCESSFUL
```

The Terminal Executive types its prompt character (!) indicating it is ready to process a TEL command. The user sets his password to SECRET and must now use it whenever logging on until he or the system manager changes it.

-page heading-

```
!OFF (RET)
```

The Terminal Executive types its prompt character and the user logs off. Password SECRET remains set.

-accounting summary-

Example 5. Logging on with Password and then Cancelling Password

```
UTS AT YOUR SERVICE
ON AT 14:45 MAR 17, '71
LOGON PLEASE:  2232,HALL (RET)
```

The user logs on but forgets to use his new password.

```
PASSWORD ?
```

The system indicates that the password was not entered.

```
LOGON PLEASE: 2232,HALL,SECERT (RET)
```

The user logs on with an incorrect password.

```
PASSWORD SECERT ?
```

The system indicates that the password is invalid.

```
LOGON PLEASE: 2232,HALL,SECRET (RET)
```

The user now logs on with the correct password.

-page heading-

```
!PASSWORD (RET)
```

He cancels his password by typing the PASSWORD command and specifying no password.

```
PASSWORD CHANGE SUCCESSFUL
!OFF (RET)
```

He then logs off. Next time he logs on, no password will be required.

-accounting summary-

Example 6. Setting a Password and Suppressing Its Printing

```
UTS AT YOUR SERVICE
ON AT 09:05 MAR 20, '71
LOGON PLEASE: 2232,HALL (RET)
```

The user logs on.

```
-page heading-

!PASSWORD (ESC) E (ESC) E (RET)
```

The Terminal Executive types its prompt character (!) indicating it is ready to process a TEL command. The user sets his password but suppresses its printing by typing (ESC) E before the password (the first E is not actually echoed), then turns echoing on again. He must now use the password he has just set whenever logging on, until he or the system manager changes it. Any sequence of 1-8 permissable characters may be used as a password.

```
PASSWORD CHANGE SUCCESSFUL
!OFF (RET)
```

The Terminal Executive types its prompt character indicating it is again ready for a TEL command. The user logs off. The next time he logs on, he must use the password just set.

```
-accounting summary-
```

# 3. TERMINAL INTERFACE

## INTRODUCTION

This chapter describes methods for correcting, modifying, and deleting terminal input and the use of the TERMINAL, PLATEN, and TABS commands.

## EDITING OF TERMINAL INPUT

A line of terminal input may be corrected, modified, or deleted, before the line is released to the system (with RET). This may be done by way of either character or line deletion:

1. Editing by Character Deletion: On detecting a typing error within a few characters of the point of error, you may delete the last characters typed by typing a corresponding number of rubout RUB characters (echoed with a \ character), and continuing the line from the (deleted) point of error. (Any n successive RUB characters effectively delete the n successive characters immediately preceding the first RUB character.)

2. Editing by Line Deletion: To delete a complete line of input — before giving a carriage return, simultaneously depress the CTRL and X keys ($X^c$ in conventional notation). The system echoes $X^c$ with a ← (left arrow), effectively deletes the line, and gives a carriage-return/line-feed. (The previous prompt character, if any, is not repeated.) The input can then be repeated in correct form.

These editing features apply to any untransmitted line of terminal input, under TEL or any other subsystem except Delta.

Example 7. Making Corrections to TEL Commands

```
UTS AT YOUR SERVICE
ON AT 15:30 MAR 22,'71
LOGON PLEASE: 2232,HALK\L RET
```

While logging on, the user hits a K instead of an L. To delete K, he strikes the rubout key which echoes back to the terminal as a backslash. Then he types L and completes the logon sequence. (Note that the characters printed at the terminal are those echoed back to the terminal and are not necessarily the same ones typed, as for example \ for RUB.)

```
- page heading -

!QASSWORD Y07←
```

The user then types in a password command but notices an error (password misspelled) before striking the carriage return key. Instead he depresses CONTROL and X simultaneously, which the system echoes back as a left arrow (or possibly an underline). This causes the line to be cancelled and a carriage return.

```
PASTWO\\\SWORD Y07 RET

PASSWORD CHANGE SUCCESSFUL
```

The user notices still another error. This time he deletes three characters and then completes the command successfully. Note that prompt character (!) is not repeated.

```
!OFF RET
```

He then logs off.

```
- accounting summary -
```

## TERMINAL COMMAND

The TERMINAL command is used to inform the system of the type of terminal used, and is required only if the terminal differs from a type of terminal unit specified as standard by the system. (This information can be obtained from the installation manager.)

Format:

TERMINAL tc

where tc is a two-character alphanumeric terminal code:

33 for Teletype®† Model 33

35 for Teletype®† Model 35

37 for Teletype®† Model 37

(Additional terminal codes will become applicable as more types of terminal units are added to the system's capabilities.)

Example 8. Use of TERMINAL Command

```
UTS AT YOUR SERVICE
ON AT 11:45 MAR 23,'71
LOGON PLEASE: 2232,HALL,YO7

- page heading -

!TERMINAL 37 (RET)

      Indicates a Model 37 Teletype®†. The system will use this information to modify response to input/
      output for different types of terminals, as necessary. For the rest of the session, the Monitor recog-
      nizes the terminal as a Model 37 Teletype®†.

      .
      .
      .

!OFF (RET)

- accounting summary -
```

## PLATEN COMMAND

A page width of 72 characters and a printable page size of 54 lines are normally used for all terminal input and output. The PLATEN command can be used to change the page width and/or page length. If an output line is longer than the effective page width (as in Example 9), the system breaks the line by inserting carriage-return/line-feed characters. Note: The minimum page width is 12 characters and the minimum page length is 12 lines.

The PLATEN command can also be used to suppress page headings by specifying a value of less than 12 for the page length, effectively giving a page unlimited length. If a value of less than 12 is specified for the page width, the page width is treated by the system as unlimited in size. That is, the width of a printed or displayed line is limited only by the physical constraints of the device on which the line is produced (up to a maximum of 140 characters).

Example 9 shows how you can use PLATEN to change page width. This example contains four job steps, i.e., major functions during a session that cause the invoking of processors such as EDIT, PCL, or META.

---

† ® Registered trademark of the Teletype Corporation.

Example 9. Using PLATEN Command to Change Page Width

```
UTS AT YOUR SERVICE
ON AT 17:24 APR 15,'71
LOGON PLEASE: 14777,N.U.USER (RET)

17:24 04/15/71 14777 N.U.USER      23-7[1]

!BUILD TEST1 (RET)

      1.000 12345678901234567890123456789012345678901234567890 (RET)
      2.000 (RET)
```

The user enters the BUILD command to build file TEST1. File building is described in detail in Chapter 4. (This is the first job step of the session.)

```
!PLATEN 20 (RET)
```

The page width is set to 20.

```
!COPY TEST1 ON ME (RET)
12345678901234567890
12345678901234567890
```

File TEST1 is printed at the terminal. The page width is now 20. (The COPY command, which implicitly invokes the PCL subsystem, is the second job step.)

```
!PLATEN 39 (RET)
```

The page width is set to 39.

```
!COPY TEST1 ON ME (RET)
123456789012345678901234567890123456789
0
```

File TEST1 is printed again but with 39 characters per line. (This COPY constitutes a third job step.)

```
!PLATEN 12 (RET)
```

The page width is set to 12.

```
!COPY TEST1 ON ME (RET)
123456789012
345678901234
567890123456
7890
```

File TEST1 is printed again but now has 12 characters per line. (This is the fourth and last job step.)

```
!OFF (RET)
```

The user logs off. Note that the last PLATEN command is still in effect.

```
CPU = .0097
CON= :03 INT
 = 13 CHG =
42
```

Example 10 shows how PLATEN is used to change page length. (This example also contains four job steps.) Note that the page-length specification refers to the number of single-spaced lines in the body of the page, i.e., excluding top-of-page heading and spacing. Each line of double-spaced output, where double spacing occurs, counts as two lines. Therefore if n double-spaced print lines are desired, the page-length must be specified as nx2. Occasionally, only n-1 lines, or less, will be printed due to various circumstances, e.g., an intervening single-spaced command line.

Format:

PLATEN [w][, l]

where

w is an optional integer specifying the page width, in number of characters per line (140 maximum). If the w field is null, the prior width setting is retained.

l is an optional integer specifying the page length, in number of line-positions per page, exclusive of top-of-page heading and space (a total of 12 lines); l may have a maximum value of 256. If the l field is null the prior length setting is retained.

The PLATEN command may be given during a process interrupt, i.e., prior to the issuance of a CONTINUE or QUIT command.

Example 10. Using PLATEN Command to Change Page Length

```
UTS AT YOUR SERVICE
ON AT 17:29 APR 15,'71
LOGON PLEASE: 14777,N.U.USER(RET)

17:30 04/15/71 14777 N.U.USER 21-9[1]

!BUILD TEST2

      1.000 1 (RET)
      2.000 2 (RET)
      3.000 3 (RET)
      4.000 4 (RET)
      5.000 5 (RET)
      6.000 6 (RET)
      7.000 7 (RET)
      8.000 8 (RET)
      9.000 9 (RET)
     10.000 10(RET)
     11.000 (RET)

      The user builds file TEST2.

!PLATEN 72,12 (RET)

      The PLATEN command sets page width to 72 characters, length to 12 lines.

17:30 04/15/71 14777 N.U.USER 21-9[2]

      The system prints the second page heading, on overflow of newly set page length.

!COPY TEST2 ON ME(RET)

1

2
```

```
3

4

5

6
```

A copy of file TEST2 is printed at the terminal. (Note that a file built at the terminal, as in this case, is copied back to the terminal double-spaced. A later example will show how to suppress this double spacing.)

```
17:31 04/15/71 14777 N.U.USER 21-9[3]
```

The third page heading prints.

```
7

8

9

10


!PLATEN ,18 (RET)
```

This PLATEN command causes the current page to be lengthened from 12 to 18 lines. (Page width remains at 72.)

```
!COPY TEST2 ON ME (RET)

1

2

3

17:31 04/15/71 14777 N.U.USER 21-9[4]

4

5

6

7

8

9

10


!OFF (RET)
```

The 18-line page is still in effect.

```
17:31 04/15/71 14777 N.U.USER 21-9[5]

CPU=.0129 CON=:05 INT = 19 CHG = 55
```

**TABS COMMAND**

The TABS command is used to simulate typewriter-like tab stops for terminal input and output. TABS supplies the tab-setting values that are to be used by the system when it encounters a 'tab character' in the input or output line.

You can then tabulate by typing $I^C$ (CONTROL and I) in your input wherever you desire a tab in both the input and corresponding output.

The tab settings can be changed by another TABS command. (Tab simulation can be turned off, and then back on, with the key sequence (ESC) and T.)

Example 11. Using the TABS Command

```
UTS AT YOUR SERVICE
ON AT 17:35 APR 15,'71
LOGON PLEASE: 14777,N.U.USER (RET)


- page heading -


!TABS 8,22,37,45,52 (RET)

      The user sets tab-stop values for terminal input and output.

!BUILD TEST3 (RET)

    1.000 THIS (TAB) EXAMPLE (TAB) ILLUSTRATES (TAB) USE (TAB) OF (TAB) TABS (RET)
    2.000 (RET)

      File TEST3 is built using tabulation ( (TAB) = I^C).

!COPY TEST3 TO ME (RET)
THIS    EXAMPLE    ILLUSTRATES    USE    OF    TABS

      This file is printed with tab simulation on.

! (ESC) T (RET)

      The user now turns off tab simulation with the sequence (ESC) T.

!COPY TEST3 TO ME (RET)
THIS EXAMPLE ILLUSTRATES USE OF TABS

      The file now prints with no tabbing.

!OFF (RET)


- accounting summary -
```

# 4. MANIPULATING FILES

## FILES IN UTS

Almost from the moment you become a user of UTS, you start accumulating data – the information upon which the system must operate to provide the answers to the problems you pose. All of the subsystems mentioned in Chapter 1 produce some kind of data; for example:

- EDIT allows you to create the collection of statements necessary to phrase a problem-solving procedure in the language of an assembler or compiler, called a source program, and to create input data for such programs.

- BASIC, FORTRAN, and META allow you to translate a source program, which is only a model of the external idea, into a form suitable for execution by the machine. This translation produces object code as the result of either a "compilation" or "assembly" process.

- LINK prepares the relocatable object code for machine execution, in the form of a load module.

These different kinds of data have at least one characteristic in common: each must be stored in some retrievable form, both between the steps of an information-processing operation, and between executions of the same operation.

Conventional batch systems provide the user with several ways of storing data, principally on punched cards or magnetic tape. Although these media provide low cost, long-term storage, they require operator intervention at the central computer site when the stored information is to be accessed or updated. This intervention may be merely inconvenient for batch operation when the information is used frequently, but it is generally infeasible for on-line use of a time-sharing system.

UTS file management capabilities provide an alternative and remarkably versatile medium for maintaining your working data – a medium which greatly lessens your dependence on conventional external forms of storage, and increases the flexibility with which the data can be manipulated. File storage in UTS is implemented through use of the XDS Rapid Access Data Storage System (RAD). Generally speaking, a RAD is a non-removable rotating-disk memory device containing approximately 1.5 million words of storage, any portion of which can be accessed within a very short time. (UTS allows you, the on-line user, to access conventional peripheral-storage devices also, if you are so authorized.)

The monitor, through its file-management system, allows information to be stored in RAD and identified symbolically, simply by a file name chosen by the user. The files are segregated by account number (your identifying number assigned by the installation manager). Therefore, you cannot inadvertently generate file names that conflict with those of other users outside your account. Certain other information about the file, such as restrictions on access by other users, is also kept with the file.

Files can be used to store any kind of information. They can contain source-language programs built with Edit or BASIC, translated source programs produced by a compiler (relocatable object code), or object code in executable form (a load module) produced by the link-loader. They can also contain collections of alphanumeric data, and natural-language text.

A file is identified by a name of 1-10 characters constructed from a prescribed set of characters. (Some processors, such as Edit, allow up to 31-character file names for special purposes.) The permissible character set contains all of the alphabetic and numeric characters plus most of the commonly used special symbols. Typically, you will need no more than the alphabetic and numeric characters. In the command-language formats given throughout this manual, the symbol most commonly used to indicate a "file name" is fid, which stands for file-identification. A file-identification actually can include an explicit account number and password, as well as the file name. But in our examples, and in most actual usage, fid is interpreted simply as a file name. Complete rules for the structure of file-identifications are given in Appendix B.

Having created a file of information, you are completely free to access or delete it, replace it, or modify it, through the on-line services of UTS, without any operator intervention. A general rule is that you may not delete or modify files not in your account, though often you may access such files.

Sometimes files are created automatically for you by the system. It is possible to call a processor such as FORTRAN to translate a source program without specifying a file into which to store the object code. In this case, the system creates a unique temporary file, associated with your account, for output storage. You may refer to this file with the single character $ under certain subsystems. The $ file is temporary in the sense that when you log off, the file is automatically released. This is useful when creating test programs where nothing of permanent value is being created as output.

Any file that is explicitly named for output is permanent, i.e., retained in the file-management system across the periods between on-line sessions. All files explicitly created with BUILD and COPY commands are also permanent. Permanent-file content is maintained and updated solely by the user. However, file storage space is a chargeable resource, and it is in your interest to delete unneeded files whenever possible.

When dealing with files throughout UTS, there are two command modifiers of importance: ON and OVER. ON implies that the named file does not yet exist. If such a file does indeed exist and ON is used, an error message is sent to the user. (In general, the word TO may be substituted for ON, with the same effect.)

OVER implies that the file may exist already and, if so, is to be reused for the new operation. Using OVER also results in a completely new version of the file; any old data in the file is lost. If the file does not exist and OVER is specified, no error is noted, and the file is automatically created. There is no limit to the number of operations that may be performed OVER a file.

Information about the immediate intended use of a file is called a file specification. These specifications are made implicitly by the use of several commands, particularly COMMENT, LIST, and OUTPUT. An explicit specification can be made by the use of the SET command. (See Chapter 10, "DCB Assignments".) For our present purposes, file specifications may be considered to indicate that a file is an input or an output file, and if an output file, what type(s) of output the particular file is to receive.

Once a file specification has been made, it remains in effect throughout a terminal session until changed or deleted by another specification – the one exception concerns source-input files (operational label SI), which always default to the user's terminal at each job step. If, for example, listing output is directed to the file "DATA", then all listing output generated by a series of assemblies or compilations are placed on this file, one behind the other. This convention is known as "file extension" and is automatically in effect for output operations on standard system-assigned files – or more precisely, through certain system-created Data Control Blocks (DCBs). (DCBs are described in the BPM/BP, RT Reference Manual, Publication 90 09 54, and discussed further in the TEL chapter of the UTS/TS Reference Manual, Publication 90 09 07.) References to DCBs are gradually introduced further along in this manual, and they are treated specifically in Chapter 10, "DCB Assignments".

File extension is an important feature to keep in mind when operating at the terminal, especially when it is not desirable to stack any output during multiple-job-step operations. File extension is reset to the beginning of the file upon any new specification, even if the specification refers to an already existent file. For example:

| | |
|---|---|
| !LIST ON GRUNCH | Listing output directed to file GRUNCH. |
| !OUTPUT ON RUNFILE | Object-code output directed to file RUNFILE. |
| !META SOURCE | Read input-file and assemble (job step 1). |
| !META ME | Assemble from terminal (job step 2). |
| !META TESTY | Read input-file and assemble (job step 3). |

This sequence of commands results in all output being stacked on their respective files, GRUNCH or RUNFILE.

The new listing-output specification and further job step

| | |
|---|---|
| !LIST OVER GRUNCH | |
| !META | (job step 4) |

has the effect of replacing the old contents of GRUNCH with the new assembly listing as the source input is entered from the user's terminal. The object-code output would still be stacked at the end of file RUNFILE. This basic use of file extension logic applies independent of the manner in which file specifications are made, i.e., through the SET command or through commands implying a specification.

To understand certain error comments you may encounter, you will need some knowledge of file organization. This refers to the way the file's contents, i.e., its individual records, are ordered. Two possible organizations are

- Consecutive, where the records can be accessed in sequential order only.
- Keyed, where the records may be accessed directly (randomly) or sequentially.

Files built under the Edit subsystem, having a line number associated with each record (line), are an example of a kind of keyed file. Most files you will use will probably be keyed, but you may see a system comment stating "...file not keyed...". Certain UTS subsystems are not keyed-file oriented, e.g., BASIC and PCL, though they handle keyed files properly in most cases. Later we indicate what you can do in other cases.

## EDIT SUBSYSTEM

The Edit subsystem is a general-purpose, line-number oriented text editor. It may be used to create or modify source programs, data files, reports, etc. for other UTS subsystems, specifically for FORTRAN, Meta-Symbol, BASIC, and the BATCH subsystem.

Edit provides file editing capability, i.e., the ability to build, delete, copy, or merge files; to edit within a line of a file; and to do a complex editing operation on each line in a specified range of lines.

The examples in this section illustrate how the Edit subsystem is used to perform file editing, and to access a file and perform record (line) editing functions such as displaying (TY), inserting (IN), and deleting (DE).

One example of intrarecord, or multiline editing is also given, as a basis for general use of the intrarecord-command group. Edit commands not covered here are described in the Edit/Reference Manual, Publication 90 16 33.

In the command descriptions that follow, the word "line" refers to a line typed by the user; the word "record" refers to a line that has already been transmitted to the system and exists on some file. Thus, we can say "...the line numbered n replaces any identically numbered record..." (i.e., already on the file) without ambiguity. The examples are intended to illustrate usage of the various commands and do not necessarily show the most appropriate way of dealing with a particular kind of file content. More appropriate means may become apparent in later chapters, especially in regard to manipulation of BASIC program text.

### HOW EDIT WORKS

Edit is a line-number oriented editor in that it automatically associates a line-sequence number with each line of a file built under Edit. All record and intrarecord editing is performed with reference to these sequence numbers. That is, one or more sequence numbers must be specified for record and intrarecord editing commands and also in certain usages of the file editing commands.

To edit a file that does not have sequence numbers associated with it (e.g., a file built under BASIC or under certain batch-mode facilities), you can add the numbers by copying the file with "resequencing" (see Example Copy and Resequence).

Edit prompts with an asterisk (*) character to indicate that it is ready to accept a command. For file-building or line-insertion input, it prompts with a sequence number.

### FILE EDITING COMMANDS

All file editing commands explicitly name one or more files. The Edit and TEL/Edit commands at the file-editing level are:

- !EDIT [fid]

  Calls the Edit subsystem and optionally names a file to be edited, at TEL level (!).

- *EDIT fid

  Names a file to be edited, at Edit subsystem level (*).

- !BUILD fid [n]

  Calls the Edit subsystem and names a file to be built, at TEL level (!).

- *BUILD fid [n]

  Names a file to be built, at Edit subsystem level (*). The optional number (n) specifies the sequence number with which the file is to begin. If not specified, 1 is assumed by Edit.

- COPY $fid_1$ {ON / OVER} $fid_2$ [,n]

  Copies contents of $fid_1$ either ON a new file or OVER an existing file, $fid_2$; and, optionally, resequences (i.e., renumbers) $fid_2$ starting with sequence-number n. COPY can also be used to produce a sequence-numbered (keyed) version of an unkeyed file, in which case n must be specified.

- DELETE fid

  Deletes the named file from the system.

- MERGE $fid_1$ [,$n_1$-$n_2$] INTO $fid_2$, $n_3$-$n_4$

  Replaces records $n_3$ through $n_4$ of $fid_2$ with the contents of (or record $n_1$ through $n_2$ of) $fid_1$; the merged records – from $fid_1$ – are renumbered in $fid_2$ starting with sequence-number $n_3$. Note: If $fid_2$ does not already exist, the specified records on $fid_1$ are copied to the new file and numbered starting with $n_3$ (i.e., a "selective copy" operation is performed).

- END

  Terminates execution of Edit and returns control to the system (TEL) level.

**RECORD EDITING COMMANDS**

To use any of the record or intrarecord editing commands, the applicable file must first be specified with an EDIT-fid command either at the TEL or the Edit subsystem level. None of the record and intrarecord-level commands themselves can specify a file.

A useful record-editing command is TY – Type Record(s), Including Sequence Number – which displays one, several, or all of the records in a file:

TY $n_1$ [-$n_2$]

where

$n_1$ is the sequence number of the first or only line to be typed.

$n_2$ is the optional ending sequence number of a range of lines to be typed.

More record-editing commands are described following the next example.

Example 12. Using EDIT to Build and Display a Source File

In this example, the user builds a BASIC program file, copies it to another file, displays the copy, and deletes the original file.

```
UTS AT YOUR SERVICE
ON AT 15:12 MAR 28,'71
LOGON PLEASE:  2232,HALL (RET)
```

- page heading -

```
!BUILD PRIME (RET)
```

The user wants to create a file called PRIME. Edit is called implicitly.

```
1.000  10   REM    GENERATE PRIMES GR THAN 3 (RET)
```

Edit prompts for input by printing 1.000. The user types the first line, then types lines 2-10 in response to more prompts by Edit.

```
2.000  20  P = 1 (RET)
3.000  30  P=P +4,S+0 (RET)
4.000  40  FOR I = 5 TO SQR (P) = 1 STEP 2 (RET)
5.000  50    Q=INT(P/I) (RET)
6.000  60    IF Q*I=P THEN 80 (RET)
7.000  70  PRINT P''TAB (0) (RET)
8.000  80  IF S=1 THEN 30 (RET)
9.000  90  S=1, P=P+2 (RET)
10.000  100  GOTO 40 (RET)
11.000  (RET)
```

The user types a carriage return immediately following the prompt character to indicate end-of-file, that is, that the last line of the file has been entered. (Control returns directly to TEL, rather than to Edit, because BUILD was given at the TEL level.)

```
!EDIT (RET)
```

TEL prompts for another command. The user calls Edit again, explicitly this time, to use a command not available at TEL level.

```
EDIT HERE
*COPY PRIME ON PRIMES (RET)
```

Edit acknowledges its presence, and prompts. The user decides to change the name of his program file from PRIME to PRIMES, so he copies it to a new file named PRIMES.

```
..COPYING
..COPY DONE
*EDIT PRIMES (RET)
```

He then indicates that he wants to edit (actually only display) file PRIMES.

```
*TY 1-10 (RET)
```

He indicates that he wants the whole file, lines 1 through 10, typed. (A larger ending number, e.g., TY 1-99, would do the same job.)

```
1.000  10  REM   GENERATE PRIMES GR THAN 3
2.000  20  P=1
3.000  30  P=P+4,S=0
4.000  40  FOR I = 5 TO SQR(P) + 1 STEP 2
5.000  50    Q=INT(P/1)
6.000  60    IF Q*I=P THEN 80
7.000  70  PRINT P''TAB(0)
8.000  80  IF S=1 THEN 30
9.000  90  S=1, P=P+2
10.000  100  GOTO 40
*DELETE PRIME (RET)
```

Edit displays the copy, and prompts. The user sees that the copy is OK and decides to delete the original file, PRIME, so as not to tie up RAD space unnecessarily.

..EDIT STOPPED
..DELETED
*END ⏎

He then indicates that he is finished with Edit.

!OFF ⏎

and logs off.

- accounting summary -

## MORE RECORD EDITING COMMANDS

Two more commonly used record-level commands are IN (Insert Records) and DE (Delete Records). The IN command is used to insert one or more lines between two records of a file or, alternatively, to replace one record of the file with the first (or only) insert line. (The IN command can be used to replace only one record, though more records may be inserted immediately following the replacement.) The IN command format is

IN n[,i]

where

n is the sequence number of the first or only line to be inserted.

i is the optional increment value that Edit is to add to succeeding insertion-line sequence numbers.

Detailed rules for the use of IN are given following the next example.

The DE command deletes one or more (successive) records from the file. It has the format

DE n[-m]

where

n is the sequence-number of the first or only record to be deleted.

m is the optional end sequence number of a range of records to be deleted.

Example 13. Using EDIT to Modify a Source File

In this example, the user (after "desk-checking" his initial source program) sees that a logically required BASIC statement (NEXT) is missing, and inserts it. He then realizes that this original program will produce an endless listing of prime numbers, and prepares a different version, using MERGE to excerpt a portion of the original program for modification, and then to recombine this portion, after modification, with a copy of the original (thus, retaining the original version also).

UTS AT YOUR SERVICE
ON AT 15:28 MAR 28,'71
LOGON PLEASE: 2232,HALL ⏎

- page heading -

!PLATEN 72,10 ⏎

The user suppresses further page headings by giving a page length of less than 12. (This practice is not recommended for normal production work, where the page headings delimit a uniform document size and provide useful identification: name, date, time, page number.)

```
!EDIT PRIMES (RET)
```

He then indicates he wants to edit PRIMES.

```
EDIT HERE
*TY 6-99 (RET)
```

He asks for display of lines 6 through end-of-program, i.e., line number 99 is in this case sufficiently large to include the whole file. (Note that "6-99" is equivalent to "6.0-99.0" or "6.000-99.000", etc.)

```
  6.000 60   IF Q*I=P THEN 80
  7.000 70 PRINT P''TAB(0)
  8.000 80 IF S=1 THEN 30
  9.000 90 S=1, P=P+2
 10.000 100 GOTO 40
--EOF HIT AFTER 10.
```

This message means: "End-of-file was found following line 10".

```
*IN 6.5 (RET)
```

The user asks to insert a line numbered 6.5, to add the missing statement.

```
 6.500 65 NEXT I (RET)
```

Edit prompts for the insertion-line with the line number. It then prompts for another command with an asterisk.

```
*TY 6-7 (RET)
```

The user requests a display of lines 6 through 7, to see if the insert really worked.

```
    6.000 60    IF Q*I=P THEN 80
    6.500 65 NEXT I
    7.000 70 PRINT P''TAB(0)
*MERGE PRIMES, 6.1-10 INTO NEWEND,7 (RET)
```

He then asks for a portion of PRIMES to be copied on a new, empty file, NEWEND, and for the lines to be renumbered, starting with 7.

```
..EDIT STOPPED
..MERGE STARTED
--DONE AT 11.
*EDIT NEWEND
*TY 1-11 (RET)
```

He requests a display to see if 7-11 was "excerpted" all right.

```
   7.000 65 NEXT I
   8.000 70 PRINT P''TAB(0)
   9.000 80 IF S=1 THEN 30
  10.000 90 S=1, P=P+2
  11.000 100 GOTO 40
*IN 7.5 (RET)
```

He requests an insert numbered 7.5, enters the insertion as shown below, and then requests an insert at the end of the file, i.e., line 12.

```
    7.500 66 IF P > 1000 GOTO 110(RET)
*IN 12(RET)
   12.000 110 END(RET)
   13.000 (RET)
```

Edit prompts for another insertion, line 13; the user replies with an immediate (RET), signifying "done".

```
*TY 6-12(RET)
```

He then requests display of lines 7-12 (no line lower than 7 should exist).

```
   7.000 65 NEXT I
   7.500 66 IF P >  1000 GOTO 110
   8.000 70 PRINT P''TAB(0)
   9.000 80 IF S=1 THEN 30
  10.000 90 S=1, P=P+2
  11.000 100 GOTO 40
  12.000 110 END
*COPY PRIMES TO LOPRIM(RET)
```

He requests an extra copy of PRIMES on new file LOPRIM.

```
..EDIT STOPPED
..COPYING
..COPY DONE
*MERGE NEWEND, 7-12 INTO LOPRIM,6.1-10(RET)
```

He then asks for a replacement of the original program lines 6.1-10, with the modified program ending from NEWEND.

```
..MERGE STARTED
--DONE AT 12.1
*EDIT LOPRIM(RET)
*TY 5-13(RET)
```

He requests display of lines 5 through end-of-file on LOPRIM.

```
   5.000 50   Q=INT(P/I)
   6.000 60   IF Q*I=P THEN 80
   6.100 65 NEXT I
   7.100 66 IF P > 1000 GOTO 110
   8.100 70 PRINT P''TAB(0)
   9.100 80 IF S=1 THEN 30
  10.100 90 S=1, P=P+2
  11.100 100 GOTO 40
  12.100 110 END
--EOF HIT AFTER 12.1
*DELETE NEWEND(RET)
```

Since NEWEND is now appended, he deletes the file for the sake of economy.

```
..EDIT STOPPED
..DELETED
*IN 1, .1(RET)
```

He then decides to replace the original 'remarks' line (1.000), and specifies a small increment to allow room for further insertion lines before line 2.

```
    1.000 REM GENERATE PRIMES OVER 3 AND UNDER 1000
    1.100 REM (THIS PROGRAM IS A LIMITED VERSION OF
    1.200 REM   MY PROGRAM "PRIMES", WHICH HAS NO
    1.300 REM   UPPER LIMIT BUILT IN.)
    1.400
*TY 1-15 (RET)
```

He requests display of result.

```
  1.000 REM GENERATE PRIMES OVER 3 AND UNDER 1000
  1.100 REM (THIS PROGRAM IS A LIMITED VERSION OF
  1.200 REM   MY PROGRAM "PRIMES", WHICH HAS NO
  1.300 REM   UPPER LIMIT BUILT IN.)
  2.000 20 P=1
  3.000 30 P=P+4,S=0
   .
   .
   .
  10.100 90 S=1, P=P+2
  11.100 100 GOTO 40
  12.100 110 END
--EOF HIT AFTER 12.1
*END (RET)

!OFF (RET)

- accounting summary -
```

## RULES FOR USE OF IN

The rules applicable to the IN command are summarized below. For ease of reference, the IN command format is repeated:

IN n[,i]

1. If n matches a sequence number already in the file, the first (or only) insertion line replaces the identically numbered line in the file.
2. If n does not match a sequence number in the file, the first (or only) insertion line n is inserted immediately following the next lower-numbered line (or at the beginning of the file if a lower line number does not exist).
3. If the insertion sequence number increment, i, is not specified, Edit assumes as a default value for i either the increment specified in the most recent record-level command given during the current Edit session, or the value 1 if no increment has been previously specified.
4. Following each record insertion, Edit prompts for further insertion lines with incremented sequence numbers, until either the incremented sequence number equals or exceeds a sequence number already existing in the file, or the user responds with a carriage return only. (In the first case, Edit rings the console bell and returns immediately to command-input mode, issuing an asterisk.)

## RULES FOR USE OF MERGE

A more complete form of the MERGE command than initially presented is

MERGE $fid_1[,n_1-n_2]$INTO $fid_2,n_3[-n_4][,i]$

The optional increment value, i, was not previously presented. It is used to control renumbering of merged records. For example, by specifying a small fractional (decimal) increment it is possible to pack more records into the destination file than might otherwise be possible. The rules for MERGE are as follows:

1. The sequence numbers $n_3 - n_4$ specify the range of records to be deleted from the destination file ($fid_2$), whether or not a one-for-one replacement occurs. (If $n_4$ is omitted only record $n_3$ is deleted, i.e., $n_3$ is assumed as the value for $n_4$.)

2. Sequence numbers $n_1 - n_2$ specify the maximum range of lines to be transmitted from the source file ($fid_1$); default value of $n_1 - n_2$ is 1 through EOF. (The actual number of records moved is controlled by the next sequence value above $n_4$; see rule 4 below.)

3. Renumbering of the records from $fid_1$ in $fid_2$ proceeds from $n_3$, incrementing either by i or the default value, 1.

4. Records $n_1$ through $n_2$ are moved into the interval $n_3 - n_4$ on $fid_2$, renumbered, until either the incremented sequence number of a moved record equals or exceeds the sequence number of the successor of $n_4$, or the range of records $n_1 - n_2$ is exhausted.

5. Value $n_2$ may equal $n_1$; $n_4$ may equal $n_3$.

Note these characteristics of MERGE: (1) the number of $fid_1$ records moved is largely independent of the number of $fid_2$ records deleted; (2) sequence number discontinuities may be introduced into $fid_2$; and (3) by adjusting the increment value, the set of deleted records may be replaced by a much larger set of records. Note also that though it is a file-level command, MERGE has record-editing capabilities.

The rules for IN and MERGE can be used as a general guide to the operation of other record-level commands with similar formats.

## STRING SEARCH COMMANDS

The string-search type of command involves an automatic search by Edit for the occurrence of a certain string of characters within specified columns of a range of records. The records are searched one at a time and, if a "hit" is made on one or more of the records, the action specified by the command is performed (type or delete record). You specify the range of records to be searched, the string to search for, and the record columns within which the search is to be made ("all" by default). Edit does the rest. Note that the line number is not considered a part of the record, and that column 1 is the character position following the line number and single space issued by Edit.

Two string-search commands are available at the record-editing level.

- Find and Type Records.
- Find and Delete Records.

The command formats are

FT $n_1$ [-$n_2$],/string/[,$c_1$,$c_2$]

and

FD $n_1$ [-$n_2$],/string/[,$c_1$,$c_2$]

where

$n_1$ is the sequence number of the first or only record to be searched.

$n_2$ is the sequence number of the last of a range of records to be searched (default value = $n_1$).

string delimited by slashes (/.../), is any sequence of characters that may exist in the file.

$c_1$ is the number of the column at which the search is to start in each record (default value = 1).

$c_2$ is the number of the column (inclusive) at which the search is to end in each record (default value = 140).

The specified string must be found entirely within the columns specified. The columns of a record (or line) are numbered from 1 through 140, and though 72 is the upper limit for a Teletype® line, columns 73-140 may exist in a record, as discussed below. (Other string-search commands are available at the intrarecord-editing level, and are generally more useful and efficient than those described above.)

## HOW TO ENTER MULTILINE RECORDS

On a terminal unit having an inherent line-width limit of less than 140 (e.g., Teletype® models 33, 35, and 37), a single, multiline record may be entered into a file (using the BUILD or IN commands, for example) in either of two ways:

1. Using the local-carriage-return key marked LOC CR, if present, to "break" the input line without releasing it to the system.

2. Using the simulated local-carriage-return sequence (ESC) (RET) for the same purpose.

Either method permits entering a record of up to 139 characters plus (RET) on virtually any terminal unit.

Example 14. Using String-Search Commands and Local-Carriage-Return

```
   .
   .
   .
!EDIT (RET)
EDIT HERE
*COPY LOPRIM TO SCRATCH (RET)
```

The user copies his program to a new file in order to experiment with FG, FD, and IN.

```
*EDIT SCRATCH (RET)
*FT 2-15,/P=/ (RET)
```

He requests a search of records 2 through 15, all columns, for the character string "P=", with the record displayed on each hit.

```
   2.000 20 P=1
   3.000 30 P=P+4,S=0
  10.100 90 S=1, P=P+2
--EOF HIT AFTER 12.1
*FT 1-13,/=P/ (RET)
```

He then asks for a search on "=P" in lines 1 through 13.

```
   3.000 30 P=P+4,S=0
   6.000 60   IF Q*I=P THEN 80
  10.100 90 S=1, P=P+2
--EOF HIT AFTER 12.1
*FT 1-2,/PR/ (RET)
```

He now changes the string to "PR", lines 1 and 2.

```
   1.000 REM GENERATE PRIMES OVER 3 AND UNDER 1000
   1.100 REM (THIS PROGRAM IS A LIMITED VERSION OF
   1.200 REM   MY PROGRAM "PRIMES", WHICH HAS NO
*FT 1-2,/REM/,4,60 (RET)
```

He tries a "negative" test of the column-delimiting capabilities,

```
--NONE
*FD 1.1-2,/REM/ (RET)
```

then a find-and-delete of records 1.1 through 2, inclusive, containing "REM".

```
--003 RECS DLTED
*TY 1-4 (RET)
```

He requests display of results.

```
    1.000 REM GENERATE PRIMES OVER 3 AND UNDER 1000
    2.000 20 P=1
    3.000 30 P=P+4,S=0
    4.000 40 FOR I= 5 TO SQR(P)+1 STEP 2.
*IN 1.5 (RET)
    1.500 REM (THIS PROGRAM IS A LIMITED VERSION OF (ESC)(RET)
PROGRAM "PRIMES", WHICH HAS NO SET UPPER LIMIT.) (RET)
```

He tries to reenter former lines 1.1 and 1.2 as one record, with a local line-break ((ESC)(RET)).

```
*FT 1-3,/REM/ (RET)
  1.000 REM GENERATE PRIMES OVER 3 AND UNDER 1000
  1.500 REM (THIS PROGRAM IS A LIMITED VERSION OF PROGRAM "PRIMES", WHIC
H HAS NO SET UPPER LIMIT.)
```

Note, (1) that the user neglected to supply a blank (or space) following "of" prior to or after the local carriage-return, and (2) that the system "folds" the record indiscriminately when the physical line-width limit is reached.

```
*END (RET)
```

## INTRARECORD COMMAND USAGE

As a group, intrarecord commands are characterized by not indicating the record(s) on which they are to operate; the exceptions are the specific record-selection commands SE, SS, and ST, which supply this indication. Therefore, one of the record-selection commands must precede any of the other intrarecord commands.

The SE (Select Intrarecord Mode) command simply selects a range of records, and optionally a field within each record, for subsequent intrarecord "processing" commands to operate on. The selection remains in effect for any number of subsequent commands until a new selection is made, or a record editing command is given.

After the SE is given, Edit prompts for further commands. You can then issue one processing command, or several commands separated by semicolons (;) on the same input line.

If one command is issued per line, the processing specified by that command is performed against each record in the range specified by the SE. However, if more than one command is issued per line the whole set of commands will be processed successively against the first record, then against the second record, etc. (Obviously, if the range selected is only one record, the result is the same in either case.)

The format of the SE command is

SE $n_1[-n_2][,c_1,c_2]$

where the meanings and defaults of the record and column selection parameters are the same as for the FT and FD string-searching commands.

Two very useful and similar processing commands are S (String Substitution) and D (Delete String). The S command format is

$[j]/string_1/S/string_2/$

where

$string_1$ is the string to be searched for.

$string_2$ is the string to be substituted in place of $string_1$.

j is an integer that indicates that only the jth occurrence of $string_1$ within the search field of each record is to be replaced by $string_2$ (default value = 1). If all occurrences of $string_1$ are to be replaced, j must be specified as zero.

The D command format is

[j]/string/D

where

string is the string to be deleted.

j has the same meaning as in the S command.

Note that when substituting a longer string for a shorter string, the remainder of the line (if any) is moved right only as far as needed to preserve a single-blank separator if at least one blank existed to the right of the original string. That is, in certain cases multiple blanks to the right of the insert may be lost. (This is useful in preserving columnar alignment.)

In this regard, you may include initial, embedded, or terminal blanks (i.e., spaces) in either string. Edit treats the blank in general like any other printing character, the major exception being the suppression of multiple blanks in certain cases of string substitution and deletion. (A Blank-Preservation-Mode command, BP, in intrarecord operations provides for cases where multiple blanks must not be lost, as in "quoted" character-string literals.)

A number of other very useful, more specialized intratrecord commands exist for record modifications, but most of these are logical shortcuts to results that can usually be achieved with S and D commands only.

The two intrarecord display commands, TY (Type, Including Sequence Number) and TS (Type, Suppressing Sequence Number) are analogous to their record-level counterparts, but do not specify record numbers (i.e., you enter TY or TS only). With or without sequence numbers, the commands display the currently active record(s), as illustrated by the following example.

Example 15. Using Intrarecord Commands

```
   :
!EDIT SCRATCH (RET)
EDIT HERE
*FT 1-2,/PROG/(RET)
```

The user enters the FT command to find and type lines containing PROG, within the range 1 through 2, inclusive. (Only line 1.5 should satisfy the requirement.)

```
    1.500 REM (THIS PROGRAM IS A LIMITED VERSION OFPROGRAM "PRIMES", WHIC
H HAS NO SET UPPER LIMIT.)
*SE 1.5(RET)
```

He then enters the intrarecord mode selection command, which is required to fix the error in the line by string substitution.

```
*/OFPRO/S/OF PRO/ (RET)
```

He substitutes OF PRO for OFPRO (first instance only: j = 1 by default).

```
*TY (RET)
   1.500 REM (THIS PROGRAM IS A LIMITED VERSION OF PROGRAM "PRIMES", WHI
CH HAS NO SET UPPER LIMIT.)
*TS (RET)
```

To see if the line-break problem in line 1.5 would disappear if the line were displayed without its sequence number (as will happen under the BASIC subsystem), he uses the TS command.

```
REM (THIS PROGRAM IS A LIMITED VERSION OF PROGRAM "PRIMES", WHICH HAS NO
 SET UPPER LIMIT.)
*0/P/S/N/ (RET)
```

This does in fact solve the problem. Now he wants to change all program variables named P to Ns. (This will have no effect on the program, since we have no variables named N.)

```
*TY (RET)
   1.500 REM (THIS NROGRAM IS A LIMITED VERSION OF NROGRAM "NRIMES", WHI
CH HAS NO SET UNNER LIMIT.)
```

He forgot to reset the range selection (SE).

```
*0/N/S/P/ (RET)
*SE 2-13 (RET)
*0/P/S/N/ (RET)
```

He reverses the N for P substitution in line 1.5, then sets proper range, and tries his original substitution again;

```
--EOF HIT AFTER 12.1
*TY (RET)
```

and checks the result.

```
   2.000 20 N=1
   3.000 30 N=N+4,S=0
   4.000 40 FOR I=5 TO SQR(N)+1 STEN 2
   5.000 50   Q=INT(N/I)
   6.000 60   IF Q*I=N THEN 80
   6.100 65 NEXT I
   7.100 66 IF N > 1000 GOTO 110
   8.100 70 NRINT N''TAB(0)
   9.100 80 IF S=1 THEN 30
  10.100 90 S=1, N=N+2
  11.100 100 GOTO 40
  12.100 110 END
--EOF HIT AFTER 12.1
```

The substitution worked, except that it was not possible to delimit the search string narrowly enough; STEP to STEN, and PRINT to NRINT, were changed as well.

```
*/STEN/S/STEP/;/NRI/S/PRI (RET)
```

This reverses the change.

```
--C1:NO SUCH STRG
*SE 4;TY (RET)
```

(The meaning of this message is that the searched-for string was not found in at least one of the records in the range of the search. As in this case, it does not necessarily indicate an error condition.)

The user requests a display of line 4 and, below, of line 8.1.

```
   4.000 40 FOR I=5 TO SQR(N)+1 STEP2
*SE 8.1;TY (RET)

   8.100 70 PRINT N''TAB(0)
*DELETE SCRATCH (RET)
```

Since he did not actually need this file, he deletes it.

```
..EDIT STOPPED
..DELETED
*END (RET)

!OFF (RET)

- accounting summary -
```

### TEL EDITING COMMANDS VS EDIT COMMANDS

The TEL command !EDIT fid implies the sequence

```
!EDIT
*EDIT fid
```

The TEL command !BUILD implies the sequence

```
!EDIT
*BUILD
:
*END
!
```

Both are, therefore, shortcuts provided for your convenience. However, note that the TEL EDIT command must be given before the Edit COPY command can be used, as distinct from the TEL COPY command. The TEL COPY command implies a call to the PCL subsystem; the COPY command under PCL is different from the Edit COPY command in scope, intent, and format.

TEL/PCL COPY and other PCL commands are described in the next section.

## PCL SUBSYSTEM

The peripheral Conversion Language, PCL, provides you with on-line facilities for initiating and controlling:

- Movement of files between peripheral storage devices.
- Movement of files between peripheral storage devices and RAD storage (or other forms of secondary storage).
- Movement of files within RAD storage.
- Concatenation of files and selection of records from files during file movement.
- Data-record formatting and code conversion during file movement.
- Deletion of files.
- File building on any type of device or storage media from an on-line terminal.

- Display of peripheral input-device files or RAD files on an on-line terminal.
- Listing of a RAD file directory or of file names on a labeled magnetic tape.
- Positioning (and releasing) of magnetic-tape volumes.

The peripheral storage devices referred to may be

1. Magnetic-tape drives: labeled or unlabeled tape.
2. Unit-record devices: card punch and line printer (card reader cannot be requested on-line).

As mentioned before, one common characteristic of peripheral devices is that they generally require operator intervention, e.g., for the mounting and dismounting of physical file volumes. Therefore, an on-line user must be specially authorized in order to be able to use these devices via PCL (or any other on-line means); otherwise he will simply receive an error message on any attempt to do so.

Many of the facilities listed above are mainly of interest to the experienced on-line user doing the kinds of programming that were heretofore necessarily restricted to central-site batch operations: commercial and large-scale scientific applications involving large volumes of input and output data, system development, etc. Actually, the complete set of PCL facilities, plus the TEL SET command and direct user-to-operator messages, provide control of total system resources analogous to that obtainable only with "hands on", central-site batch operations under previous systems.

We will describe only the PCL functions, commonly used by all on-line users. These include keyed-file merging, building of unkeyed files, concatenation of unkeyed files, terminal display of either type of file, listing of file names, and file deletion.

**PCL COMMANDS**

The PCL COPY command may be given at TEL level, but PCL must be called explicitly (!PCL) for all other PCL commands. PCL prompts for command input with the less-than (<) character, and for file input and responses to questions with a period.

The PCL commands covered here are COPY, LIST, DELETE, and DELETEALL. COPY allows a vast array of options in its variable field; it is the workhorse of the PCL language. Therefore, only a subset of the possible variations of the command is described here.

The COPY command format is

$$\text{COPY}\,[d]\left[/\text{fid}_1\,[,\text{fid}_2,\ldots,\text{fid}_n]\right]\left[\left\{\begin{matrix}\text{ON}\\\text{OVER}\end{matrix}\right\}[d][/\text{fid}_m]\right]$$

where

d is a device-identification code, which may include

DC - RAD file storage (default value for d).

ME - User's terminal.

LP - Line printer.

CP - Card punch.

LT [#reel no. ... #reel no.] - labeled magnetic tape (reel no. default is "scratch tape").

FT [#reel no. ... #reel no.] - free-form magnetic tape, i.e., unlabeled.

fid is a file identification, for DC or LT files only; normally only a file name. Each device code or fid can be followed immediately by one or more special options in parentheses: i.e., d(option) or d/fid(option). One such option is NC, which we will show in use further on. If the default device code DC is not explicitly specified, the slash (/) preceding $\text{fid}_1$ should be omitted; see the following example.

The choice of ON or OVER is made as for the Edit COPY command (TO may be substituted for ON). The ON/OVER clause is optional following a prior COPY specifying an ON/OVER destination file or device (during the same session with PCL). If the ON/OVER clause is omitted under these circumstances, the last-named file will be extended according to the file-extension convention. A subsequent ON/OVER clause or an exit from PCL terminates file extension.

If multiple source files – e.g., FILA, FILB, FILC – are specified, the several file contents are either concatenated, i.e., joined end to end, on the destination file in the case of unkeyed files, or merged on the basis of record-key values in the case of keyed files. Both cases are illustrated in the following examples.

Even compared to just a partial version of the COPY command, the full LIST, DELETE, and DELETEALL commands are simple:

LIST – lists all your RAD-file names (i.e., all names in your account directory).

or

LIST LT #reel no. ... [#reel no.] – lists all labeled-tape file names on the specified reels.

DELETE $fid_1[,fid_2,\ldots,fid_n]$ – deletes the named files.

DELETEALL – causes PCL to ask for a confirmation:

DELETEALL?

_YES$ – then it deletes all of your RAD files. ("YES$" is the only correct positive response.)

Example 16. Keyed-File Update and Display, Using PCL COPY

The user wants to produce another version of the PRIMES program that will allow him to set, via the terminal, the range of the prime numbers produced during each run. He creates the modification files using Edit BUILD, but uses PCL COPY to achieve the actual file updating.

```
  :
!BUILD MOD1, 1, .125
```

The user wants to build a file starting with sequence number 1 and incrementing by only .125, instead of the standard (default) increment of 1. Note that we have added an i parameter (.125) to BUILD that corresponds to that of MERGE and INsert.

```
  1.000 10 REM GENERATE PRIME NUMBERS (>3) WITHIN USER-SET LIMITS (RET)
  1.125 11 PRINT'ENTER LOWER BOUND FOR PRIMES' (RET)
  1.250 12 INPUT L (RET)
  1.375 13 PRINT'ENTER UPPER BOUND FOR PRIMES' (RET)
  1.500 14 INPUT U (RET)
  1.625 (RET)
!BUILD MOD2, 7.1, .5 (RET)
```

He requests a second new file, starting with sequence number 7.1, but incrementing by .5 in this case.

```
  7.100 67 IF P < L THEN 80 (RET)
  7.600 68 IF P > U THEN 110 (RET)
  8.100 (RET)
!COPY LOPRIM,MOD1,MOD2 TO VPRIM (RET)
```

He requests PCL to copy files LOPRIM, MOD1, and MOD2, in succession to form new file VPRIM. Note that these are keyed files, and as such are not simply linked together end-to-end on VPRIM. MOD1 is merged with LOPRIM, records from MOD1 replacing any records from LOPRIM having matching keys, and all nonmatching records falling into their natural sequence. The same process is repeated between MOD2 and the results of LOPRIM,MOD1 – and so on if more files were specified. (The source files themselves are not modified in any way.)

```
!COPY VPRIM TO ME (NC) (RET)
```

He displays the results directly, using the PCL COPY command. Note that ME is a device code, not a name; VPRIM, not being a device code, is understood by default as DC/VPRIM. The NC option effectively prevents double-spacing of the terminal display by stripping the "redundant" carriage-control characters that are included in any file built with the Edit subsystem.

```
10 REM GENERATE PRIME NUMBERS (>3) WITHIN USER-SET LIMITS
11 PRINT 'ENTER LOWER BOUND FOR PRIMES'
12 INPUT L
13 PRINT 'ENTER UPPER BOUND FOR PRIMES'
14 INPUT U
20 P=1
    :
    :
65 NEXT I
67 IF P<L THEN 80
68 IF P>U THEN 110
70 PRINT P''TAB(0)
    :
    :
110 END

!EDIT VPRIM (RET)
EDIT HERE
*TY 1-13 (RET)
```

The user then displays the same results using Edit.

```
   1.000 10 REM GENERATE PRIME NUMBERS (>3) WITHIN USER-SET LIMITS
   1.125 11 PRINT 'ENTER LOWER BOUND FOR PRIMES'
   1.250 12 INPUT L
   1.375 13 PRINT 'ENTER UPPER BOUND FOR PRIMES'
   1.500 14 INPUT U
   2.000 20 P=1
   3.000 30 P=P+4,S=0
   4.000 40 FOR I=5 TO SQR(P)+1 STEP 2
   5.000 50   Q=INT(P/I)
   6.000 60   IF Q*I=P THEN 80
   6.100 65 NEXT I
   7.100 67 IF P<L THEN 80
   7.600 68 IF P>U THEN 110
   8.100 79 PRINT P''TAB(0)
   9.100 80 IF S=1 THEN 30
  10.100 90 S=1, P=P+2
  11.100 100 GOTO 40
  12.100 110 END
--EOF HIT AFTER 12.1
*END (RET)
      :
      :
```

The last example points to several differences between the Edit COPY and the PCL COPY: The Edit Copy can only specify RAD filenames; the PCL COPY can specify or imply devices (e.g., ME, DC, LT) and filenames, either singly or in combination as appropriate. Note that the specification DC/ME is possible and results in no ambiguity, though in this case "DC/" must be specified. A second difference is that the PCL COPY TO ME, though it accepts keyed files, does not display the keys as sequence numbers as does Edit TY; it is functionally the same as Edit TS in this respect.

The next example is designed simply to illustrate these differences as well as to further clarify the merging action of PCL COPY on keyed files.

Example 17. Keyed-File Update and Display (Further Examples)

```
      :
      :
!BUILD FILA (RET)
   1.000 LINE 1 IN FILA (RET)
   2.000 LINE 2 IN FILA (RET)
   3.000 LINE 3 IN FILA (RET)
   4.000 LINE 4 IN FILA (RET)
   5.000 (RET)
!BUILD FILB, .5, .5 (RET)
```

Here the user requests a new file starting with sequence number .5 and incrementing by .5 also

```
   0.500 LINE 1 IN B (RET)
   1.000 LINE 2 IN B (RET)
   1.500 LINE 3 IN B (RET)
   2.000 LINE 4 IN B (RET)
   2.500 LINE 5 IN B (RET)
   3.000 LINE 6 IN B (RET)
   3.500 LINE 7 IN B (RET)
   4.000 (RET)
!BUILD FILC, 2, .75
```

and a new file starting at 2 and incrementing by .75.

```
   2.000 LINE 1 IN C (RET)
   2.750 LINE 2 IN C (RET)
   3.500 LINE 3 IN C (RET)
   4.250 LINE 4 IN C (RET)
   5.000 LINE 5 IN C (RET)
   5.750 LINE 6 IN C (RET)
   6.500 (RET)
!COPY FILA, FILB, FILC TO DC/ME (RET)
```

He combines the three files on new (RAD) file ME.

```
!COPY DC/ME (NC) (RET)
```

With PCL COPY he displays file ME on device ME, with the NC (no carriage-controls) option specified.

```
LINE 1 IN B
LINE 2 IN B
LINE 3 IN B
LINE 1 IN C
LINE 5 IN B
LINE 2 IN C
LINE 6 IN B
LINE 3 IN C
LINE 4 IN FILA
LINE 4 IN C
LINE 5 IN C
LINE 6 IN C

!EDIT ME (RET)
EDIT HERE
*TY .5-6 (RET)
```

Then he displays it with Edit.

```
   0.500 LINE 1 IN B
   1.000 LINE 2 IN B
   1.500 LINE 3 IN B
   2.000 LINE 1 IN C
   2.500 LINE 5 IN B
   2.750 LINE 2 IN C
   3.000 LINE 6 IN B
   3.500 LINE 3 IN C
   4.000 LINE 4 IN FILA
   4.250 LINE 4 IN C
   5.000 LINE 5 IN C
   5.750 LINE 6 IN C
--EOF HIT AFTER 5.75
*END (RET)
```

Note that the merging action of the multiple-file PCL COPY eliminates duplicately keyed records on file ME by successive replacement: record n from FILB replaces record n from FILA, and is in turn replaced by record n (if any) from FILC. Only record 4.000 survives from FILA, for example.

```
!PCL (RET)
PCL HERE
<LIST DC (RET)
```

Now he asks for a listing of current RAD-file names, to see which are deletable.

```
FILA
FILB
FILC
LOPRIM
ME
MOD1
MOD2
PRIMES
VPRIM
 DELETE FILA,FILB,FILC,ME,MOD1,MOD2 (RET)
..   6 FILES DELETED
<END (RET)
    .
    .
    .
```

Example 18. Building and Concatenating Unkeyed Files

In this example, the user creates two unkeyed files using PCL COPY; in most real instances of file concatenation, however, the files are outputs of other processors, e.g., FORTRAN. The user copies the files in the desired order to a single new file. A display of the resultant file shows the ordering of records produced by a multiple unkeyed-file copy. This example also shows how to copy a file to the system line printer. (Note that permission for such use of central-site peripherals requires explicit installation authorization; the system carries a record of this authorization.)

```
   .
   .
   .
!PCL (RET)
PCL HERE

<COPY ME TO A (RET)
```

The user requests a copy of terminal input to file A.

```
.1ST LINE IN A (RET)
.2ND LINE IN A (RET)
.3RD LINE IN A (RET)
.4TH LINE IN A (RET)
. (ESC)F (RET)
```

He enters input to new file A from the terminal. PCL prompts for input of each data line. An Escape-F sequence ends the data input, i.e., indicates end-of-file.

```
<COPY ME TO B (RET)
.1ST LINE IN B (RET)
.2ND LINE IN B (RET)
.3RD LINE IN B (RET)
.4TH LINE IN B (RET)
. (ESC) F
```

He enters input to new file B from the terminal.

```
<COPY DC/A,B TO DC/C (RET)
```

He copies files A and B in succession to form new file C, incidentally showing the syntax of explicit device identification (optional) in the case of multiple-file specification.

```
<COPY C TO ME(NC) (RET)
```

He now copies the contents of file C to the terminal.

```
1ST LINE IN A
2ND LINE IN A
3RD LINE IN A
4TH LINE IN A
1ST LINE IN B
2ND LINE IN B
3RD LINE IN B
4TH LINE IN B
```

```
<COPY  A TO LP (RET)
```

He then asks PCL to print file A on the system printer.

```
<DELETE A (RET)
..   1 FILES DELETED
<END (RET)
!
```

# 5. USING LANGUAGE PROCESSORS

## INTRODUCTION

The term "language processor" refers to a UTS subsystem that processes a specific programming language. Such processing consists essentially of some form of translation of the source language to the internal language of the computer, or machine language. (This machine-language translation is also commonly referred to as "object code".)

The language processors available under UTS in on-line mode are BASIC, Extended FORTRAN IV, and Meta-Symbol. Although these processors are also available for batch-mode operations, this guide is limited to a description of their on-line usage.

It is important at this point to distinguish between a programming language and the on-line command language associated with it. You use statements (i.e., sentences) of the programming language to form a program, whereas the commands are used to control what is done to or with that program. This chapter is intended to illustrate elementary uses of the command languages. (Succeeding chapters cover increasingly complex usages.) Therefore, to understand the program content of any of the following examples, knowledge of the relevant programming languages is necessary.

The following manuals contain descriptions of the languages processors:

Xerox BASIC/LN, OPS Reference Manual, Publication 90 15 46.

Xerox Extended FORTRAN IV/LN Reference Manual, Publication 90 90 56.

Xerox Extended FORTRAN IV/OPS Reference Manual, Publication 90 11 43

Xerox Symbol and Meta-Symbol/LN, OPS Reference Manual, Publication 90 09 52.

## BASIC SUBSYSTEM

The UTS BASIC processor is a compiler for a significantly extended and enhanced XDS version of the standard BASIC language (Beginner's All-Purpose Symbolic Instruction Code), a mathematical language designed specifically for time-sharing usage.

BASIC is particularly suited to small and medium scale computational applications. The outstanding advantage of BASIC is that it is easy to learn and simple to use. It is an ideal "starter" language, even though it does offer sophisticated problem-solving capabilities.

The BASIC subsystem is called with the TEL command BASIC. The subsystem then prompts for either BASIC program statements or BASIC commands with a "greater than" (>) character; during program execution, it prompts for program-requested terminal input (if any) with a question mark (?). When you have finished using BASIC, you exit back to TEL by giving the SYS(tem) command.

Since BASIC includes a program-building and editing facility, a program file need not be built under EDIT (as is the general case for FORTRAN and Meta-Symbol programs).

A useful XDS enhancement of BASIC is its capability for direct execution of individual statements. This allows you to operate UTS BASIC in the "desk-calculator mode", without building a program; it also provides a powerful on-line debugging feature. These topics are discussed in a later section of this chapter.

### PROGRAM BUILDING, EDITING, AND EXECUTION

Having called BASIC, you build a source program simply by entering BASIC program statements — each beginning with the required one-five digit step number (see following example)—in response to the prompt character (>). Typing error corrections can be made before the line is released with the RUB or $X^c$ controls as usual. Program statements entered in this fashion reside in an internal program-text area and constitute the current program.

The complete set of statements that are to constitute a given program need not be entered consecutively (e.g., BASIC commands may intervene), or entered in a sequence corresponding to their step numbers. The step numbers of the individual statements completely control the logical ordering of the statements within the program, providing for automatic insertion, replacement, and deletion of single statements on the basis of relative step numbers, as follows:

- Insertion - A statement entered with a step number falling in numerical sequence between the step numbers of two previously entered statements is automatically inserted between those two statements.
- Replacement - A statement entered with a step number matching the step number of a previously entered statement automatically replaces that previously entered statement.
- Deletion - A step number followed immediately by (RET), i.e., a "null statement", causes any previously entered statement having a matching step number to be deleted.

(Explicit editing commands that can affect more than one statement are covered in a subsequent section.)

After entering a program in this manner, you can have it compiled, error-checked, and executed (if no detectable errors exist) by issuing the RUN command. Syntax (i.e., language) errors, if any, will be reported by the subsystem, and a prompt character (>) issued. You may at this point correct these errors, via statement insertion, replacement, or deletion as described above. Note that when terminal input is requested by your program during its execution, a question mark (?) is issued as a prompt character.

Once a program has been tested and is known to be working correctly, you can request subsequent executions with the command FAST instead of RUN. FAST bypasses the checking of indices for subscripted variables.

The following example illustrates these elementary operations. In the sample program, three intrinsic – or built in – functions are used: DEG(x) - convert x from radians to degrees; ASN(x) - calculate arcsin of x, in radians; and ABS(x) - use absolute value of x. The first two, DEG and ASN, are specific UTS additions to the standard BASIC language. Also used in this example is the >SET command (distinct from the TEL SET), which affects the maximum size of strings assigned to character-string variables within the program. This type of variable is another significant UTS BASIC extension. The SET command usage in the example is self-explanatory. (SET is also used to set maximum array dimensions.)

Example 19. BASIC Program Building, Editing, and Execution

```
    :
    :
!BASIC
```

The user calls the BASIC subsystem, and begins to build a program, entering a BASIC statement in response to each prompt character.

```
>10 REM SAMPLE PROGRAM (RET)
>15 REM  $A IN STMT 20 IS A STRING VARIABLE (RET)
>20 $A = "COMPUTE ARCSINE OF X, IN DEGREES" (RET)
>30 PRINT $A (RET)
>40 FOR I -\= 1 TO 5 (RET)
```

After typing the minus-sign (or dash) character by mistake – i.e., by forgetting to shift – he uses a (RUB), echoed as \, to erase it and continues.

```
>50   INPUT X (RET)
>60   PRINT DEG (ASN(X)) " = ARCSIN OF "X (RET)
>70 NEXT I (RET)
>80 END (RET)
>RUN (RET)
```

He enters the final statement (step 80) and then requests compilation and execution with the RUN command.

```
16:13     NOV 09 RUNIDAA...
COMPUTE ARCSINE OF
?.5 (RET)
```

He notes that the complete character string assigned to $A was not printed (the default maximum string length being 18 characters), but he proceeds to test the program by entering a commonly-known sine value, .5 (sine 30° = .5).

```
30.0000 = ARCSIN OF .500000
? (BRK) ←
```

He gets a correct answer, then halts further execution with a (BRK) response to an input prompt, which is echoed by a "←". This returns him to editing/command level.

```
>SET $ = 40 (RET)
```

With a SET $ command he resets the maximum string length for a character variable to 40, from its default value of 18.

```
>RUN (RET)
16:18     NOV 09 RUNIDAA...
COMPUTE ARCSINE OF X, IN DEGREES
?.001 (RET)
 5.72958E-02 = ARCSIN OF 1.00000E-03
?.707 (RET)
 44.9913 = ARCSIN OF .707000
?-0.707 (RET)
-44.9913 = ARCSIN OF -.707000
?3.246 (RET)
```

He now tries a value that is much too large.

```
  60 ASN-ACS ARG ERROR
```

He gets a subsystem error message, and a return to editing/command level (where he will enter additional program statements for detecting the out-of-range condition).

```
>55 IF ABS(X) > 1 THEN 90 (RET)
>90 PRINT X; "VALUE OUT OF RANGE" (RET)
>95 GOTO 70 (RET)
>RUN (RET)
```

After inserting steps 55, 90, and 95, he tests again.

```
16:27     NOV 09 RUNIDAA...
COMPUTE ARCSINE OF X, IN DEGREES
?1.5 (RET)
 1.50000  VALUE OUT OF RANGE
? (BRK) ←
```

He gets the desired result on the exception condition, and terminates execution.

```
>
```

⋮

**PROGRAM SAVING, LOADING, AND RENAMING**

Programs created under BASIC can be saved on either a temporary or permanent file with the SAVE command, and can be subsequently reloaded for execution with the LOAD command. The command form SAVE ON filename (where filename does not name an already-existing file) creates a temporary file named as specified, on which your current program is copied. Being temporary, a file so created is released automatically at log-off, but can be retrieved by

name anytime before log-off. If, on the other hand, you use the command SAVE OVER filename, your current program is copied on the named file. If the specified name is not that of an already existing file, a new permanent file is created. If the SAVE OVER command specified an existing temporary file, e.g., one created by a SAVE ON during the current session, the program will be copied on it but the file will remain a temporary one.

To retrieve a saved program, whether from a permanent or temporary file, you use the command LOAD filename. In general, the LOAD command causes loading of the named program into the program-text area, but the results of this loading will depend upon the state of this area at the time the command is given. If the program-text area is empty, i.e., no current program has been entered or loaded during the current BASIC session, the saved program simply becomes the current program.

If, however, the program-text area is not empty at the time of the LOAD, the result depends on the current operating mode, or status. If the status is not "running", i.e., not execution mode, then the statements of the saved program are "woven" into the current program, on the basis of step numbers. This "weaving" process is analogous to a linked-file PCL COPY $f_1$, $f_2$ to $f_3$.

The result is not usually the one desired; it can be circumvented by using the command CLEAR, prior to the LOAD. The CLEAR command clears the contents of the program-text area, i.e., the current program. This command may be given at any time. If the subsystem is in execution mode, or "running", the program-text area is automatically clear prior to loading. The operating mode can be ascertained at any time by use of the STATUS command. Use it frequently to become familiar with mode transitions. (The responses to STATUS are EDITING, COMPILING, or RUNNING.)

## ADDITIONAL EDITING FACILITIES

Two BASIC editing commands that facilitate the display and deletion of current-program statements are LIST and DELETE, respectively. These two commands have identical formats; LIST is shown:

LIST $[s_1\ [-s_2]][,s_3\ [-s_4\ ]...]$

where $s_i$ is a step number.

If one or more pairs of dash-separated step numbers ($s_1$-$s_2$, etc.) are specified, the corresponding range(s) of statements are listed or deleted. If only $s_1$ (,$s_3$, etc.) is specified, only the corresponding individual statement(s) is listed or deleted. If no step number is specified in a LIST command, the entire current program is listed. Note: The command form DELETE (RET), is ignored; to delete the entire program you must use the CLEAR command.

Example 20. Program Modification, Saving, and Reloading

```
    :
    :

      (This example takes up at the point at which we left Example 19.)

>10 (RET)
>15 (RET)

      The user deletes the two REM statements (which are not necessary to the program's operation) and,
      below, requests a display (LIST) of the current program.

>LIST (RET)
20 $A = "COMPUTE ARCSIN OF X, IN DEGREES"
30 PRINT $A
40 FOR I = 1 TO 5
50   INPUT X
55   IF ABS(X) > 1 THEN 90
60   PRINT DEG(ASN(X)) ' = ARCSIN OF 'X
70 NEXT I
80 END
90 PRINT X; "VALUE OUT OF RANGE"
95 GOTO 70
```

```
>25 REM NEXT STMT SHOWS STRING CONCATENATION (+) (RET)
>30 PRINT $A + ", TESTING FOR OUT-OF-RANGE VALUES" (RET)
>40 FOR I =1 TO 2 (RET)
>LIST 20-40 (RET)
```

He inserts step 25, replaces steps 30 and 40, and lists steps 20 through 40 to observe the results.

```
20 $A = "COMPUTE ARCSIN OF X, IN DEGREES"
25 REM NEXT STMT SHOWS STRING CONCATENATION (+)
30 PRINT $A + ", TESTING FOR OUT-OF-RANGE VALUES"
40 FOR I = 1 TO 2
>RUN (RET)
14:51   NOV 10 RUNIDAA...
COMPUTE ARCSIN OF X, IN DEGREES, TESTING
? (BRK)←
```

He breaks off execution, noting that the maximum string-length limit, which he set previously to 40, is still too small for concatenated string that he has now assigned to $A.

```
>SET $=72 (RET)
>RUN (RET)
14:53    NOV 10 RUNIDAA...
COMPUTE ARCSIN OF X, IN DEGREES, TESTING FOR OUT-OF-RANGE VALUES
?.253877 (RET)
14:7071 = ARCSIN OF  .253877
?-.00000009 (RET)
-5.15662E-06 = ARCSIN OF -9.00000E-08

     80 HALT

>45 PRINT "ENTER SINE VALUE, PLEASE" (RET)
```

He inserts a final modification, then saves the program on new permanent file ARCSINE.

```
>SAVE OVER ARCSINE (RET)
>CLEAR (RET)
>LOAD ARCSINE (RET)
```

After clearing the program-text area, he loads the saved copy back in, and tries it once more (note the new current-program name when he executes it again).

```
>RUN (RET)
15:04    NOV 10 ARCSINE...
COMPUTE ARCSIN OF X, IN DEGREES, TESTING FOR OUT-OF-RANGE VALUES
ENTER SINE VALUE, PLEASE
?.001 (RET)
 5.72958E-02 = ARCSIN OF 1.00000E-03
ENTER SINE VALUE, PLEASE
?.002 (RET)
 .114592 = ARCSIN OF 2.00000E-03


     80 HALT

>
```

## TEMPORARY SAVING, RENAMING, AND RENUMBERING OF CURRENT PROGRAM

The FILE and NAME commands, used in conjunction with CLEAR and LOAD, provide a convenient short-cut means of temporarily saving the current program, e.g., for "back-up" purposes prior to extensive modification, and of renaming the current program for the execution-report heading. (You will have noticed a default program name, i.e., RUNIDAA, in the previous examples – this default name varies from session to session.)

The command FILE simply causes the current program to be copied onto a temporary file (known as the "runfile" in other XDS operating-system environments). This copy of the program can be explicitly named by using the command NAME newname prior to the FILE command. If the NAME command is not used (or no name is specified) the default program name applies. At any point after a program has been FILEd, a CLEAR and then a LOAD, with no filename, reestablishes the filed copy as the current program. The copy will remain on file during the whole terminal session until another program is FILEd over it.

When using FILE and LOAD (no name), it is important to remember that these commands always refer to the last runfile referred to with a NAME command, if one or more have been issued. If not, the default runfile name is "current". (The default runfile name can be reestablished with a null NAME command, i.e., simply NAME (RET).) Multiple runfiles can exist concurrently, resulting from multiple pairs of NAME newname and FILE commands having been issued; they can be selectively retrieved by a LOAD name command. Note: If NAME newname is used, newname may not also be used as the name of a permanent file during the same terminal session.

The command sequence for changing the execution name of a current program would be:

```
:
:
>NAME newname
>FILE
>CLEAR
>LOAD
```

Note that in this instance the CLEAR command that precedes the LOAD is functionally unnecessary since the current program and the filed program are identical, but it is included because of resultant savings in processing time and space.

At any time you can cause your current program to be automatically renumbered by giving a RENUMBER (or REN) command. Its format is

REN[UMBER] [$s_1$ [,$s_2$ [,i]]]

where

$s_1$ is the initial new step number (default value = 100).

$s_2$ is the old step number at which to begin renumbering (default value = 1, i.e., "first statement").

i is the increment by which successive new step numbers are to be increased (default value = 10).

(For example, REN alone is equivalent to RENUMBER 100, 1, 10; REN, 10 equivalent to RENUMBER 100, 10, 10; REN, , 5 to RENUMBER 100, 1, 5.)

This command allows you to clean up, or regularize, the numbering of your program with full control over the starting value, the point in the program at which to begin, and the step-value spacing. During the renumbering process, proper replacements are made for all step-number references, i.e., in GOTO statements, THEN clauses, etc., within the program. If the renumbered program was loaded from a permanent file, simply resaving it over the same file will make the renumbering "permanent".

Example 21. Temporary Filing, Reloading, and Renaming

```
    :
    :
```

(Continued from previous example.)

```
>FILE (RET)
```

The user files the current program (ARCSINE) on a temporary "runfile", under the default "runfile" name — whatever that is.

```
>CLEAR (RET)
```

He then clears the current program, to play safe.

```
>LOAD VPRIM (RET)
>RUN (RET)
```

He loads and executes the program VPRIM, previously built under Edit. (If it were now to be resaved over VPRIM, that file would no longer be keyed, as BASIC is not a keyed-file oriented subsystem — BASIC step numbers are part of the file records, not record keys.)

```
18:30    NOV 12 VPRIM...
ENTER LOWER BOUND FOR PRIMES
?100 (RET)
ENTER UPPER BOUND FOR PRIMES
?250 (RET)
 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191

 193 197 199 211 223 227 229 233 239 241
     180 HALT
>STATUS (RET)
```

Following a successful execution of VPRIM, he requests the current subsystem status.

```
 RUNNING
>LOAD (RET)
```

He now loads the default-named temporary file, without a preceding CLEAR, since clearing is automatic in execution mode.

```
>RUN (RET)
18:05    NOV 12 RUNIDAA...
COMPUTE ARCSIN OF X, IN DEGREES, TESTING FOR OUT-OF-RANGE VALUES
ENTER SINE VALUE, PLEASE
? (BRK)←
>NAME ARC (RET)
```

He breaks off execution, names another temporary "runfile", and (below) files the current program on it.

```
>FILE (RET)
>CLEAR (RET)
>LOAD (RET)
```

Note here that LOAD with no name specified will refer to the last-named "runfile" (if any — otherwise to the default-named "runfile").

```
>RUN (RET)
18:09    NOV 12 ARC...
COMPUTE ARCSIN OF X, IN DEGREES, TESTING FOR OUT-OF-RANGE VALUES
ENTER SINE VALUE, PLEASE
?.4 (RET)
```

```
 23.5782 = ARCSIN OF X
ENTER SINE VALUE, PLEASE
? (BRK)←
>SYS (RET)
```

This command, SYS, causes an exit from BASIC and a return to TEL.

```
!OFF (RET)
```

## DIRECT STATEMENT EXECUTION AND DESK-CALCULATOR MODE

Direct statements are BASIC statements entered without a step number either in editing or execution mode. They can be executed either with or without a current program – the latter being called "desk-calculator mode". In normal execution mode, i.e., with a current program, direct statements are used for on-line debugging, or program verification.

Direct statements are recognized as such by the subsystem and are executed (if possible) immediately. The most commonly used forms of direct statements in editing or desk-calculator mode are PRINT statements containing an arithmetic expression, and one or more LET statements followed by a PRINT statement. For on-line debugging, GOTO, LET, and PRINT are commonly used, singly or in combination, referring to variables and statements in the current program.

Most BASIC statements can be issued as direct statements; the few exceptions, e.g., FOR, NEXT, are statements that cannot be expected to "execute" by themselves in any meaningful way.

A further enhancement of the direct-statement execution capability is the EXECUTE (or EXE) command, which is provided specifically for convenience in on-line debugging and verification. The command format is

EXE[CUTE][$s_1$][-$s_2$]

where $s_1$ and $s_2$ refer to step numbers in the current program.

In either editing or execution mode, a step number reference causes one statement or a range of statements of the current program to be executed. Note that if a range is specified, the last statement in the range, $s_2$, is not executed. (You can achieve the same effect by combinations of direct statements, but the EXECUTE command is significantly faster and more convenient.)

Example 22. Use of Direct Statements – "Desk-Calculator Mode"

```
    :
    :
!BASIC (RET)
>PRINT DEG(ASN(.5)) (RET)
```

The user calls BASIC and immediately enters a direct statement, i.e., one with no line number. Note that (1) there is no current program, and (2) the user doesn't have to be in execution mode.

```
 30.0000
>PRINT DEG(ASN(1.1)) (RET)
```

Now he tries one that should result in an error comment.

```
    ASN-ACS ARG ERROR
>LET X=SQR(2)/2 (RET)
```

Then he tries a sequence of two statements with a common variable.

```
>PRINT DEG(ASN(X)) (RET)
 45.0000
>SYS (RET)
!OFF (RET)

-accounting summary-
```

**ABBREVIATIONS OF BASIC COMMAND VERBS**

All of the BASIC command verbs may be shortened to the first three letters, i.e., CLE(ar), DEL(ete), LOA(d), etc. In addition, you may use the following short forms of the SAVE command: SAV N for SAVE ON, and SAV VER for SAVE OVER.

Another direct-statement execution example follows, illustrating EXECUTE command usage.

Example 23. Using the EXECUTE Command

```
!BASIC (RET)
>LOAD VPRIM (RET)
>1 PRINT 'P='P;'I='I;'Q='Q;'S='S (RET)
```

The user inserts a statement that facilitates inspection of variable values as he executes selected portions of it below.

```
>LIST (RET)
1 PRINT 'P='P;'I='I;'Q='Q;'S='S
10 REM GENERATE PRIME NUMBERS (>3) WITHIN USER-SET LIMITS
20 PRINT "ENTER LOWER BOUND FOR PRIMES"
30 INPUT L
40 PRINT "ENTER UPPER BOUND FOR PRIMES"
50 INPUT U
60 P=1
70 P=P+4,S=0
80 FOR I=5 TO SQR(P)+1 STEP 2
90 Q=INT (P/I)
100 IF Q*I=P THEN 150
110 NEXT I
120 IF P<L THEN 150
130 IF P>U THEN 180
140 PRINT P''TAB(0)
150 IF S=1 THEN 70
160 S=1, P=P+2
170 GOTO 80
180 END
>RUN (RET)
13:02    NOV 18 VPRIM...
P=0 I=0 Q=0 S=0

ENTER LOWER BOUND FOR PRIMES
?17 (RET)
ENTER UPPER BOUND FOR PRIMES
?17 (RET)
 17
     180 HALT
>EXE 1 (RET)
P= 19   I= 5 Q= 3 S= 1
>EXE 70-120 (RET)

     120 -EXEC- HALT
>EXE 1 (RET)
P= 23   I= 5 Q= 4 S= 0
>LET U=50 (RET)
```

Here the user enters a direct statement to change the upper-bound parameter.

```
>EXE 110-150 (RET)
 23
     150 -EXEC- HALT
```

```
>EXE 150-180 (RET)
 29 31 37 41 43 47
     180 -EXEC- HALT
>SYS (RET)

!OFF (RET)

-accounting summary-
```

## FORTRAN IV SUBSYSTEM (FORT4)

The Extended FORTRAN IV processor is a mathematical-language compiler that processes an extended version of the standard FORTRAN IV language. It is appropriate to the solution of medium-to-large scale computational problems, and offers full file input/output capabilities. Unlike BASIC – compile-and-execute processor – it produces savable and reusable object programs, eliminating the need to recompile frequently used programs.

A related facility, the FORTRAN Debug Package (FDP), permits on-line debugging during program development and checkout. Use of FDP is covered in Chapter 7, "Debugging User Programs". FORT4 accepts source-program input either from a previously built source file or directly from the user's terminal, a line at a time. Normally, the former method is employed, the file having been built under the Edit subsystem.

The line-at-a-time method has an advantage for novice FORTRAN users in that "conversational" syntax-error diagnostic comments are issued immediately following input of the line to which they refer, an effective learning device. (When using this method, a source file can be preserved for subsequent modification and recompilation by means of the special compilation option SO, and an appropriate !SET command, as shown in a subsequent example.)

The standard outputs of the compilation process are the compiled object program, called the relocatable object module (ROM), and error comments. A listing of the source program may be (and generally is) requested, with the LS compilation option, described below.

### CONTROLLING THE COMPILATION PROCESS

FORT4 is called with the TEL command FORT4. The format of this command is:

$$!\text{FORT4}\ \left[\begin{matrix}\text{source}\\ \text{ME}\end{matrix}\right]\left[\begin{matrix}\text{ON}\\ \text{OVER}\end{matrix}\right]\ [\text{rom}][,\text{list}]$$

where

source specifies a RAD file containing the source program.

ME indicates source input from the user's terminal (the default assumption for this field).

rom specifies the RAD file that is to receive the object program (the default temporary file name is "$").

list specifies the destination of source-listing output: either a RAD file (fid), the terminal (ME), or central-site line printer (LP) – no default assumption is made.

Note that the ON or OVER qualifier refers only to the rom file, but one of the two must be given if either rom or list is specified. For example: FORT4 ME ON ,LISTFIL.

This command, then, serves to call the subsystem, identify the input source, and direct the compiler outputs.

Note that you can direct the compiler outputs prior to giving the FORT4 command with the !OUTPUT, !LIST, and !COMMENT commands (the latter allows a separation of source-listing and error-commentary destination). If the OUTPUT command is used, the default value for the rom given above does not apply.

After the FORT4 command, the subsystem responds with the question OPTIONS>, at which point you may enter one or more compilation options. These control the compilation process and are used primarily to request optional outputs or suppress standard outputs. The options controlling listing of outputs are:

LS - Produce source-program listing and full compilation summary.

LO - Produce source- and object-program listing, and full compilation summary.

PS - Produce partial instead of full compilation summary (default option).

NS - Suppress compilation summary.

Note that if none of the above are specified, a partial summary only is produced, as if the PS option were specified. The PS or NS option can be used in conjunction with LS or LO, or NS alone may be specified (no listing, no summary). If you use the LS or LO option, a destination for this output must be assigned, as discussed above.

There are many other options, mostly having to do with the nature of the source input and the object output. These are described in the UTS/TS Reference Manual, Publication 90 09 07, and the FORTRAN IV/OPS Reference Manual, Publication 90 11 43.

If you enter source lines directly from the terminal (i.e., FORT4 ME ...), you may want to use the SO option, which requests that the source program be reproduced as an output; in this case you must assign this output (the M:SO DCB) to a file with a !SET command (see Chapter 11, "DCB Assignments"). Several examples are given below.

Following a successful compilation, link-loading and execution of the resulting object program can be requested with the !RUN command. This command, as well as the related !LINK and !START commands are described in Chapter 6, "Loading and Executing Object Programs". Simple uses of !RUN are shown in the following examples.

In the following example, the user employs Edit to create a file, INPUT, containing the source program. Note that the source lines contain a tab character: for FORT 4, only one tab per line is accepted, and its value is fixed by the compiler as column 7 (regardless of specified setting). The example program computes the length of a three-dimensional vector, D, for input values of X, Y, and Z. Execution-time input to the program is initially from file DATA; input is terminated by a zero value for X. Output is initially directed to the terminal.

The user then decides to execute again with new X, Y, and Z values from the terminal. Accordingly, he changes the DCB assignments for FORTRAN I/O units 5 and 6 so that data input is from the terminal and program output goes to the RAD file VALUES. To examine this output, he issues a !COPY command to copy this file to the terminal.

Example 24. Compiling and Executing FORTRAN Input from a File

```
    :
!TABS7 (RET)

     The user sets a tab value of 7, so that he can see the tab effect as he builds the file.

!BUILD INPUT (RET)
  1.000 (TAB)       WRITE (6,100) (RET)
  2.000 10 (TAB)    READ (5,200) X,Y,Z (RET)
  3.000 (TAB)       IF (X) 20,50,20 (RET)
  4.000 20 (TAB)    D = SQRT(X**2+Y**2+Z**2) (RET)
  5.000 (TAB)       WRITE (6,300)X,Y,Z,D (RET)
  6.000 (TAB)       GO TO 10 (RET)
  7.000 50 (TAB)    STOP (RET)
  8.000 100 (TAB)   FORMAT (7X,1HX,11X,1HY,11X,1HZ,11X,1HD) (RET)
  9.000 200 (TAB)   FORMAT (3E) (RET)
 10.000 300 (TAB)   FORMAT (4(1X,E11.3)) (RET)
 11.000 (TAB)       END (RET)
 12.000 (RET)

     He builds a file of source input, named INPUT.
```

```
!BUILD DATA (RET)
   1.000 1.0,2.0,3.0 (RET)
   2.000 1.0,1.0,1.0 (RET)
   3.000 0.0 (RET)
   4.000  (RET)
```

He builds a program-data file.

```
!COMMENT ON ME (RET)
```

and requests error commentary at the terminal.

```
!FORT4 INPUT ON BIN (RET)
```

He asks for a compilation of INPUT, with ROM output on BIN.

```
OPTIONS> (RET)
```

and accepts the default option, partial summary.

```
 1:        WRITE (6,100)
11:        END


HIGHEST ERROR SEVERITY:  0 (NO ERRORS)

                     DEC       HEX
                   WORDS     WORDS

 GENERATED CODE:      56     00038
      CONSTANTS:       0     00000
LOCAL VARIABLES:       4     00004
          TEMPS:       2     00002

  TOTAL PROGRAM:      62     0003E
```

The partial summary prints at the terminal.

```
!SET F:5 /DATA;IN (RET)
```

The user assigns the file DATA to the F:5 DCB, and defines it as an input file. (This is a "file assignment".)

```
!SET F:6 UC (RET)
```

He assigns the user's terminal to the F:6 DCB, via the operational label UC. (This is a "device assignment".)

```
!RUN (RET)
```

He requests a run, i.e., link-load and execution, of the program. Since the RUN command assumes as its input the results of the latest compilation or assembly if no input file is specified, he does not need to specify BIN.

```
    LINKING $
P1 ASSOCIATED
```

The loader's messages print.

```
      X             Y             Z             D
   .100E 01      .200E 01      .300E 01      .374E 01
   .100E 0       .100E 01      .100E 01      .173E 01
*STOP* 0
```

Then the program's output appears, and a normal program-halt is indicated.

```
!SET F:5 UC (RET)
```

The user resets the input unit to the terminal

```
!SET F:6 /OUTPUT;OUT (RET)
```

and the output unit to a file, named OUTPUT.

```
!RUN BIN (RET)
```

He reruns, this time specifying the ROM name (not actually required in this case, as explained above).

```
     LINKING BIN
P1 ASSOCIATED
?4.4,5.5,6.6 (RET)
?0.0 (RET)
```

The program-input-request prompt is given, and the user enters a set of values and a zero value to indicate end of data.

```
*STOP* 0
```

A normal program halt is indicated.

```
!COPY OUTPUT TO ME (RET)
```

The user requests a copy of the output file to the terminal.

```
      X             Y             Z             D
   .440E 01      .550E 01      .660E 01      .965E 01

!      .
       .
       .
```

The next example shows a very simple program entered directly from the terminal. The user requests the source program to be reproduced (SO), and uses a !SET command to assign a source-output file, SOURCE. He also uses the !SET command to assign FORTRAN unit 6, the program output, to the terminal.

Example 25. Submitting Terminal Input for FORTRAN Compilation

```
      .
      .
      .
!SET M:SO     /SOURCE (RET)
```

The user sets the DCB for source output produced by the compiler, M:SO, to the file SOURCE, Here, since he does not specify a file function (e.g., IN, OUT), OUT is assumed by default.

```
!FORT4 ME (RET)
```

He asks for a compilation of direct terminal input.

```
OPTIONS>  NS,SO (RET)
```

He suppresses the partial summary (NS), and requests source output (SO).

```
>C THIS EXAMPLE ILLUSTRATES HOW SOURCE LINES ARE ENTERED (RET)
>C DIRECTLY FROM THE TERMINAL, AND HOW A LINE IS CONTINUED. (RET)
> (TAB)    WRITE (6,100) (RET)
>100 (TAB) FORMAT (1X,: (RET)
>      C25HTHIS IS A CONTINUED LINE.) (RET)
> (TAB)    END (RET)
```

FORTRAN prompts for source input with >. Note that the C in the fifth line is in column 6, the rest following a tab starting in column 7. (The user assumes that the tab setting of 7 from the previous example is still in effect.)

```
!SET F:6 UC (RET)
```

He sets the output unit to the terminal (operational label UC),

```
!RUN (RET)
```

and requests a run.

```
     LINKING $
P1 ASSOCIATED
 THIS IS A CONTINUED LINE.
 *STOP* 0
```

```
!COPY SOURCE TO ME (RET)
```

He displays the reproduced source file.

```
C THIS EXAMPLE ILLUSTRATES HOW SOURCE LINES ARE ENTERED
C DIRECTLY FROM THE TERMINAL, AND HOW A LINE IS CONTINUED.
      WRITE (6,100)
100   FORMAT (1X,
     C25HTHIS IS A CONTINUED LINE.)
      END
```

Note that the "to be continued" marker (:) in the fourth line has been stripped off by the subsystem.

```
!OFF (RET)

-accounting summary-
```

## META-SYMBOL SUBSYSTEM (META)

Meta-Symbol is a macro-assembler that processes an assembly language, Symbol, (which is a symbolic representation of the machine language) and macro-procedure language, Meta-Symbol (which is a powerful logical extension of the assembly language). Assembly language is the "lowest level" language normally used for programming.

The advantage of Meta-Symbol programming is the maximum speed and efficiency that is possible in the resultant object programs. Its disadvantage is that it is more time-consuming to learn and to use than "higher-level" languages such as FORTRAN.

Also available is an extensive and sophisticated debugging subsystem, Delta, designed specifically – though not exclusively – for degubbing Meta-Symbol object programs. Its use is covered in Chapter 7 and in Example 43.

The META subsystem is called by the TEL command META. There are many examples throughout the following chapters that illustrate the use of META:

Example 34 shows an assembly and execution.

Example 39 shows use of OUTPUT, LIST, and COMMENT commands.

Example 40 shows use of SET commands before calling META.

Example 41 shows discontinuation and resumption of output while assembling with META.

These examples all illustrate the use of META to assemble from a source file. META can also be used to assemble source lines directly from the terminal, as shown in the following example. Unlike the FORT4 subsystem however, diagnostics are not produced until after the END statement is received. ("Diagnostic" is a general term for the warning and error commentaries resulting from the error checking performed by the assembler.)

The format of the META command is

$$\text{META}\left[\begin{matrix}\text{source}\\ \text{ME}\end{matrix}\right]\left[\begin{matrix}\text{ON}\\ \text{OVER}\end{matrix}\ [\text{rom}][,\text{list}]\right]$$

where

source specifies a RAD file containing the source program.

ME indicates source input from the terminal (the default assumption for this field).

rom specifies the RAD file that is to receive the object program (the default temporary-file name is "$").

list specifies the destination of the source-program listing: either a RAD file (fid), the terminal (ME), or the line printer (LP) – ME is assumed by default but no listing output is produced (unless !LIST is issued during a subsequent interruption of the assembly).

Note that the ON or OVER qualifier refers only to the rom file, but one of the two must be given if either rom or list is specified.

The effect of the META command variable field is to assign the M:SI (source input) DCB, the M:GO ("go", or object output) DCB, and the M:LO (listing output) DCB to the source, rom, and list specifications, or to their defaults. Note also that if these DCBs have been assigned previously in the session, either by an !OUTPUT, !LIST, !FORT4, or prior META command, the corresponding default values given above do not apply. (The effect of the !COMMENT command is to explicitly assign the M:DO (diagnostic output) DCB; i.e., to specify a destination for diagnostics separate from the source listing, if any.)

After the META command is given, the subsystem asks for assembly options: WITH>. A description of these options for on-line usage can be found in the UTS/TS Reference Manual, Publication 90 09 07, Chapter 4. The only options we need mention here are SO (source output), which functions exactly as in a FORTRAN compilation – shown in the previous section – and SD (symbolic-debugging), which is covered in Chapter 7. Note that source-listing output is implicitly requested or suppressed by the list parameter in the META command, unless a LIST command is given before the META command.

Note also that the format of the assembler source listing is not very suitable for display at the terminal, and is best directed to the line printer (LP), or omitted. Comments go to the terminal (by default) in either case.

Example 26. Using META to Assemble Terminal Input

```
    :
!META ME (RET)
```

The user asks for an assembly of terminal input, with no source listing.

```
WITH (RET)
```

He doesn't request any assembly options and, below, begins to type in the source lines following META's prompt character (>). (A tab setting and tab characters could be used to achieve the desired starting columns as shown in later examples.)

```
>*THIS EXAMPLE ILLUSTRATES DIRECT INPUT FROM TERMINAL. (RET)
>*IT ALSO SHOWS HOW TO CONTINUE A LINE. (RET)
>          SYSTEM BPM (RET)
>          SYSTEM SIG7 (RET)
>          REF M:UC (RET)
>START     M:WRITE M:UC,(BUF,M;   (CONTINUED LINE) (RET)
```

```
>         ES),(SIZE,26)    (CONTINUATION) (RET)
>         M:EXIT (RET)
>MES      TEXT "EXAMPLE OF CONT",; (RET)
>         "INUED LINE." (RET)
>         END START (RET)

 *     ERROR SEVERITY LEVEL:  0
 * NO ERROR LINES

!RUN (NP)
```

Here the library-search option NP is used to suppress association of the default public library, P1, by the loader, as it is only required for FORTRAN programs.

```
      LINKING $
EXAMPLE OF CONTINUED LINE.
```

The program executes, printing its output, and control returns to TEL.

```
!OFF
```

# 6. LOADING AND EXECUTION OBJECT PROGRAMS

## LINK SUBSYSTEM

The LINK subsystem consists of a one-pass link-editor/loader, or linking loader. The essential functions of the linking loader are to combine a number of separate program elements into a single executable entity called a load module (LM), and to load it for execution. You can request these two functions together with the RUN command, or separately with the LINK and (e.g.) START commands, respectively. In its linking operation, LINK merges internal symbol tables of several relocatable object modules (ROMs) presented to it and searches one or more subroutine libraries to satisfy external references, where required. It makes full use of the UTS Sigma 7 hardware memory-mapping, allocating virtual data space as needed for association of a public core library such as the FORTRAN P0 or P1 libraries.

The linking loader must be used both to link-edit and load one ROM, i.e., the output of one compilation or assembly, along with any necessary system-supplied service procedures and library subroutines, or to link two or more ROMs from separate compilations or assemblies, with their combined system-related references, into one load module.

### RUN COMMAND

The TEL command RUN requests linking, loading, and executing of one or more ROMs. Forms of the RUN command are as follows:

1. RUN (or RUN $)

   These forms simply request that the ROM created by the last compilation or assembly be linked, loaded, and executed. The two forms shown are synonymous. (Input is taken from the file last assigned to the M:GO DCB; LM output is placed on a special temporary file.)

2. RUN rom

   The ROM stored on the RAD file specified by rom is to be linked, loaded, and executed. (LM output is placed on a special temporary file.)

3. RUN [rom] $\left\{\begin{matrix}\text{ON} \\ \text{OVER}\end{matrix}\right\}$ lmn

   ROM input may be specified as in 2, above, but the LM output may also be directed to the file named lmn.

In each case, the LM output is available for a subsequent reexecution via the START command. In all three cases, the public core library P1 is implicitly associated with the object program to satisfy any external references, if possible.

The general formats of the RUN and LINK commands are identical; thus the more complicated form shown for LINK in the next section is equally applicable to RUN, and vice versa.

Example 27. Using the RUN Command

```
:
:
!EDIT (RET)

        The user calls Edit.

EDIT HERE

!TA M (RET)

        He uses the Edit Tabs command (TA) and specifies the Meta-Symbol (M) tab setting (10, 19,
        and 37). (Other sets are available; see Edit manual.) He then builds a source file, INPUT.
```

```
*BUILD INPUT

 1.000 (TAB)          SYSTEM (TAB)   SIG7 (RET)
 2.000 (TAB)          SYSTEM (TAB)   BPM (RET)
 3.000 (TAB)          REF (TAB)      M:UC (RET)
 4.000 BEGIN (TAB)    M:WRITE (TAB)  M:UC,(BUF,MESS),(SIZE,43) (RET)
 5.000 (TAB)          M:EXIT (RET)
 6.000 MESS (TAB)     TEXT (TAB)     'THIS MESSAGE SHOULD',; (RET)
 7.000 o(TAB)         'PRINT AT THE TERMINAL.' (RET)
 8.000 (TAB)          DATA (TAB)     X'15000000' (TAB)     NEW LINE CHARACTER (RET)
 9.000 (TAB)          END (TAB)      BEGIN (RET)
10.000 (RET)
*END (RET)
!META INPUT ON BIN (RET)
```

META is called to assemble source file INPUT with ROM output going to file BIN and no assembly listing produced.

```
WITH> (RET)
```

No assembly options are desired.

```
!RUN (RET)
```

A run is requested from the last compilation/assembly output, i.e., BIN in this case.

```
  LINKING $
```

The system acknowledges the LINK function (the LINK subsystem is implicitly called).

```
DEFAULT CORE LIBRARY IS NOT NEEDED
```

See Example 28 for meaning of this message.

```
THIS MESSAGE SHOULD PRINT AT THE TERMINAL
```

The program's output is printed.

```
!OFF (RET)
```

The user logs off. The temporary file containing the load-module output of RUN is now lost. (The ROM file BIN is permanent, however.)

## LINK COMMAND

The LINK command requests link-editing, as does RUN, but does not cause loading and execution of the resulting load module. A more complex variable-field format than those shown in the previous section for RUN is given here:

$$\text{LINK}\left[\text{rom}_1, \text{rom}_2, \ldots, \text{rom}_n\right]\left[\begin{matrix}\text{ON}\\ \text{OVER}\end{matrix}\ \text{lmn}\right]\left[;\text{lid}_1\left[,\text{lid}_2, \ldots, \text{lid}_n\right]\right]$$

where

$\text{rom}_i$ specifies a RAD file containing a ROM.

lmn specifies a RAD file for the LM output.

$\text{lid}_i$ specifies a RAD file containing a user's subroutine-library.

In the format above, the several ROMs specified will be linked into one LM, with user's libraries $\text{lid}_1$ through $\text{lid}_n$ searched (prior to any public or system libraries) to satisfy external references, and the result placed ON or OVER lmn if specified.

In addition to the above, a parenthesized library-search code may be given. It is conventionally placed after the command verb, as in LINK (code) . . . . These codes request or suppress searching of system-supplied libraries, and are listed in Appendix D. Also, internal symbol tables for several ROMs may be merged or selectively deleted in the load module (see the UTS/TS Reference Manual, Publication 90 09 07, for these formats).

Example 28. Using the LINK Command

```
:
:
!LINK BIN ON LOAD (RET)

        A link-edit of the ROM on file BIN is requested, with the resultant LM placed on LOAD.

  LINKING BIN

        The system responds to the LINK command.

DEFAULT CORE LIBRARY IS NOT NEEDED

        The absence of a library search code (see Appendix D) in the LINK command causes this message if
        the default library (P1) is not required. The specification of search code NP will suppress associa-
        tion of P1 and also suppress this message.

!OFF (RET)

        Since the user does not want to execute the program at this time, he logs off. Files BIN and LOAD
        are permanent and can be accessed in subsequent sessions.


- accounting summary -
```

**START COMMAND**

The !START command can be used to load and execute a load module produced by a prior LINK command, or to reexecute an LM already RUN (or STARTed). Three forms are applicable:

1. START

   This form causes the last LM produced, either via a LINK or RUN, to be loaded and executed. Note that the prior LINK or RUN must have been given during the current terminal session; the load-module file may have been explicitly named (lmn), or named by default ($).

2. START $

   This form causes the last LM produced on the temporary file $ to be loaded and executed; the load-module file must have been named by default ($).

3. START lmn

   This form causes the load module contained on the specified file to be loaded and executed. The LM may have been the result of either a LINK or RUN operation.

   See Chapter 8, User Programs, for an alternate way of loading and executing user-developed object programs:

Example 29. Using the START Command

```
:
:
!START LOAD (RET)

        The load module LOAD created in Example 28 is loaded into core and execution begun.
```

```
THIS MESSAGE SHOULD PRINT AT THE TERMINAL
        The program's output is printed.
!OFF (RET)

- accounting summary -
```

# 7. DEBUGGING USER PROGRAMS

Two dynamic debugging facilities are available for on-line use under UTS:

- Delta Subsystem for debugging Meta-Symbol programs.
- FORTRAN Debug Package (FDP) for debugging Extended FORTRAN IV programs.

"Debugging" is a general term for program-error detection and correction; dynamic debugging implies that the debugging process is carried out during the execution of an assembled or compiled program (as opposed to "desk checking"). Both Delta and FDP allow symbolic, i.e., source-program level references to elements of the object program.

## ASSEMBLY LANGUAGE DEBUGGING (DELTA)

The Delta subsystem provides conversational debugging capability for checkout and modification of Meta-Symbol programs at execution time. Delta allows full use of symbolic references to elements of the object program, and enables you to

- Control program execution, i.e., stop and restart it at any point, by means of breakpoints that you may insert in the program at your discretion. These breakpoints may be unconditional ("always stop"), conditional ("stop under certain circumstances"), or based on changes in data values.
- Examine, modify, and insert various program elements: instructions, constants, variable values, and encoded data of all types and formats. This can be done both prior to execution and during any halt in execution (e.g., due to a breakpoint).
- Trace continuous program execution by requesting a repeated display of specified sets of related information: register contents, switches, data values, etc., at specified points in the program.
- Search programs and data for specific elements and values.

Please refer to the Delta (for UTS)/Reference Manual, Publication 90 16 34, for a comprehensive description and explanation of the commands available under Delta.

### EXECUTING IN DEBUG MODE

To initiate execution of a program in debug mode, you must append the clause UNDER DELTA to your RUN or START command. Also, you must specify the SD (symbolic debugging) assembly option in response to WITH> to preserve the internal symbol table(s) of your program, if you want to refer to internal symbols with Delta commands – the normal case. (Internal symbols are those whose point of definition and points of use are entirely within one ROM.)

Note that the global (or external) symbols of your program are always available for reference (see the following section).

When UNDER DELTA has been specified, the Delta subsystem intervenes between program loading and initial execution. At this point you can issue debugging commands to examine or modify locations, insert breakpoints, start execution at specified point, etc. Delta also assumes control at any halt in execution.

The following example illustrates the usual method of using Delta in the debug mode of execution. A simple program is assembled with the SD option, run UNDER DELTA, and patched to create a missing M:EXIT statement. Note that before you refer to internal symbols you must tell Delta the name(s) of the desired symbol table(s) by ROM-file name (even though only one ROM may have been assembled).

To leave Delta and return to TEL at the end of execution, you issue a $Y^C$ control combination.

Example 30. Assembling and Loading in the Debug Mode.

```
:
:
!META ME ON BFILE (RET)
WITH> SD (RET)
```

The user calls META to assemble statements from the terminal. He uses the SD option to cause an internal symbol table to be produced.

```
>        SYSTEM SIG7 (RET)
>        SYSTEM BPM (RET)
>        REF M:UC (RET)
>BEGIN   M:WRITE M:UC,(BUF,MES),(SIZE,9)
>MES     TEXT  "GREETINGS" (RET)
>        END BEGIN (RET)

*    ERROR SEVERITY LEVEL:  0
* NO ERROR LINES
```

Although there are no assembly errors, the user notes that he forgot to include an M:EXIT in the program, and decides to make this correction with patches.

```
!RUN BFILE UNDER DELTA (NP) (RET)
    LINKING BFILE
```

He links and loads UNDER DELTA, suppressing loading of the default library with the code NP.

```
DELTA HERE

  "ring"
```

Delta identifies itself and prompts with a ring of the console bell.

```
BFILE;S (RET)
```

The user selects the internal symbol table associated with ROM BFILE.

```
BEGIN/  CAL1,1  MES+.3 (RET)
```

This command opens the cell at location BEGIN and displays its contents.

```
BEGIN(X/  .410C004 (TAB)
```

A command is now entered to cause the contents of BEGIN to be displayed in hexadecimal format. The user terminates the command with the tab-key sequence, CONTROL and I, which causes the cell addressed by this command (location C004) to be opened and displayed.

```
MES+.3/   .11009000 (LF)
MES+.4/   .30000000 (LF)
MES+.5/   .C001 (LF)
MES+.6/   .9 (RET)
```

The contents of the function parameter table (FPT) referenced by the M:WRITE (at location BEGIN) is displayed. Note that location C004 is shown symbolically as MES+.3. A line feed causes the next cell to be opened and displayed. A carriage return terminates the sequence. Note that the hexadecimal conversion format is maintained over the (TAB) and (LF).

```
BEGIN\ B MES+20, (RET)
```

The user issues a command to open the cell at BEGIN and enters a branch to location MES+20, a patch area he has chosen that is well beyond the main program and the FPT displayed above.

```
MES+20\ CAL1,1 MES+3 (LF)
```

He enters symbolic code for the M:WRITE instruction (as originally contained in BEGIN, displayed above). The line feed causes the next cell to be opened for modification.

```
MES+.15/   .0 CAL1,9 1 (RET)
```

The next location prints with a hexadecimal displacement. He then enters symbolic code corresponding to an M:EXIT. He has now entered all his patches.

```
BEGIN;G (RET)
```

He initiates execution.

```
GREETINGS


ABORT CODE = 0 SUBCODE = 0
```

The output message prints, and Delta reports execution of the M:EXIT.

```
Y^C ←
```

The user interrupts with $Y^C$ and then logs off.

```
!OFF

FINI
```

The system informs him that Delta was terminated.

```
- accounting summary -
```

## USING DELTA IN NONDEBUG MODE

Delta may also be called for use when you have not initially executed in debug mode, i.e., you did not specify UNDER DELTA in your RUN or START command. The next example illustrates this type of usage.

Note that only the global symbol table is available, and that the user's first Delta command must be ;S to cause this symbol table – associated with the load module as a whole – to be loaded. Otherwise, no symbols will be available for reference. (If rom;S is specified, as was possible in the preceding example, the global table is loaded implicitly.)

Example 31. Calling Delta after Assembling and Executing in Nondebug Mode

```
      :
      :
```

The user wants to assemble lines from the terminal and to default all options.

```
!META ME (RET)
WITH> (RET)
>       SYSTEM SIG7 (RET)
>       SYSTEM BPM (RET)
>       DEF START (RET)
>       REF M:UC (RET)
>START  LI,3 55 (RET)
>       M:STIMER (SEC,5),XY (RET)
>AB     LI,4 0 (RET)
>       STW,4 X (RET)
```

```
>        LW,4 X (RET)
>        CI,4 0 (RET)
>        BE $-2 (RET)
>        M:STIMER (SEC,5),XY (RET)
>        M:WRITE M:UC,(BUF,MES),(SIZE,14) (RET)
>        BDR,3 AB (RET)
>        M:EXIT (RET)
>XY      LI,4 1 (RET)
>        STW,4 X (RET)
>        M:TRTN (RET)
>X       RES 1 (RET)
>MES     TEXT  "5-SEC INTERVAL" (RET)
>        END START (RET)

*    ERROR SEVERITY LEVEL: 0
* NO ERROR LINES

!RUN (NP) (RET)

     LINKING $
```

He initiates loading and execution of the program.

```
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
```

The program output begins to print.

```
Y^C←
```

The user notes that the program is looping more than was intended, and notices that an error was made in the first statement (he typed 55 instead of 5), and decides to interrupt with $Y^C$ and call Delta to enter a patch.

```
!DELTA (RET)
```

Control goes to TEL. The user calls the Delta subsystem.

```
DELTA HERE

  "ring"
```

Delta identifies itself and prompts with a bell.

```
;S (RET)
```

The user loads the global symbol table. The only symbol that can be referred to is START which is the only DEF in the program.

```
START(X/ .22300037   .22300005 (RET)
```

He enters a Delta command to display the contents of START in hexadecimal format, and changes this value to the hexadecimal equivalent of LI,3 5.

```
START;G (RET)
```

He directs execution to the beginning of the program (location START).

```
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
5-SEC INTERVAL
```

This time the program executes as was intended.

```
ABORT CODE = 0 SUBCODE = 0
```

Delta reports execution of the M:EXIT, i.e., normal termination.

```
Y^C←
```

CONTROL and Y interrupts Delta and returns control to TEL.

```
!OFF (RET)
```

The user logs off.

```
FINI
```

The system informs him that Delta was terminated.

```
- accounting summary -
```

## FORTRAN DEBUGGING (FDP)

The FORTRAN Debug Package provides a powerful conversational facility for convenient and rapid checkout of FORTRAN IV programs. The debugging features provided are dynamically controllable from the terminal at program-execution time, and include the following:

- Statement stepping.
- Conditional breakpoints.
- Data-change breakpoints.
- Execution-flow tracing and event-history recording.
- Display and modification of scalar and array-element values.
- Branching.
- Program restart.
- Statement skipping and deletion.
- Automatic calling-argument display.

You may refer to variables by name and to statements by source-line number or statement label. These references may be further qualified by subprogram name.

The FDP facility consists of a sublibrary of run-time subroutines (a portion of public library P0), plus the necessary symbol tables and in-line coding generated by the compiler when debug-mode is requested. (FDP can be used only when debug-mode compilation has been performed.) Programs compiled in debug mode should not be used indiscriminately, as they require approximately 2.5 times the amount of memory required for nondebug runs and may even double normal execution times.

In order to use FDP, you must do the following:

1. Specify the DEBUG compilation option when FORT4 prompts for options.

2. Issue the SET command shown below, prior to issuing the LINK or RUN command (but not prior to FORT4), if M:SI has been previously set to a file either explicitly or implicitly (i.e., for a compilation or assembly):

   !SET M:SI UC

3. Specify in the RUN or LINK command either one of the library-search options (FDP or P0), or the clause UNDER FDP (the three forms are synonymous).

The two examples given here illustrate, in addition to TEL command usage, some of the more commonly used FDP commands. See the FDP/Reference Manual, Publication 90 16 77, for a complete description of the FDP commands, and a full explanation of their use.

In the following example, the user compiles file INPUT, created in a previous example, in debug mode. Values for X, Y, and Z are read from file DATA (also created in the prior example). The ON debugging command causes values of D, X, Y, and Z to be displayed whenever D is computed.

Example 32. Use of FDP ON and PRINT Commands

```
    :
!FORT4 INPUT ON ,ME (RET)
```

The user compiles file INPUT and directs the listing and compilation summary to the terminal.

```
OPTIONS >DEBUG,LS (RET)
```

Note the specification of DEBUG as an option.

```
      1:       WRITE (6,100)
      2: 10    READ (5,200) X,Y,Z
      3:       IF (X) 20,50,20
      4: 20    D = SQRT(X**2+Y**2+Z**2)
      5:       WRITE (6,300) X,Y,Z,D
      6:       GO TO 10
      7: 50    STOP
      8: 100   FORMAT (7X,1HX,11X,1HY,11X,1HZ,11X,1HD)
      9: 200   FORMAT (3E11.3)
     10: 300   FORMAT (4(1X,E11.3))
     11:       END

                          HEX     DEC                              HEX     DEC
NAME   TYPE CLASS          LOC    WORDS      NAME   TYPE CLASS     LOC    WORDS

D        R SCALR  00003 V     1      SQRT     R SPROG  INTRIN
X        R SCALR  00000 V     1      Y        R SCALR  00001 V       1
Z        R SCALR  00002 V     1

       HEX                HEX               HEX                HEX
LABEL  LOC        LABEL   LOC      LABEL    LOC       LABEL    LOC

   10 00008          20  00017        50 00036         100   0003A
  200 00043         300  00046

LOCAL VARIABLES (4 WORDS):

   00000 X          00001 Y          00002 Z          00003 D
```

```
BLANK COMMON (0 WORDS)

INTRINSIC SUBPROGRAMS USED:

   SQRT

EXTERNAL SUBPROGRAMS REQUIRED:

   F:UF         F:108        M:DO         M:OC         M:SI         9BCDREAD
   9BCDWRIT     9DBDCKIN     9DBFHGO      9DBFHIF      9DBINIT      9DBSCKIN
   9ENDIOL      9IODATA      9SQRT        9STOP

HIGHEST ERROR SEVERITY:  0 (NO ERRORS)

                  DEC          HEX
                 WORDS        WORDS

 GENERATED CODE:   120        00078
      CONSTANTS:     0        00000
LOCAL VARIABLES:     4        00004
          TEMPS:     4        00004

  TOTAL PROGRAM:   128        00080
```

The program listing and compilation summary is printed.

```
!SET F:6  /VECTORS;OUT (RET)
```

The user directs the program output to file VECTORS.

```
!SET F:5  /DATA;IN (RET)
```

Program input will be read from file DATA.

```
!SET M:SI UC (RET)
```

This command is necessary because the debug routines read directives from the terminal. (M:SI was set to DC/INPUT by the FORT4 command above.)

```
!RUN (FDP) (RET)
```

The user loads and executes in the debug mode. Alternatively he could have specified:

```
!RUN UNDER FDP     or
!RUN (P0)
```

```
   LINKING $
```

The loader's message prints.

```
@ON D;PRINT X,Y,Z (RET)
```

FDP prompts with @. The user enters commands to cause the value for D to be displayed each time it is stored into, and at the same time to display values for X, Y, and Z.

```
@GO (RET)
```

The user does not want to enter any more debug commands at this point and issues a GO command to start execution.

```
/4(20S): D=3.74166     X=1.00000
Y=2.00000
Z=3.00000
```

```
 4(20S): D=1.73205      X=1.00000
Y=1.00000
Z=1.00000
```

Values for D, X, Y, and Z are displayed. The slash (/) indicates main program and is followed by line number and statement number (if present) in parentheses.

```
*STOP* 0
```

This message is produced by the library subroutine STOP.

```
7(50S): RDY TO STOP
```

This message is produced by the debugger.

```
@QUIT (RET)
```

The QUIT command causes return to the monitor.

```
!OFF (RET)


- accounting summary -
```

In the next example, the user enters a FORTRAN source program from the terminal without initializing variables, setting loop control, and providing for I/O. He runs in the debug mode and issues FDP commands to provide the omitted functions.

This program generates a Fibonacci sequence, in which the value of any number (beyond the second) in the sequence is equal to the sum of the values of the two preceding numbers, e.g., 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . . .

Example 33. Further Uses of FDP Commands

```
    :

!FORT 4 ME ON ,LP (RET)

OPTIONS >DEBUG (RET)

>10      I=I+J (RET)
>20      J=I+J (RET)
>30      GO TO 10 (RET)
>40      END (RET)

HIGHEST ERROR SEVERITY: 0 (NO ERRORS)

!RUN UNDER FDP (RET)
```

This command causes the user's program to be loaded and executed, with public library FDP associated.

```
      LINKING $

@I=0 (RET)
@J=1 (RET)
```

FDP prompts with @. The user initializes I and J.

```
@ON I (RET)
@ON J (RET)
```

These ON commands will cause values for I and J to be displayed when these variables are stored into.

```
@STOP AT 3#5 (RET)
```

This command causes execution to halt the fifth time that statement 3 (the GO TO statement) is encountered.

```
@GO (RET)
```

Execution is now begun with the above commands in effect.

```
/1(10S): I=1
 2(20S): J=2
 1(10S): I=3
 2(20S): J=5
 1(10S): I=8
 2(20S): J=13
 1(10S): I=21
 2(20S): J=34
 1(10S): I=55
 2(20S): J=89
```

Values for I and J are displayed. The slash (/) indicates main program and is followed by line number and statement number (in parentheses).

```
3(30S):
```

The program halts the fifth time that statement 3 is reached.

```
@KILL (RET)
```

FDP prompts for a command. This KILL cancels all previous FDP commands.

```
@STOP ON I>500 (RET)
```

A conditional stop, or breakpoint, is set.

```
@AT 3; PRINT I,J (RET)
```

These commands will cause values of I and J to print each time statement 3 is reached.

```
@RESTART (RET)
```

This specifies restart of program from beginning.

```
@GO (RET)
```

The user resumes execution.

```
/3(30S): I=144
J=233
 3(30S): I=377
J=610
```

Values for I and J are displayed each time statement 3 is reached.

```
1(10S): I=987
```

At statement 1, the value for I exceeds 500 and the program halts.

```
@KILL ON I (RET)
```

This command cancels the last ON I, effectively the last STOP command issued.

```
@STOP ON J>10000 (RET)
```

Another conditional stop is issued.

@GO ⏎

Execution is resumed at statement 3, which is where the previous stop occurred.

```
/3(30S): I-987
J=1597
 3(30S): I=2584
J=4181
 2(20S): J=10946
```

Values for I and J print until J exceeds 10000 which occurs at statement 2.

```
@PRINT I ⏎
6765
```

Since statement 3 was not reached to cause the current value for I to be displayed, the user gives a PRINT command to cause this value to print.

@QUIT ⏎

The user now leaves FDP and returns to TEL.

```
!OFF ⏎

- accounting summary -
```

# 8. EXECUTING USER PROGRAMS

An object program stored on a file in load-module form may be called by its load-module name (lmn) used as a TEL command verb. The load-module file may be stored either in your own account, someone else's account, or the system account. Thus far, the lmn-as-verb command is synonymous to the !START lmn command (except for a difference in account-number defaults). Within the !lmn command, however, you can also very conveniently make file or device assignments for three standard system DCBs: M:SI, M:GO, and M:LO.

The format of the variable field of the command is analogous to that of the FORT4 and META commands; the full format is

$$!lmn\ [input]\left[\begin{matrix}ON\\OVER\end{matrix}\ [output_1][,output_2]\right]$$

where

lmn is the fid of an LM that can take the full form:

name [. [account] [. password]]

(see below for special defaults)

input may be a fid or ME to be assigned to the input DCB M:SI.

$output_1$ may be a fid to be assigned to the output DCB M:GO.

$output_2$ may be a fid, ME, or LP to be assigned to the output DCB M:LO.

(Normal default assignments apply. That is, the M:SI and M:LO DCBs, if referenced, default to the user's terminal, and M:GO to a temporary file named $.)

The called program must, of course, directly or indirectly utilize one or more of the above-mentioned DCBs for any of these assignments to make sense.

You can imply or specify the account number under which the LM file is stored, and specify a password, as follows:

1. filename – (alone, with no period) implies the system account.
2. filename._ – implies your account (i.e., the log-on account value).
3. filename.account – specifies an account number.
4. filename.account.password – specifies an account number and password.
5. filename..password – implies your account and a password.

Note that this particular convention of default account-number values is not the standard one that applies to most fid specification in TEL and other commands, as described in Appendix B. The reason for the system-account default in particular is that installations may want to include, at system-generation time, certain user-developed "production" programs in the system account; special forms of these may then be accorded preferential RAD-storage and loading, depending upon frequency of use and programming characteristics. (META and FORT4 commands, for example, are actually special instances of !lmn commands.)

The two examples following show very simple programs, developed wholly within the example for purposes of illustration. Actual uses of the command may, of course, call a program developed some time in the past, and possibly by another programmer.

The program in Example 34 reads a Meta-Symbol source program via M:SI, and writes out any comment lines (asterisk in column 1) contained in the program, via M:LO. The input file and output device are assigned within the calling command. As a test, the user specifies as input the source program from which the called object-program was assembled.

Example 35 merely shows a simple FORTRAN IV program call by its load-module name. Since the compiler automatically provides (indirectly) program-file DCBs identified with names of the form F:n, file/device assignments cannot also be made within the !lmn command for FORTRAN object programs.

Example 34. Using Load-Module-Name as Command Verb (Meta-Symbol Program)

```
:
:
!EDIT (RET)
EDIT HERE
*TA M (RET)
```

The user sets tabs for META. Then he builds a Meta-Symbol Program to extract comments lines from Meta-Symbol source programs. (Usage of tab control not shown.)

```
*BUILD SOURCE (RET)
  1.000 *         ***UTILITY PROGRAM "EXTRACT"*** (RET)
  2.000 *  THIS ROUTINE LISTS ONLY THE COMMENTS LINES, IF ANY, FROM (RET)
  3.000 *   A META SOURCE-PROGRAM FILE.  IT ISSUES A BLANK (RET)
  4.000 *   LINE TO INDICATE ONE OR MORE CODING LINES (RET)
  5.000 *   INTERVENING BETWEEN COMMENTS.  IT READS ITS INPUT (RET)
  6.000 *   FROM M:SI, AND WRITES TO M:LO. (RET)
  7.000          SYSTEM    BPM (RET)
  8.000          SYSTEM    SIG7 (RET)
  9.000          REF       M:SI,M:LO (RET)
 10.000 *     ***INPUT BUFFER*** (RET)
 11.000 INN      RES       20 (RET)
 12.000 *     ***80 BLANKS*** (RET)
 13.000 BLANKS   EQU       $ (RET)
 14.000          DO1       20 (RET)
 15.000          TEXT      '    ' (RET)
 16.000 *  WE GIVE A TOP-OF-PAGE AT BEGINNING (AND END) (RET)
 17.000 START    M:DEVICE M:LO, (PAGE) (RET)
 18.000 *  SWITCH: "HAVE WE ISSUED A BLANK LINE ?":  0 = YES/1 = NO (RET)
 19.000          LW,4      =0
 20.000 RDNXT    M:READ    M:SI,(BUF,INN),(SIZE,80),(ABN,EXIT)
 21.000          LB,5      INN
 22.000          CI,5      "*"
 23.000          BE        PRINT
 24.000          CI,4      0
 25.000          BE        RDNXT
 26.000          M:WRITE   M:LO,(BUF,BLANKS),(SIZE,72)
 27.000 *  WE SET THE SWITCH:  "BLANK LINE ISSUED SINCE LAST
 28.000 *    COMMENT"
 29.000          LI,4      0
 30.000          B         RDNXT
 31.000 *  WE RESET THE SWITCH:  "BLANK LINE NOT ISSUED SINCE THE
 32.000 *    LAST COMMENT"
 33.000 PRINT    LI,4      1
 34.000          LW,1      M:SI+4
 35.000          SLS,1     -17
 36.000          M:WRITE   M:LO,(BUF,INN),(SIZE,*1)
 37.000          B         RDNXT
 38.000 EXIT     M:DEVICE M:LO,(PAGE)
 39.000          M:CLOSE   M:LO,(SAVE)
 40.000          M:EXIT
 41.000          END       START

!META SOURCE (RET)

WITH> (RET)

*     ERROR SEVERITY LEVEL:  0
* NO ERROR LINES

!LINK (NP) ON EXTRACT (RET)

     LINKING $
```

```
!COPY SOURCE TO SORCNC(NC)
```

The user copies the source file using the NC option to strip the carriage-return off each record; otherwise the output below would be double spaced.

```
!EXTRACT.  SORCNC ON ,ME (RET)
```

Here he calls the LM EXTRACT, with a following period to indicate "my account", and assigns the input file, SORCNC, and directs M:LO output to the terminal.

```
*        ***UTILITY PROGRAM "EXTRACT"***
*  THIS ROUTINE LISTS ONLY THE COMMENTS LINES, IF ANY, FROM
*   A META SOURCE-PROGRAM FILE.  IT ISSUES A BLANK
*   LINE TO INDICATE ONE OR MORE CODING LINES
*   INTERVENING BETWEEN COMMENTS.  IT READS ITS INPUT
*   FROM M:SI, AND WRITES TO M:LO.

*     ***INPUT BUFFER***

*     ***80 BLANKS***

*  WE GIVE A TOP-OF-PAGE AT BEGINNING (AND END)

*  SWITCH:  "HAVE WE ISSUED A BLANK LINE ?":  0 = YES/1 = NO

*  WE SET THE SWITCH:  "BLANK LINE ISSUED SINCE LAST
*    COMMENT"

*  WE RESET THE SWITCH:  "BLANK LINE NOT ISSUED SINCE THE
*    LAST COMMENT"
```

The program's output has printed and control reverts to TEL.

```
!
.
.
.
```

Example 35. Using Load-Module-Name as Command Verb (FORTRAN Program)

-page heading-

```
!TABS 7
```

The user sets a tab stop for terminal input and turns on tab simulation.

```
!BUILD FILE1 (RET)

   1.000 (TAB)    I=1 (RET)
   2.000 (TAB)    WRITE (6,20) (RET)
   3.000 (TAB)    DO 10 J=1,10 (RET)
   4.000 (TAB)    I=I*3 (RET)
   5.000 10(TAB)  WRITE (6,30) I (RET)
   6.000 20(TAB)  FORMAT (1X,15HPOWERS OF THREE) (RET)
   7.000 30(TAB)  FORMAT (5X,I7) (RET)
   8.000 (TAB)    END (RET)
   9.000 (RET)

!COMMENT ON ME (RET)
```

```
!FORT4 FILE1 ON OUTFILE (RET)
```

FORT4 is called to compile source program FILE1, with ROM output going to OUTFILE.

```
OPTIONS>NS (RET)
```

FORT4 prompts for options. The user suppresses the partial-summary output.

```
!SET F:6 UC (RET)
```

This command will cause output to device 6 to be directed to the terminal.

```
!LINK OUTFILE ON POW3 (RET)
```

Call LINK to create load module POW3.

```
    LINKING OUTFILE
```

The LINK subsystem responds.

```
P1 ASSOCIATED

!POW3. (RET)
```

Load module POW3 is loaded into core and executed. The log-on account is used.

```
POWERS OF THREE
          3
          9
         27
         81
        243
        729
       2187
       6561
      19683
      59049

*STOP* 0
```

The program's output is printed.

```
!OFF (RET)

-accounting summary-
```

# 9. GETTING IN AND OUT OF PROCESSORS

## GENERAL

Once having logged on, you are always in one of three states of processing:

1. In a Job Step: You are in a system (or user) processor, i.e., in "normal" user-program execution.
2. In an Interrupt of a Step: You are at TEL level but have an interrupted processor associated.
3. Between Steps: You are at TEL level with no processor associated.

If you are in a processor, you can return control to TEL by depressing the CONTROL and Y keys (see BREAK and $Y^C$, below). Many TEL commands can then be issued to perform minor operations after which control can be returned to the processor that was interrupted. The issuance of certain other commands will cause either an abort of the previous job step or a diagnostic message. For example, interrupting META to call Edit will result in an abort of META; however, interrupting META to issue a DONT COMMENT command does not cause an abort and allows return of control to META. For a summary of the TEL commands and their effect when used during a job step interrupt, see Chapter 3 of the UTS/TS Reference Manual, Publication 90 09 07. A partial list is given in Chapter 11, "Output Control".

### QUIT and CONTINUE COMMANDS

After you have interrupted a processor and have optionally issued one or more commands, you have three alternative courses of action:

1. Return to the interrupted processor by issuing a CONTINUE command.
2. Discontinue the previous operation by issuing a QUIT command.
3. Call another processor, which has the effect of aborting the previous operation.

Each of these actions is illustrated in the following example.

Note that both END and STOP are equivalent to QUIT, and that GO is equivalent to CONTINUE.

Example 36. Interrupting, Continuing, and Quitting Execution

```
    :

!BUILD INPUT (RET)
1.000 SYSTEM Y^C←
```

The user wishes to build file INPUT but forgot to set tab stops before building the file. So he now interrupts Edit by simultaneously depressing CONTROL and Y which the system echoes as a left arrow. Control is given to TEL.

```
!TABS  10,19,37 (RET)
```

He now sets tab stops for the terminal I/O.

```
!CONTINUE (RET)
```

He issues a CONTINUE command which takes him back to Edit (with no prompt). He retypes his first line, since he interrupted while typing this line.

```
 (TAB)              SYSTEM (TAB) SIG7 (RET)

 2.000 (TAB)        SYSTEM (TAB) BPM (RET)

 3.000 START (TAB)  M:PRINT (TAB) (MESS,MES) (RET)

 4.000 (TAB)        M:EXIT (RET)

 5.000 MES (TAB)    TEXT (TAB)   "MESSAGE TO TERMINAL" (RET)

 6.000 (TAB)        END (TAB)    START (RET)

 7.000 (RET)
```

```
!META INPUT ON BOFILE,LP (RET)

WITH> (RET)

Y^C←
```

He calls META to assemble the program, but then spots an error in the program and interrupts with Y^C.

```
!EDIT (RET)
```

He calls Edit to correct the error.

```
FINI
```

The system informs him that META has been aborted.

```
EDIT HERE

*EDIT INPUT (RET)
```

He wants to retype line 5 to change TEXT to TEXTC.

```
*IN5 (RET)
```

He issues an Insert command (IN) to correct the line.

```
 5.000 MES (TAB)    TEXTC (TAB)  "MESSAGE TO TERMINAL" (RET)

*END (RET)

!META INPUT OVER BOFILE,LIST (RET)

WITH> (RET)
```

He again calls META to assemble file INPUT and request an output listing. Note use of OVER to reset file extension, ensuring that any output from the previous aborted assembly is overwritten.

```
!LINK (NP) BOFILE ON MES (RET)
```

LINK is called to create load module MES.

```
   LINKING BOFILE

!EDIT INPUT (RET)

EDIT HERE

*TY1-6 (RET)
```

He now wishes to see the corrected source and calls Edit to display the file.

1.000 SYSTEM SIG7

2.000 SYSTEM BPM

3.000 Y^C

He decides he does not want to see the entire file after all, so he simultaneously depresses CONTROL and Y to interrupt Edit and return control to TEL. The system did not echo a left arrow since it was active, i.e., not in a wait state.

!MES. (RET)

He now wants to load and execute program MES.

PROCESS NOW ACTIVE: QUIT OR CONTINUE

The system informs him that the program cannot be loaded without erasing the previous process (Edit), and that he must issue either a QUIT or a CONTINUE command.

!QUIT (RET)

FINI

This QUIT command now allows him to issue any TEL command since it has caused Edit to be given up.

!MES. (RET)

He now loads and executes.

MESSAGE TO TERMINAL

The program output prints.

!OFF (RET)

- accounting summary -

**BREAK and Y^C**

Any UTS subsystem or processor, as well as a user's object program, can be interrupted by depressing the BREAK key. Use of the BREAK causes one of the following to occur:

1. If you are in a processor that has no command language (e.g., assembling with META), control is given to TEL whenever a convenient interrupt point is reached. TEL then prompts for a command.

2. If you are in communication with a subsystem that has break control (e.g., Edit, BASIC), and in a subprocess such as listing or copying, control is given to the subsystem, which prompts for its next command or possibly issues an interrupt message.

3. If an object program or subsystem is in a process that does not have break control (i.e., has not used the M:INT Monitor service) control is given to TEL.

The control combination Y^C always returns control to TEL. This type of interrupt can also be caused by depressing the BREAK key more than three times. (Certain subsystems may take special action on receipt of two or three break signals.)

The next example illustrates use of the different types of interrupts.

Example 37. Using CONTROL/Y and the BREAK Key

```
    :
    :
!EDIT FILES (RET)

 EDIT HERE
```

The user decides to make changes to FILES.

```
*IN 7,1 (RET)
    7.000 Y^C←
```

He starts to modify the file but changes his mind and interrupts by hitting CONTROL and Y simultaneously. The system echoes a left arrow.

```
!PCL (RET)
```

Control returns to TEL. The user calls the PCL subsystem.

```
FINI
```

The system informs him that the previous process (Edit) has been terminated.

```
PCL HERE
```

PCL identifies itself.

```
<LIST (RET)
```

The user asks to have listed the names of the files that are currently in his RAD directory.

```
ARCSINE
CONWAY
DATA
FILES
INPUT
(BRK)

..     5 FILES LISTED.
```

He does not want to see the entire list, so he hits BREAK to stop the output.

```
<COPY FILES TO LP (RET)
```

Return is made to the command state of PCL. The user issues a COPY command to copy file FILES to the line printer.

```
<END (RET)
```

He leaves PCL.

```
!EDIT FILES (RET)
EDIT HERE
*TY1-7 (RET)
```

He calls the Edit subsystem and issues commands to type lines in file FILE.

```
1.000 * THIS PROGRAM SEARCHES NAME/ADDRESS-RECORD FILES ORDERED BY

2.000 * ZIP-CODE LOCALITIES.  IT ALSO INSERTS AND DELETES N/A RECORDS. THE

3.000 * CALLING SEQ (BRK) (BRK) (BRK) (BRK) ←
```

He does not want to see the entire file, so he returns to TEL by hitting BREAK four times. He could also have returned to TEL by depressing CONTROL and Y.

```
--ENTER X TO ABORT COMMAND.  ANY OTHER CHARACTER CONTINUES.
```

The Edit subsystem has break control and types this message in response to the first break. The subsequent breaks cause direct return to TEL.

```
!OFF (RET)

FINI


- account summary -
```

**PROGRAM ABORTS**

Many conditions can cause your program to be aborted, e. g., an invalid operation code. When an abort occurs, the system prints an abnormal or error code (e. g., 4A00) followed by a message telling you the reason for the abort. The UTS/TS Reference Manual, Publication 90 09 07, Appendix B, contains listings and explanations of the Monitor error messages.

The following example shows a program that will simply read two records of predetermined size from a file and print them at the terminal. However, a misspelled label in line 8 (BUF instead of BUFF — not a syntax error) causes an attempt at execution time to read into relative location 12. Since this location is in a write-protected procedure area of the program (i. e., the area cannot be stored into), the program is aborted and an appropriate message issued by the system. (Note that the program in this example is not intended to be realistic, but is designed solely to illustrate as simply as possible the "bug", and thereby the point of the example.)

In the following example and in all succeeding ones, we will no longer explicitly indicate carriage return and tab-character usage except as necessary to avoid possible ambiguity.

Example 38. System Handling of an Abort during Execution

```
    :
!EDIT
EDIT HERE

!BUILD READ
```

The user builds files READ and LINES (following line 14).

```
1.000          SYSTEM    SIG7

2.000          SYSTEM    BPM

3.000          REF       M:SI

4.000          REF       M:UC

5.000 BUF      EQU       10

6.000 START    M:READ     M:SI,(BUF,BUFF),(SIZE,16)

7.000          M:WRITE    M:UC,(BUF,BUFF),(SIZE,17)
```

```
   8.000          M:READ    M:SI,(BUF,BUF+2),(SIZE,8)

   9.000          M:WRITE   M:UC,(BUF,BUFF+2),(SIZE,9)

  10.000          M:EXIT

  11.000 BUFF     RES       4

  12.000          DATA,1    X'15'

  13.000          END       START

  14.000 (RET)

!BUILD LINES

  1.000 HELLO, TERMINAL!

  2.000 GOODBYE

  3.000 (RET)

!COMMENT ON ME

!META READ

WITH> (RET)

*   ERROR SEVERITY LEVEL:  0

* NO ERROR LINES
```

He calls META to assemble the source file. META produces summary messages indicating that there were no assembly errors.

```
!SET M:SI DC/LINES;IN
```

He sets M:SI to the input file.

```
!RUN (NP) (RET)

     LINKING

DEFAULT CORE LIBRARY

HELLO, TERMINAL!
```

The first record is written.

```
4A00 THAT'S NOT YOUR BUFFER
```

The system returns an abort message with an error code when the user tries to read the second record into location 12, which is in a protected area.

```
!
```

.
.
.

# 10. ASSIGNING DCBs

## DATA CONTROL BLOCKS

A data control block (DCB) is a standardized table of information about the characteristics of an existent data-file or one to be created. The system's file-management service routines use the DCBs essentially to obtain detailed information both about the file, (i.e., the data) and the physical storage media assigned to it. This, combined with information supplied in a given service request, completely defines the requested operation. These routines also use the DCB to post or update dynamically-variable "historical" information concerning the data file (specific results of the last I/O operation performed, for example) to which the user's program and other system routines may refer.

The DCB also is, effectively, the connecting link between the user's input/output service requests, file-management commands, etc., and the actual RAD-storage space or peripheral device from which or on which a given data file is to be read, written, copied, saved, deleted, and so on. Sometimes the reference to this "link" is explicit at the user's level, as for example in an M:READ or M:WRITE monitor procedure in a Meta-Symbol program, or in a SET command when the user needs to assign or reassign program input or output DCBs to specific RAD files or devices.

### MEANS OF FILE/DEVICE ASSIGNMENT

The !SET command may be used to explicitly assign any DCB (excepting M:UC, M:OC, and M:XX) to a file or device, as seen in a number of preceding examples. (SET can also be used for setting and resetting various parameters, or relatively fixed items of information in a DCB, e.g., file options, but a general discussion of this usage does not concern us here.)

The !OUTPUT, !LIST, and !COMMENT commands can be used to implicitly assign several standard system DCBs commonly used by system processors: M:GO, M:LO, and M:DO, respectively. Usage of these commands was also shown and described for specific cases in preceding chapters. And, summarizing topics covered in Chapters 5 and 8, the source, rom, and list parameters of META, FORT4, and lm-name commands implicitly assign the M:SI, M:GO, and M:LO DCBs.

In general, the SET command need only be used to assign user-program files for DCBs that have no default assignment (or an undesired one), or to assign standard system DCBs, other than the ones named above, for special-option processor outputs, e.g., the CO, BO, and SO options and the corresponding M:CO, M:BO, and M:SO DCBs. To assign M:SI, M:GO, M:LO, and M:DO, the choice between the several means described above is simply a matter of the user's convenience, as they each "do the same job", excepting that SET cannot be used in a job-step interruption. (See SET Command below, for specific information concerning BASIC.)

### STANDARD SYSTEM DCBs

The system includes an extensive set of standard DCBs that provide for the majority of system- and user-program needs. The link-loader supplies a uniform loader-constructed copy of these DCBs to the user's program as required to satisfy references thereto. These DCBs all have names of the form M:xy, where xy generally corresponds to a system-defined operational label (discussed under SET Command below). These DCBs, when used on-line, have the on-line default assignment (if any) defined by the system for the corresponding operational label. Note that the default assignments for M:UC and M:OC, the user's terminal in both cases, are really fixed assignments, i.e., you cannot change them. (The default assignments can vary with individual installations, and most of them differ for batch operations.)

Although a number of system DCBs default to the user's terminal, the M:UC DCB is unique because (1) its fixed assignment is to the terminal – like M:OC, and (2) output through it is treated differently by the Monitor than output to the terminal via any other DCB – unlike M:OC. For terminal output via any DCB other than M:UC, the Monitor's COC (Character-Oriented Communications) routines automatically append a carriage-return/line-feed combination to each record written (with or without a terminating carriage return). The COC routines do not append such a combination to output written via M:UC; it will substitute that character combination, however, for any carriage-return or line-feed character in the record. This difference allows you, when using M:UC, to produce one physical line at the terminal with a series of records. (See UTS/TS Reference Manual, Publication 90 09 07, Chapter 8, for details.)

In addition to the standard system DCBs, the link-loader will supply a uniform loader-constructed DCB for any M:ab DCB reference where M:ab is not known to the system, and for any DCB reference of the form F:ab, such as produced by FORTRAN IV for program files, where ab corresponds to the FORTRAN unit number. In these cases, the DCBs neither have a default assignment nor are they automatically defined for input or output - excepting F:101 through F:108, the FORTRAN standard units. They are also not defined for final disposition, and an I/O function and disposition parameter (e. g., IN, SAVE) may need to be set as well if the assignment is to a file or labeled tape. (These settings are described below.)

### ASSIGN/MERGE TABLE

DCB assignments, excepting those for M:SI, automatically remain in effect across job steps until reset or negated. Assignments can be reset or negated between job steps. The mechanism for setting and resetting assignments is the assign/merge table, during an on-line session. An M:SI assignment is effective only for a single job step; following that step it always reverts to its default assignment, the user's terminal.

Any assignment made by any of the means described above causes an entry to be made in your assign/merge table. At the beginning of any job step involving a processor (including LINK) or a user's program, the entries in the assign/merge table are merged into the corresponding DCBs. (An entry in the table is deleted by a !SET dcb 0.)

The apparent negation of an assignment achieved specifically by means of a DONT... command, e. g., DONT LIST, bypasses the assign/merge table and affects only a switch in the user's JIT (and not the implied DCB) at the time the command is issued, whether between job steps or during a job-step interruption. Only the standard processor outputs written via M:GO (OUTPUT), M:LO (LIST), and M:DO (COMMENT) DCBs can be affected in this way.

## OUTPUT, LIST, COMMENT COMMANDS

Control over output from META, or FORT4, or a standardized user-processor may be exercised with the following commands before the processor command is issued:

- OUTPUT ON or OUTPUT OVER followed by a file name. This command specifies the destination of the relocatable-object output (ROM) from the processor via the M:GO DCB. M:GO defaults to a special file, which you may refer to in some cases with a dollar sign ($).

- LIST ON or LIST OVER followed by ME, LP, or file names. This command specifies the destination of the listing output from the processor, via the M:LO DCB. For META and FORT4, M:LO effectively has no default assignment. Either an explicit assignment must be made or the !LIST command given to turn on the LO-output switch in the user's JIT. Apart from META and FORT4, M:LO defaults to the terminal.

- COMMENT ON or COMMENT OVER followed by ME, LP, or file name. This command specifies the destination of error commentary from the processor, via the M:DO DCB. M:DO defaults to the user's terminal. Therefore, COMMENT need not be used unless you want to direct error commentary to a destination other than your terminal.

In the following example, we specify destination files for the META output by using the LIST and OUTPUT commands, and (for purposes of illustration only) turn off the diagnostic output. We then assemble and execute the subprogram, but trap. We do not detect any errors in the source program, so in order to find out if we have assembly errors (which do show on the listing, however) we issue a COMMENT command to turn error comentary back on. We reassemble, find that we have a syntax error, and correct the line before reassembling.

Example 39. Controlling the Destination of Processor Output

```
    :

!BUILD COUNTER

  1.000         SYSTEM SIG7

  2.000         SYSTEM BPM

  3.000 BEG     LI,1 100
```

```
4.000          STW 1,X
5.000          M:EXIT
6.000 X        RES 1
7.000          END BEG
8.000 (RET)
```

```
!LIST ON LOFILE
```

Before calling META, the user directs listing output to file LOFILE.

```
!OUTPUT ON BIN
```

He specifies that the ROM output is to go to file BIN.

```
!DONT COMMENT
```

He overconfidently turns off error commentary.

```
!META COUNTER

WITH (RET)

!RUN BIN ON CNTR100
```

He requests load-module output on file CNTR100.

```
      LINKING BIN

DEFAULT CORE LIBRARY IS NOT NEEDED

    A400 YOU TRAPPED
```

An abort message prints indicating the program would not execute properly.

```
!COMMENT
```

The user does not spot any errors in the source, so he issues a COMMENT command to cause error commentary from META to appear at the terminal.

```
!META COUNTER OVER BIN,LOFILE

WITH> (RET)
```

He calls META again. Files BIN and LOFILE were created when META was previously called, so they must be respecified in order to be recreated or written over, rather than extended. This time the error commentary prints at the terminal and indicates that statement 4 is in error.

```
4    01 0000  1      35060001 N     STW 1,X
                                        4.000

**** ILLEGAL CF

*    ERROR SEVERITY LEVEL:  3

!EDIT COUNTER

EDIT HERE

*IN 4
```

He wants to change line 4 and uses the Insert (IN) command to enter a corrected statement.

```
   4.000 STW,1 X

*END

!META COUNTER OVER BIN,LOFILE

WITH> (RET)
```

He calls META again. Error commentary will still be directed to the terminal, since the previous COMMENT command is still in effect.

```
*     ERROR SEVERITY LEVEL:  0

* NO ERROR LINES

!RUN (NP) BIN OVER CNTR100

          LINKING BIN
```

Since there are no errors in the assembly, he reloads and reexecutes the program, to recheck and get an updated load module.

Normal execution is indicated by a return to TEL with no message.

```
!OFF


- accounting summary -
```

## SET COMMAND

The general form of the !SET command is given in the UTS/TS Reference Manual, Publication 90 09 07, with descriptions of its many options and varied examples of its use. It is a complex command. Several forms, selected for particular uses, are as follows:

- To Assign a RAD File

  SET dcb /fid[;filopt...;filopt]

  where

  filopt is one of the file-option parameters given in Appendix C, Table C-3. Some of these are

| | | | |
|---|---|---|---|
| IN | – | input file | function |
| OUT | – | output file | function |
| INOUT | – | update file | function |
| OUTIN | – | scratch file | function |
| REL | – | release on close | disposition |
| SAVE | – | save on close | disposition |

  The defaults for the function and disposition parameters are interrelated, as follows: for IN or INOUT files, SAVE is the default; for OUT or OUTIN files, REL is the default. (Note that for an OUT or OUTIN

file the SAVE parameter does not actually cause the file to be permanently saved, but merely allows SAVE to be effectively specified in an M:CLOSE operation.)

- To Assign a Labeled-Tape File

  SET dcb tc[#nnnn[I]] /fid[;filopt...;filopt]

  where

  tc is a magnetic-tape device code: MT - any unit; 9T - 9-track; 7T - 7 track.

  #nnnn is a tape-header serial number (i.e., an internal reel number).

  I indicates that nnnn is an input serial number (INSN). If I is not specified nnnn is an output serial number (OUTSN).

  fid is the file identification of a file on the tape.

  filopt is as above, under RAD-file assignment.

  Note that the serial number, #nnnn, is the reel-identifying number that is to be read from the tape-header sentinel record of either an input or output tape. (Also, the disposition file options, SAVE and REL, have specialized meanings for tape operations, as described in Appendix J of the BPM/BP, RT Reference Manual, Publication 90 09 54.)

- To Assign a Peripheral Device (Other Than Magnetic Tape)

  SET dcb{dev / oplb}[;devopt...;devopt]

  where

  dev is a symbiont-output device code: LP - line printer, CP - card punch.

  oplb is a system-defined operational label. These are given in Appendix C, Table C-1 (see also below).

  devopt is a device-dependent device option; these are given in Appendix C, Table C-2, and mainly concern format control and read/write codes and modes.

  The system-default value of an operational label, e.g., SI, LO, or CO, is set by the individual installation, and normally will differ from on-line to batch mode. In UTS as distributed, the following operational labels and correspondingly named DCBs default on-line to the user's terminal: C, DO, EI, LL, LO, OC, SI, SL, and UC. Excepting the special label NO, all other operational labels have no "as-directed" default value. The operational label NO has the fixed meaning "no assignment", and while it is in force, effectively prevents any default assignment from being applied. This causes any output via a so-assigned DCB to be lost, and an immediate end-of-file return on input.

- To Clear a User-Set Assignment

  SET dcb 0

  This form causes any prior assign/merge table entry for the named DCB to be deleted from the table. Thus, any system-default assignments are allowed to take effect in subsequent job steps.

## GENERAL USAGE RULES

The following usage rules apply in general:

1. File or device options can be added or respecified, between job steps, for an already assigned DCB if the assignment was made by a previous SET, OUTPUT, LIST, or COMMENT command, or a processor-call parameter.

2. As stated earlier in this chapter, when assigning a file to any nonsystem-defined, loader-constructed DCB (excluding F:101, F:102, ... F:106 for FORTRAN standard units) you must also specify one of the file-option function parameters (IN, OUT, INOUT, OUTIN), and also the disposition parameter unless the default is desired. Thus, an output-file assignment for, say, FORTRAN unit 6 would be as follows:

   !SET F:6 /OUTFIL;OUT;SAVE

3. No more than 12 concurrent DCB assignments can be in effect via the assign/merge table. (If necessary, clear any not-currently-needed entries via SET dcb 0).

**BASIC SUBSYSTEM REQUIREMENTS**

The BASIC subsystem uses the following DCBs for its I/O, with each normally assigned to the user's terminal:

- M:SI - for reading editing-mode input; it must be reset to UC if previously changed.
- M:DO - for both editing- and execution-mode diagnostic output; it must be reset to UC if previously changed.
- M:LO - for output written by the PRINT statement and LIST command.
- M:CI - for input read by the INPUT statement.

The M:CI DCB must be explicitly assigned to a file if file input is desired, and M:LO might be set to the line printer if so desired. In any case, if any of these DCBs have been affected inappropriately by any of the means discussed in this section, they may effectively be reset to normal BASIC assignments by means of the "SET dcb 0" form.

Example 40. Setting DCB Assignments and Parameters with the SET Command

This example illustrates use of the SET command to direct input to and output from an assembly. The user obtains source output on tape, a compressed-output deck, a double-spaced output listing on the printer, and ROM output on a RAD file.

```
    .
    .
    .
!SET M:SO MT#A123/Z
```

This command will cause the source output from META to go to file Z on the magnetic tape having the serial number A123.

```
!SET M:LO LO;SPACE=2
```

The user wants the output listing double-spaced. Note that he must first assign M:LO. (But, see the META command below, where this assignment is changed.)

```
!SET M:BO /BINOUT
```

He wants the binary output to go to RAD file BINOUT. He could alternatively have used the command:

```
!OUTPUT ON BINOUT
```

```
!SET M:CO CP
```

This command assigns the DCB for compressed output to the card punch (utilization privilege is required).

```
!META INFILE ON ,LP
WITH>SO,CO
```

The user now calls META to assemble a source file. He requests a source listing on the line printer (privilege required) and a compressed output deck.

```
!RUN (NP) BINOUT
```

He now calls RUN to load and execute the program.

```
  LINKING BINOUT
!
  :
  :
```

Control returns to TEL, with no error messages having been issued.

# 11. CONTROLLING OUTPUT

## GENERAL

The several outputs from a compilation or assembly (or "standardized" user processor) can be selectively turned off by a DONT LIST, DONT OUTPUT, or DONT COMMENT command, either before calling a processor or during an interrupt of the processor. These specifications retain their effect across job steps, until reset or negated. Outputs may be resumed by a LIST, OUTPUT, or COMMENT command, or by specifying output destinations in a META, FORT4, or user-processor command. LIST affects the M:LO DCB, normally used for listing output; OUTPUT affects the M:GO DCB, normally used for ROM output, and COMMENT affects the M:DO DCB, normally used for diagnostic output. (M:DO cannot be affected with the META, FORT4, etc., command parameters.)

If you have assigned output to a "symbiont device", such as line printer or card punch, the output is stored on RAD until you give an explicit or implicit indication that it is complete and ready to be printed or punched. You do this explicitly by issuing the TEL command PRINT, or implicitly by logging off. (Note that utilization privilege is required for these central-site units, however.)

### DISCONTINUING AND RESUMING STANDARD OUTPUTS

You may interrupt META or FORT4 and turn off output by one of the following commands:

- DONT LIST turns off list output.
- DONT OUTPUT turns off binary output.
- DONT COMMENT turns off error commentary.

The DONT LIST and DONT OUTPUT commands may also be given before calling META or FORT4 if these outputs are not desired.

Output may be resumed by one of the following commands:

- LIST resumes list output as previously specified.
- OUTPUT resumes binary output as previously specified.
- COMMENT resumes error commentary as previously specified, or at the terminal by default.

Each of the above commands remains in effect during a session until you issue another command to redirect output.

The forms of these commands for explicitly directing or redirecting outputs are given in Chapter 10.

Example 41. Discontinuing and Resuming Output by OUTPUT, LIST, and COMMENT Commands

```
    :

!BUILD INFILE

     The user builds a source file of Meta-Symbol statements.

1.000            SYSTEM      BPM
2.000            SYSTEM      SIG7
3.000 START      LI,R1       1
4.000            SLS,R1      24
5.000            LW,R2       R1
6.000            STW,R2      Y
```

```
 7.000              M:EXIT
 8.000 Y            RES         1
 9.000              END         START
10.000 (RET)


!OUTPUT ON OUTFILE
!LIST ON LFILE
```

The user specifies the destination files for binary output and object listing.

```
!DONT OUTPUT
!DONT LIST
```

For his first assembly, he only wants to test for assembly errors, and so he turns off the OUTPUT and LIST options.

```
!META INFILE
WITH> (RET)
```

He calls META to assemble the source file.

```
        3   01 00000    22000001 N   START   LI,R1 1
                                     3.000
**** UNDEF SYM
        4   01 00001    25000018 N           SLS,R1 24
                                     4.000
**** UNDEF SYM
        5   01 00002    32000000 N           LW,R2 R1
                                     5.000
Y^C←
!QUIT
```

The user notices that he forgot to define R1 and R2, and so he interrupts by depressing $Y^C$ and aborts META by typing a QUIT command.

```
FINI
!EDIT INFILE
EDIT HERE
```

He calls Edit to insert definitions into the source file.

```
*IN2.5,.1
   2.500 R1    EQU     1
   2.600 R2    EQU     2
   2.700 (RET)
*END
```

He then leaves Edit.

```
! OUTPUT
!LIST
```

This time he believes that the program is error-free, and so he now resets the OUTPUT and LIST options.

```
!META INFILE
```

He now reassembles. File INFILE must be respecified since M:SI defaults to the terminal at each new job step.

```
WITH> (RET)
 *   ERROR SEVERITY LEVEL: 0
 * NO ERROR LINES
!
    :
    :
```

## PRINT COMMAND

Output directed to the symbiont output devices (card punch and printer) is normally not queued for actual output on those devices until you log off. This feature has the advantage of causing all of the output for one job to come out together.

However, you may want some of your output printed or punched immediately. The PRINT command causes your symbiont files to be closed and queued for output at once (if you have the required utilization permission).

Example 42. Causing Printer or Punch Output to be Queued by Issuing a PRINT Command

```
    :
    :
!COPY ME TO LP (RET)
```

The user wants to enter lines at the terminal to be copied to the line printer.

```
.THESE LINES ARE DIRECTED TO THE LINE PRINTER
.THEY ARE NORMALLY NOT QUEUED FOR PRINTING UNTIL THE USER LOGS OFF.
.THE FOLLOWING PRINT COMMAND WILL CAUSE THEM TO BE PRINTED.
.(ESC)F
```

The Escape/F signals end-of-input.

```
!PRINT (RET)
```

This command causes the line printer output to be queued immediately.

```
!
    :
    :
```

The session continues.

# 12. SAVING/RESTORING CORE IMAGES AND FILES

## GENERAL

A core image of a program in process, along with relevant program context, can be saved on a RAD file during an interruption of execution. You might often want to save the core image of a patched program at one or several stages of a complex debugging process, e.g., to ensure against errors in ensuing patches. The saving and reloading of core images is achieved with the !SAVE and !GET commands, shown in the first two examples to follow.

Although program-I/O file identification information is saved along with the core image, file-positioning information is not saved (and the files themselves may not be saved if closed automatically by the system). If, however, a given program is not sensitive to these considerations, then SAVE/GET can also be used as a production checkpoint-restart mechanism.

RAD storage is the predominant file-storage medium for the on-line user, because of the nature of remote on-line operation and the central role played by this type of storage in integrated batch/time-sharing operating systems such as UTS. The advantages of RAD storage over other types of file media were discussed briefly in Chapter 4. RAD files are, however, susceptible to loss in certain types of catastrophic system failures, or "crashes", that sometimes occur. Although the system provides extensive, automatic protection against complete file loss (as described below), generally on an "all files" basis, you can selectively create backup files anytime you feel this action is indicated. (For example, after creating an important file when working with a relatively new installation that has not yet ironed out all the wrinkles.)

Files may be saved or backed up either on the standard system save/restore magnetic tape, or on your own private tape (or on punched cards) by means of the !BACKUP, ≤COPYALL, and ≤COPY commands.

Another characteristic of RAD storage is that its capacity is fixed in a sense that magnetic-tape or punched-card storage is not, and that you can easily misuse it: (1) by not promptly deleting unneeded files (you are normally charged for permanent RAD space actually used, not the total extent allowed for your use), and (2) by allowing little-used files to remain on RAD. You can transfer files of the latter class to tapes by the same means used to create backup copies – but do not forget to delete them from RAD after verification of the copying!

## SAVE AND GET COMMANDS

You can take a "checkpoint" of a core image at some desired point by interrupting the execution and issuing a SAVE command. The core image of the program and other information that enables the system to reconstruct the program's environment (other than I/O-file positioning) are then saved on RAD. After you issue the SAVE command, the interrupted program can be resumed by a GO or CONTINUE command.

Later you can restore the checkpointed program to core by issuing a GET command. Following the GET command by a GO or CONTINUE command causes processing to be resumed at the point at which the checkpoint was taken.

In the next example, we assemble a program with META but discover coding errors when it does not execute properly. Instead of editing the source file and reassembling, we choose to enter patches with DELTA (see Chapter 7). To preserve a patched version of the program, we interrupt prior to execution and issue a SAVE command. The patched version is restored by a GET command in Example 44 and executed again.

Example 43. Saving a Core Image of a Program (SAVE Command)

```
    :

!BUILD INPUT (RET)

 1.000          SYSTEM          BPM

 2.000          SYSTEM          SIG7
```

```
 3.000            REF             M:UC
 4.000 START      LI,1            RETURN
 5.000            STW,1           EXIT
 6.000            LCI             0
 7.000            STM,0           REGS
 8.000            B               SUBR
 9.000 RETURN     CW,15           REGS+15
10.000            BNE             ERROR
11.000            CW,14           REGS+14
12.000            BNE             ERROR
13.000            LI,14           BA(REGS)
14.000            LI,15           56
15.000            SLS,15          24
16.000            CBS,14          0
17.000            BNE             ERROR
18.000            M:EXIT
19.000 ERROR      LB,2            ERR
20.000            M:WRITE         M:UC,(BUF,ERR),(SIZE,*2),(BTD,1)
21.000            M:EXIT
22.000 ERR        TEXTC           "REGISTERS NOT PRESERVED IN SUBR"
23.000 REGS       RES             16
24.000 SUBR       LI,2            100
25.000            BDR,2           $
26.000            B               *EXIT
27.000 EXIT       RES             1
28.000            END             START
29.000 (RET)

!META INPUT ON BO

WITH>SD
```

The user assembles the program, and asks for symbolic debugging code to be produced.

```
*     ERROR SEVERITY LEVEL:  0

* NO ERROR LINES

!RUN BO

     LINKING BO

DEFAULT CORE LIBRARY IS NOT NEEDED

REGISTERS NOT PRESERVED IN SUBR
```

The program error message prints.

```
!RUN BO UNDER DELTA
```

The user now runs under Delta because he wants to add patches to the program before executing again.

```
     LINKING BO
DEFAULT CORE LIBRARY IS NOT NEEDED
DELTA HERE
"ring"
```

The Delta debugging processor identifies itself and prompts with a bell.

```
BO;S (RET)
```

The user identifies the symbol table associated with the ROM (BO).

```
EXIT+20\ STW,1 EXIT+19 (LF)
EXIT+.15/ LI,1 100 (LF)
EXIT+.16/ BDR,1 $ (LF)
EXIT+.17/ LW,1 EXIT+19 (LF)
EXIT+.18/ B *EXIT (RET)
SUBR\ B EXIT+20 (RET)
```

He enters patches into the program (see Example 30 for meaning of these commands).

```
Y^C←
```

He depresses CONTROL and Y after the prompt to interrupt Delta. The system echoes a left arrow.

```
!SAVE MYJOB
```

He issues a SAVE command to save the patched program on file MYJOB.

```
!GO
```

The GO command takes him back to Delta.

```
START;G
```

He issues a Delta command to start execution of the program. (No prompt is given.)

```
ABORT CODE = 0 SUBCODE = 0
```

Delta prints this message on execution of an M:EXIT.

```
Y^C←
```

The user now leaves Delta.

```
!
```

Example 44. Restoring a Checkpointed Program (GET Command)

```
    :
!GET MYJOB
```

The user restores the checkpointed core image (the patched program on file MYJOB). This file was created by the SAVE command in Example 43.

```
!GO
```

The GO command causes a return to Delta, the processor that was interrupted to perform the SAVE.

```
START;G
```

The user initiates execution. (No prompt is given for this line.)

```
ABORT CODE = 0  SUBCODE = 0
```

The program completes execution.

```
Y^C←
!
    :
```

## BACKUP COMMAND

The BACKUP command provides a means of creating backup files. Files are copied to the standard system backup tape. Note that the usage of BACKUP may be subject to rules and restrictions conditioned by specific installation practices concerning the saving/restoring of files.

A keyed file called MAILBOX in the user's account will contain completion messages resulting from the backup process.

The next example illustrates use of the BACKUP command. This example also shows that we must wait for the backup process to complete before finding out what is in the MAILBOX file.

Example 45. Saving a File on the Standard System Backup Tape

```
    :
!BUILD MYFILE
```

The user builds file MYFILE.

```
    :
!BACKUP MYFILE
```

He issues a BACKUP command to copy the file on the system backup tape.

```
!COPY MAILBOX TO ME (RET)
            $
  300 FILE DOES NOT EXIST.
```

He wants to see what is in the MAILBOX file, but this file does not yet exist because the backup process is not complete.

```
!COPY MAILBOX TO ME (RET)
```

He waits a minute or so for the backup process to complete then issues the COPY command again.

```
16:18 MAY 25, '71 BY ABCD    BACKED UP FILE MYFILE
```

The completion message in file MAILBOX now prints.

```
!
    :
```

## COPYALL AND COPY COMMANDS

### SAVING ON TAPE

Many times you will want to save one, several, or all of your files on magnetic tape, in addition to those implied by the "proper usage" guideline offered near the beginning of this chapter. For instance, if you were going on an extended vacation, you might want to copy all your RAD files to tape and thus save RAD-storage charges. Or, if you are making many versions of a file, perhaps many assemblies, you may want to back up the original file on tape. In the next example, the user transfers all the files in his account to tape and pulls them off as needed in the session. He also transfers a single, additional file to the tape. The example shows the use of labeled tape rather than free-form tape. Using labeled tape, the user can request his files by name from the tape, as he would call them by name from his RAD directory. With free-form tape, he would have to know the ordinal positions of the files on the tape, and space forward or backward the appropriate number of files before he could read or write the files he wanted. (This example is illustrative of what can be done, but not necessarily of what may normally be done, since it implies usage of a central-site resource in a manner that may or may not be allowed in a given installation.) Note that copying one file or the entire account to tape does not automatically delete the file(s) from RAD; RAD files must be explicitly deleted by command. Also, tape files are no longer "in the system". That is, the system "knows" only of files in the user's account on RAD; the user is responsible for knowing the tape's label or number, and for notifying the central-site operator of the same (see Chapter 14).

Remember that any utilization of magnetic tape, as well as any other central-site resource, requires a prior permission by the installation.

See the UTS/TS Reference Manual, Publication 90 09 07, Chapter 5, for a description of the Rewind (REW) and Space-to-EOT (SPE) commands shown in the next example.

Example 46. Transfer of All Files in User's Account to Labeled Tape

```
    :
    :
!PCL
PCL HERE
<LIST DC
ARCSINE
DATAFIL
JOBFIL
ROMFIL
SOURCE
VPRIME
```

The user lists the names of the files in his RAD account.

```
<REW #3B96
```

He rewinds the tape (#3B96) he is going to write on, to be certain it is positioned to start of tape. This assumes, of course, that the tape has been mounted at the computer site. See Chapter 14 on user-operator communication. If the tape already contained files he wanted to keep, he would instead want to skip to the position following the last file on the tape by issuing the command <SPE LT#3B96.

```
<COPYALL DC TO LT#3B96 (RET)
```

He copies all the files in his account, in succession, to labeled tape (LT) #3B96.

```
<REW #3B96
```

He winds tape #3B96 that he has just written on before listing it. He visually compares the list with the one he received when he listed the RAD directory.

```
<LIST LT#3B96

ARCSINE
DATAFIL
JOBFIL
ROMFIL
SOURCE
VPRIME

<COPY ME TO NEW
.   :
    :
    :
. (ESC)F
```

The user creates a new file from the terminal.

```
_SPE LT#3B96
```

He spaces labeled tape #3B96 to the mark following the last file on the tape.

```
<COPY NEW TO LT#3B96/NEW
```

He adds file NEW to labeled tape #3B96.

```
<REW #3B96
<LIST LT#3B96

ARCSINE
DATAFIL
JOBFIL
ROMFIL
SOURCE
VPRIME
NEW
```

He rewinds tape #3B96 and lists the names of its files. PCL has successfully added file NEW after the last file written to tape, VPRIME.

```
<DELETEALL
DELETEALL?
<YES$
..   7 FILES DELETED
```

Satisfied that our files are safely stored on tape, he now deletes all files in his RAD account. PCL requires a "YES$" verification of the DELETEALL request, and upon its receipt, PCL deletes all of the user's files and so notifies him.

```
<REW #3B96
<COPY LT#3B96/NEW TO DC/INPUT (RET)
```

The user wishes to use file NEW as input to a processor later in the session. He calls it from the tape back to his account with a new name, INPUT.

```
<REMOVE #3B96
```

He issues the REMOVE command, which rewinds the tape and automatically issues a "dismount" message to the control-site operator.

```
<END
!
    .
    .
    .
```

# 13. SUBMITTING BATCH JOBS

## BATCH SUBSYSTEM

The BATCH Job-Entry Subsystem is used to submit a batch job deck stored on a file to the batch input stream. This job deck must include all appropriate batch control cards that would be needed for normal batch job submission. Later you may ask for the status of the job (e.g., WAITING, RUNNING, COMPLETED) by issuing the !JOB command. (This command is illustrated in the second example.)

Example 47. Submitting a Job via BATCH Subsystems for Execution

```
    :
!EDIT
EDIT HERE
*BUILD BATCHIN
```

The user builds a source program that he wishes to assemble in the batch environment.

```
1.000           SYSTEM          SIG7
2.000           SYSTEM          BPM
3.000           REF             M:LO
4.000 START     M:WRITE         M:LO, (BUF,MES), (SIZE,9)
5.000           M:EXIT
6.000           ERROR           THIS LINE CONTAINS AN ERROR
7.000           TEXT            "IT WORKS."
8.000           END             START
9.000 (RET)
*BUILD JOBA
```

He builds a file containing a batch job-control deck that will assemble the file called BATCHIN.

```
1.000 !JOB 2232, JONES
2.000 !ASSIGN M:SI, (FILE, BATCHIN)
3.000 !ASSIGN M:BO, (FILE, BINARY)
4.000 !ASSIGN M:DO, (FILE, ERRORS)
5.000 !METASYM SI, BO, LO
6.000 (RET)
*END (RET)
!BATCH JOBA (RET)
ID=0028   SUBMITTED   9:13 MAY 26, '71
```

He submits the batch job he has just created. The job identification (ID) prints as a hexadecimal value.

```
!
    :
```

## JOB COMMAND

The JOB command is used to ask for the status of a batch job. The system responds that the job is either completed, running, or still waiting to be run. The format of the command is

```
!JOB xxxx
```

where xxxx is a hexidecimal job ID from one to four characters in length (leading zeros may be omitted.)

Example 48. Using the JOB Command

```
    :
!JOB 28
```

The user requests the status of the job submitted in the previous example. The job identification 0028 is the same as the one reported when the job was submitted using the BATCH command.

```
WAITING:  1 TO RUN
```

The system answers that there is one batch still to be run before this job is run.

```
    :
!JOB 28
```

Later the user asks again.

```
COMPLETED
```

Now the job is complete

```
!COPY ERRORS ON ME
```

The user displays the diagnostics from the job at the terminal. Note that M:DO was assigned to file ERRORS in the job-deck JOBA.

```
      6           ERROR    THIS LINE CONTAINS AN ERROR
                  6.000
**** UNDEF SYM
**** ILLEGAL AF
****
 *    ERROR SEVERITY LEVEL:  3
!
    :
```

# 14. COMMUNICATION WITH THE OPERATOR

## MESSAGE COMMAND

The MESSAGE command causes a message to be sent to the central-site computer operator. The message length is limited to 50 characters.

The format of the command is

!MESSAGE message-text

In the next example, the user informs the operator that he needs a scratch tape.

Example 49. Sending a Message to the Operator

```
    :
!MESSAGE READY SCRATCH TAPE TO BECOME #9055.

      The user sends a message to the computer operator requesting that he be ready to mount a scratch tape.

!BUILD DATAPOINTS
      He builds file DATAPOINTS.

    :
!COPY DATAPOINTS TO FT#9055

      He requests that the file to be copied to a free-form tape with the serial number 9055. The system
      informs the operator where to mount the requested scratch tape.

    :
```

## MESSAGES FROM THE OPERATOR

The computer operator can send a message to an individual terminal or broadcast a message to all users. When he broadcasts a message, the message is placed into the right-hand part of the terminal page title, and it will be seen by a user when he receives a new page heading. A message sent to an individual terminal may appear anywhere in the user's output.

Note: If the PLATEN command was used to turn page headings off, the broadcast message will not appear.

In the next example, the user receives a message informing him that the system will soon go off. He issues a BACKUP command before logging off to insure that his latest files will be saved. This action may not really be necessary, depending on installation practice; the system normally will save all files automatically before going off.

Example 50. Receiving a Message from the Operator

```
    :

!BUILD XYZ

      The user builds a source file.
```

```
1.000           SYSTEM          BPM
2.000           SYSTEM          SIG7
3.000           REF             M:UC,M:LO

    :
    :
    :

!L^c
```

He wants to continue file building (or processing) on a new page, so he simultaneously depresses CONTROL and L to cause a page eject.

```
(page eject)


23:45 05/26/71  JONES   ABC  1BB-F[17] UTS WILL GO OFF AT 2400
```

Included as part of the page heading is the message from the operator UTS WILL GO OFF AT 2400. This message has been sent to all users.

```
!BACKUP XYZ
```

The user decides to terminate his processing at this time, and he uses a BACKUP command to save his last file.

```
!OFF

-accounting summary-
```

# APPENDIX A. TEL COMMAND SUMMARY

Table A-1 is a summary of TEL commands. The first column gives the command format, the second column gives the command's function and option codes. For the structure of file names (fid, rom, lmn) see Table B-1.

Table A-1. TEL Command Summary

| Command | Description |
|---|---|
| BACKUP fid | Saves the specified file on a system tape. In case of a crash in which files are lost, files on the tape will be restored. |
| BATCH fid | Enters the specified file in the batch job stream. |
| BUILD fid | Accepts a new file from the terminal. |
| COMMENT {ON / OVER} list | Directs error commentary to the specified device.<br>Options: list may be fid, LP, or ME. |
| CONTINUE | Continues processing from the point of interruption. |
| COPY d[(s)][/fid[(s)][,fid[(s)]]...][;d[(s)]<br>[/fid[(s)]][,fid[(s)]]...] ... [TO / OVER]<br>d[(s)][/fid[(s)]] | Copies file between devices or between RAD storage and devices:<br>Options:<br>d may be CP, DC, FT, LP, LT, or ME.<br>s may be a data code (E, H); a data format (X); a mode (BCD, BIN, 7T, 9T, PK, UPK, SSP, DSP, VFC, NC) or selection (x-y). |
| DELETE fid | Deletes the specified file. |
| DELTA | Calls the DELTA subsystem. |
| DISPLAY | Lists the current values of various system parameters. |
| DONT COMMENT | Stops error commentary output. |
| DONT LIST | Stops listing output. |
| DONT OUTPUT | Stops object output. |
| EDIT fid | Calls Edit to modify a file. |
| END | Terminates the current job step. |
| FORT4[sp] [ON / OVER [rom][,list]] | Compiles an XDS Extended FORTRAN IV source program.<br>Options:<br>sp may be fid or ME.<br>rom may be fid only.<br>list may be fid, LP, or ME. |

Table A-1. TEL Command Summary (cont.)

| Command | Description |
|---|---|
| FORT4[sp] [ON / OVER [rom][,list]] (cont.) | Output may be interrupted and continued by the following commands:<br><br>LIST — DONT LIST<br>OUTPUT — DONT OUTPUT<br>COMMENT — DONT COMMENT<br>CONTINUE — GO |
| GET fid | Restores the previously saved core image. |
| GO | Continues processing from point of interruption. |
| JOB jid | Requests the status of remotely entered jobs. |
| LINK[codes]mfl[,mfl]...[,mfl][ON / OVER lmn] [;lid[,lid]...[,lid]][UNDER FDP] | Forms the load module as specified.<br><br>Options:<br><br>library search: (L), (NL), (Pi), (FDP), (NP)<br>default: (NL), (P1)<br>display: (D), (ND), (C), (NC), (M), (NM)<br>default: (D), (C), (NM)<br>symbol tables: (I), (NI)<br>default: (I)<br><br>mfl may be fid or $; parentheses enclosing mfls cause merge of symbol tables.<br><br>lid must be a library fid. |
| LIST{ON / OVER} list | Directs the listing output to the specified device.<br><br>Options: list may be fid, LP, or ME. |
| lmn[sp] [ON / OVER [rom][,list]] | Initiates execution of a load module.<br><br>Options:<br><br>lmn has the form:<br><br>name [.[account][.password]]<br><br>absence of period and account specifies system account.<br><br>presence of period and absence of account specifies log-in account.<br><br>M:SI DCB is assigned to sp.<br><br>M:GO DCB is assigned to rom.<br><br>M:LO DCB is assigned to list. |

Table A-1. TEL Command Summary (cont.)

| Command | Description |
|---|---|
| MESSAGE text | Sends the specified message to the operator. |
| META[sp][ON / OVER [rom][, list]] | Assembles the specified source program.<br>Options:<br>sp may be fid or ME.<br>rom may be fid only.<br>list may be fid, LP or ME.<br>Output may be interrupted and continued by the following commands:<br>LIST, DONT LIST<br>OUTPUT, DONT OUTPUT<br>COMMENT, DONT COMMENT<br>CONTINUE |
| OFF | Disconnects terminal from system and provides accounting summary. |
| OUTPUT{ON / OVER}rom | Directs object output to the specified device.<br>Options: rom may be fid only. |
| PASSWORD xxxx | Assigns a new log-in password for the user. xxxx is 1-8 characters. Any of the following characters may be used: A-Z, a-z, 0-9, _, $, *, %, :, #, @, -, backspace. |
| PLATEN w[, l] | Sets the value of the terminal platen width and page length. |
| PRINT | Sends print output to the line printer and punch output to the punch. |
| QUIT | Terminates the current job step. |
| RUN[codes]mfl[,mfl]...[, mfl][ON / OVER lmn][;lid ... [, lid]...[, lid]][UNDER DELTA / FDP] | Loads the specified load module and starts execution.<br>Options:<br>library search: (L), (NL), (Pi), (FDP), (NP)<br>default: (NL), (P1)<br>display: (D), (ND), (C), (NC), (M), (NM)<br>default: (D), (C), (NM)<br>symbol table: (I), (NI)<br>default: (I)<br>mfl may be fid or $; parentheses enclosing mfls cause merge of symbol tables.<br>lid must be a library fid. |

Table A-1. TEL Command Summary (cont.)

| Command | Description |
|---|---|
| SAVE{ON / OVER} fid | Saves the current core image on the designated file. |
| SET dcb 0<br><br>SET dcb [oplabel / device / tapecode [tapeid]] [;dopt[;dopt]...[;dopt]]<br><br>SET dcb [tapecode [tapeid]/fid / filecode/fid] [;fopt[;fopt]...[fopt]] | Assigns file or device to a DCB or sets DCB parameter.<br><br>Options: see Tables C-1, C-2, and C-3. |
| START[lmn / $] [UNDER DELTA] | Begins execution of the program just loaded, either with or without an associated debugger. |
| STOP | Terminates the current job step. |
| STATUS | Displays the current accounting values. |
| Subsystem Calls<br><br>BASIC<br><br>CONTROL<br><br>DELTA<br><br>EDIT<br><br>FORT4<br><br>META<br><br>PCL<br><br>SUPER<br><br>lmn (user's program) | These calls are entered while TEL is in control of the terminal. They turn over control of the terminal to the subsystem. |
| TABS s[,s]...[,s] | Sets the simulated tab stops at the terminal. |
| TERMINAL type | Sets the terminal type for proper I/O translations. Type may be 33, 35, or 37. |

# APPENDIX B. FILE IDENTIFIERS AND THEIR PARTS

A file identification (fid) consists of a file name and optionally an account and/or a password. Special types of files are an lmn (load module) which is produced as a result of a LINK or RUN command, and a rom (relocatable object module) which is produced by an assembler or compiler. Table B-1 illustrates the structure of a fid.

Table B-1. File Identifiers and Their Parts

| Symbol | Structure |
|---|---|
| lmn | a file identifier (fid) that names a load module. |
| rom | a file identifier (fid) that names a relocatable object module. |
| fid | name [.[account] [.password]][t] |
| name | 1 to 10 characters of the X character set. |
| account | 1 to 8 characters of the X character set. |
| password | 1 to 8 characters of the X character set. |
| X character set | A-Z a-z 0-9 _ $ * % : @ - |

[t]The usage "name." is valid only when the fid is an lmn used as a command verb (see Chapter 8).

# APPENDIX C. SET COMMAND CODES

Tables C-1 through C-3 define the codes which may be used as options in the SET command.

Table C-1. DCB Assignment Codes - SET Command

| Type | Codes | Description |
|---|---|---|
| Operational Label | BI, BO, C, CI, CO, DC, DO, EI, EO, GO, LL, LO, PO, SI, SL, SO, UC | When the DCB is assigned to one of the system operational labels, the actual device connected to the DCB is that implied by the operational label, if any, for on-line mode. |
| | NO | No assignment, i.e., no default is to be applied. |
| Device | CP<br>PP<br>LP | Card punch.<br>Paper tape punch.<br>Line printer. |
| Magnetic Tape (tapecode) | 9T<br>7T<br>MT | 9-track tape.<br>7-track tape.<br>Any magnetic tape. |
| Secondary Storage (filecode) | DC | Any data file. (This is the default code if no other code is given.) |

Table C-2. Device Options - SET Command

| Format | Description |
|---|---|
| TAB = tab[, tab]... [, tab] | Specifies simulated tab stops and is followed by a list of up to 16 decimal numbers, separated by commas, giving the column position of the stops. If all 16 stops are not specified, the stops given are assigned to the first stops and the remainder are reset. |
| LINES = value | Gives the number of printable lines per page and is a single decimal value. The maximum value is 255. |
| SPACE = value | Gives the number of lines of space after printing and is a single decimal value. Values of 0 or 1 result in single spacing. The maximum value is 255. |
| DRC, NODRC | Turns the special formatting of records on and off. DRC specifies that the Monitor is not to do special formatting of records on read or write operations. NODRC specifies the Monitor is to do special formatting. If neither DRC nor NODRC is specified, NODRC is assumed by default. |
| VFC, NOVFC | Controls the formatting of printing by using the first character of each record. VFC specifies that the first character of each record is a format-control character. NOVFC specifies that records do not contain a format-control character. NOVFC is assumed by default. |
| COUNT = value | Turns on page counting and specifies the column number at which the page number is to be printed. |
| BCD, BIN | Controls the binary-EBCDIC mode for device read and write operations. |

Table C-2. Device Options - SET Command

| Format | Description |
|---|---|
| FBCD, NOFBCD | Controls the automatic conversion between external Hollerith code and internal EBCDIC code (FORTRAN BCD conversion). NOFBCD is assumed by default. |
| PACK, UNPACK | Controls the packed or unpacked mode of writing 7-track tape. PACK is assumed by default. |
| DATA = value | Controls the beginning column for printing or punching and is a decimal value. The maximum value is 144. |
| SEQ = value | Specifies that sequence numbers are to be punched in columns 77-80 of punched output. Four characters of nonblank sequence identification may be given for columns 73-76. Fewer than four characters are left-justified and filled with blanks. |
| L, NOL | Identifies the device type. L specifies that the device must be listing type. NOL specifies that it need not be listing type. NOL is assumed by default. |

Table C-3. File Options - SET Command

| Type | Format | Description |
|---|---|---|
| Organization | CONSEC | Consecutive record organization. |
| | KEYED | Keyed record organization. |
| Access | SEQUEN | Records will be accessed sequentially. |
| | DIRECT | Records will be accessed by key. |
| Function | IN | File is read only. |
| | OUT | File is write only. |
| | INOUT | File is to be updated. |
| | OUTIN | File is scratch. |
| Disposition | REL | File is to be released on closing. |
| | SAVE | File is prepared to be saved on closing. |

# APPENDIX D. LINK AND RUN COMMAND CODES

Tables D-1 and D-2 define the codes that may be used in the LINK and RUN commands.

Table D-1. Library Search Codes

| Code | Meaning |
|---|---|
| (L) | Specifies that the system library is to be searched to satisfy external references that have not been satisfied by the program. (This is the default option.) |
| (NL) | Specifies that a system library search is not required. |
| (Pi) | Specifies that the ith public core library is to be searched for unsatisfied external references. Default is to P1 if no other public core library is specified. Only one public library may be associated with a program. |
| (FDP or (P0) | Specifies that the FORTRAN Subprogram library P0, that includes the Debug routines, is required. |
| (NP) | Specifies that a public core library is not required. |
| Note: The sequence of the library search is as follows: User libraries are searched first, the public library is associated, then the system library is searched. | |

Table D-2. Error Displays

| Code | Meaning |
|---|---|
| (D) | Specifies that all unsatisfied internal and external symbols are to be displayed at the completion of the linking process (including library searches, if specified). The unsatisfied symbols are identified as to whether they are internal or external and to which module they belong. |
| (ND) | Specifies that the unsatisfied internal and external symbols are not to be displayed. |
| (C) | Specifies that all conflicting internal and external symbols are to be displayed. The symbols are displayed with their source (module name) and type (internal or external). |
| (NC) | Specifies that the conflicting symbols are not to be displayed. |
| (M) | Specifies that the load map is to be displayed upon completion of the linking process. The symbols are displayed by source with type resolution and value. |
| (NM) | Specifies that the load map is not to be displayed. |
| Note: The normal default options are D, C, and NM. | |

# APPENDIX E. SPECIAL TERMINAL KEYS

Certain terminal keys, key sequences, and key combinations cause action to be taken other than simple transmission of the character. Table E-1 illustrates these key sequences and the action produced.

Table E-1. Special Terminal Keys

| Key Sequence | Action Produced |
|---|---|
| (ESC)E | Sets or resets the flag that controls echoplex output. |
| (ESC)F | Causes end-of-file action on input. |
| (ESC)I or $I^C$ | Functions as a tab key. |
| (ESC)L or $L^C$ | Causes spacing to a new page and printing of new page heading. |
| (ESC)(LF) or (ESC)(RET) | Simulates a local line feed. No activation occurs. |
| (ESC)S | Sets or resets the flag that controls space-insertion mode. |
| (ESC)T | Sets or resets the flag that controls tab simulation. |
| (ESC)U | Sets or resets the flag that controls translation of lower case characters. |
| (ESC)X or $X^C$ | Erases current partial input line. |
| (BRK) | Causes an interrupt and return of control to processor, if processor has break control; otherwise control goes to TEL. More than three BREAKs cause return of control to TEL. |
| (LF) or (RET) | Causes a carriage return. |
| (RUB) | Deletes the last character received. |
| $Y^C$ | Causes an interrupt and return of control to TEL. |

Xerox Data Systems

# READER COMMENT FORM

XEROX

**We would appreciate your comments and suggestions for improving this publication.**

| Publication No. | Rev. Letter | Title | Current Date |
|---|---|---|---|
| | | | |

**How did you use this publication?**

- ☐ Learning
- ☐ Installing
- ☐ Operating
- ☐ Reference
- ☐ Maintaining
- ☐ Sales

**Is the material presented effectively?**

- ☐ Fully covered
- ☐ Well illustrated
- ☐ Clear
- ☐ Well organized

**What is your overall rating of this publication?**

- ☐ Very good
- ☐ Fair
- ☐ Very poor
- ☐ Good
- ☐ Poor

**What is your occupation?**

**Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the XDS Software Improvement or Difficulty Report (1188) instead of this form.**

**Thank you for your interest.**

Fold and fasten as shown on back.
No postage needed if mailed in U.S.A.

**Your name and return address.**

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 229
EL SEGUNDO, CALIF.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

**Xerox Data Systems**

701 South Aviation Boulevard
El Segundo, California 90245

ATTN: PROGRAMMING PUBLICATIONS

CUT ALONG LINE

FOLD