Price: $5.00

TYMSHARE MANUALS
TYMCOM-IX

EXECUTIVE

REFERENCE MANUAL

MAY 1976

TYMSHARE, INC.
CUPERTINO, CALIFORNIA 95014

TYMSHARE, INC., Litho in U.S.A.

CONTENTS

Page

Page

Section 1
INTRODUCTION

The EXECUTIVE is the key to the Tymshare TYMCOM-IX computer system. All the system's languages and applications programs are called from the EXECUTIVE. In addition, certain functions which do not require a subsystem can be performed directly by EXECUTIVE commands. These functions include:

- Entering and leaving the Tymshare system
- Creating, deleting, and renaming files
- Setting access controls for files and file directories
- Determining terminal connect time, machine number, and other utility information.

The main purpose of the EXECUTIVE is to make it easy to use the Tymshare system. The commands are simple, straightforward, and easy to learn, and they have many optional features that can be learned in a few minutes.

Files are an important part of the Tymshare system. Simply, a file is a program, text, or data stored on a disk or other storage device. Files are permanent storage; that is, they stay on the system until they are explicitly deleted. Files are identified by the name the user gives them when he creates them.

SYMBOL CONVENTIONS

The symbols used in this manual to indicate carriage return, line feed, alt mode/escape, and the emergency exit key are as follows:

carriage return:	↩
line feed:	↴
alt mode/escape:	⊕
emergency exit key:	☆

The keyboard position for the emergency exit key varies among terminals. It is usually a control back arrow (←^{c}) or a control underscore (_^{c}). The character has ASCII code 159 (237 octal) and the internal code 127 (177 octal). The use of the emergency exit key is discussed on page 18.

To indicate clearly what is printed by the computer and what is typed by the user, all user-typed information is underlined. For example, in the lines

```
-COPY ↩
FROM FILE: T↩
TO DATAFI ↩
```

the computer prints the prompt character (–) and the user types COPY followed by a carriage return. The computer prints FROM FILE: and the user responds with T and a carriage return. Then the computer prints TO and the user types DATAFI and a carriage return.

Lowercase letters in a command form identify the type of information to be entered. For example, the characters *file name* in the command form

–RUN file name ↩

indicate that the user types the name of a file at that point.

Control characters are denoted in this manual by a superscript c. For example, D^c denotes control D. Control characters do not print on the terminal but are shown in examples for clarity.

ABOUT THIS MANUAL

Section 2 of this manual discusses the basic elements of using the Tymshare TYMCOM-IX system. It explains how to connect to the Tymshare network, pass through the security precautions, and begin using EXECUTIVE commands. This section also describes some mechanical procedures for easing system interaction.

Sections 3 through 8 document all major EXECUTIVE commands: for transferring to and from subsystems, creating and deleting files, controlling file security, obtaining file information, and manipulating files.

Sections 9 through 13 document more advanced features, in particular, several programs that have been designed to supplement and enhance the EXECUTIVE capabilities.

Finally, the two appendixes provide summaries of control characters and commands for easy reference by the experienced EXECUTIVE user.

Section 2
USING THE TYMSHARE SYSTEM

The TYMNET access telephone numbers provide local access to the TYMNET Supervisor, which verifies the user's name and password and directs him to the computer assigned to his user name. Tymshare's communications network is compatible with many various terminals. It can transmit at any one of several transmission speeds, accommodate both uppercase and mixed-case terminals, and adjust for half duplex or full duplex operation. Tymshare makes available the FAILSAFE option, which protects the user from losing data upon an accidental disconnect, and the ONESC/OFFESC option, which can be used to assure uninterrupted operation during the running of programs from a command file or from tapes.

Security is provided by a password, one associated with each user name, which the user must know to get access to a computer. To maintain maximum security, the password for a given user name should be known only by those who need access to that particular user name, and it should be changed frequently. It will not be echoed on the system on full duplex terminals.

The user can abort any command by typing an alt mode/escape at any time before the carriage return or line feed is typed.

The WHY command can help if the EXECUTIVE has printed an error message. The user simply types WHY and the EXECUTIVE describes the error.

ENTERING AND LEAVING THE SYSTEM

Three steps are required to enter the Tymshare system: calling the Tymshare network (TYMNET), identifying the type of terminal, and identifying the user. The process is commonly called logging in. To leave the Tymshare system the user logs out.

Calling the Tymshare Network

The specific procedure for contacting the computer depends on the terminal arrangement. Two typical communication devices (data modems) are the acoustic coupler and the Bell System Dataphone. The procedures for using these two devices are described below. The Tymshare network normally operates terminals in the full duplex mode. The user should check his terminal for a full duplex/half duplex mode switch. Any questions about calling the Tymshare network can be answered by the local Tymshare representative.

Acoustic Coupler

The terminal is put in the on line mode, and the power cords from both the terminal and the acoustic coupler are plugged into standard three-prong wall outlets (or the terminal can be plugged into the coupler). The 25-pin EIA interface connector is plugged into the coupler.

A regular telephone handset is used to dial the local TYMNET access number. Lines that go through switchboards or multiphone lines should be avoided, as any second entry onto the line when the user is logged in will create line noise, even occasional disconnects.

A high-pitched tone signifies that the connection is made to TYMNET. The telephone handset is then placed in the coupler with the telephone cord facing as indicated on the coupler. The POWER or ORIGINATE button on the acoustic coupler is depressed.

Dataphone

The terminal is put in the on line mode and the TALK button is depressed. The local TYMNET access number is dialed. When a high-pitched tone sounds, the DATA button is depressed and the handset is replaced.

Identifying the Terminal

As soon as the connection to the Tymshare computer is made, the system activates the terminal and sends the message

```
please type your terminal identifier
```

This is sent at 30 characters per second (CPS) and is readable only on a 30 CPS terminal. On other terminals, a sequence of unintelligible characters is printed, and then the terminal pauses.

The user must at this point type the identification character for his terminal. The table below lists the terminal identification characters.[1] If the user has a question about which one applies to the particular terminal, he should contact his Tymshare representative.

[1] – The terminal identification character and other terminal input/output characteristics can be changed by using the self-documenting TIO program. To access it, the user types TIO at EXECUTIVE command level. When the TIO prompt character (:) appears, the user can type INSTRUCTIONS to obtain a printout of the program's capabilities.

Terminal Identification Characters

CPS (In/Out)	Identification Character	Tymshare Model Number	Other Terminals	Comments (Unless noted, ASCII and no parity assumed)
10	D	200	Teletype Model 33	
15	B		Teletype Model 37 (without parity)	
15	Carriage return		IBM 2741, Datel, AJ, or any Selectric™ terminal	Correspondence or EBCD code depending on the telephone number called
15	J		Teletype Model 37 (with even parity)	ASCII (even parity)
15/30	F		Syner-Data Beta	Terminal must be equipped with dual speed input/output
30	A	125,225, 315,325	CRT Terminal	No carriage return or line feed delay
30	C	310,311	Syner-Data Beta, UNIVAC DCT 500	
30	G		TermiNet 300, Memorex 1240 and 1280	Delay after line feed
30	E	100,110, 212,213, 1030	Execuport, Texas Instruments Silent 700 Series, NCR 260 Series	Short carriage return delay

Logging In

The log in procedure requires typing a user name and password, both of which are registered with Tymshare. The system checks both the user name and the password before admitting the user to the system. An optional project code can be typed during the log in. The project code is included in the billing information sent to the customer and can be used to assign costs.[1]

After the system prints

PLEASE LOG IN:

the user types a carriage return. The system replies with a request for the user name. The user types his user name, a colon, and the number of the system that he wishes to log in to, followed by a carriage return. If the user does not specify the system number, he is automatically logged in to a system that has been designated as his "home" system. The system next requests the password. The user types the password and a carriage return.

[1] – For more information about project codes, the user should refer to the *Tymshare TYMCOM-IX Account Supervisor Manual.*

A user's password must contain at least seven characters. A password may contain any characters except semicolon (;), control M (carriage return), control J (line feed), control shift K, control shift L, control shift M, control shift N, control shift O, and alt mode/escape. In addition, on lowercase terminals, the characters }, {, and ~ are not permitted. For security, the computer does not print the password on the terminal. After checking the password, the system requests the final information, the optional project code. For example,

```
please log in:↩

user name:  DELEON:2↩

password:↩                *The password is entered by the user but does not print at the terminal.*
PROJ CODE:  J-17↩
```

The user types a project code and a carriage return. If no project code is wanted, he types a carriage return only.

Any character typed in error in a project code may be deleted with control A (A^c).[1] However, control characters cannot be used to correct a user name or password. If a mistake is made in typing the user name or password, the escape key should be pressed once. The system will again request the user name or password.

After the user has entered the requested information, the system types the system number, date, and time. For example:

```
TYMSHARE   C2 3/12/76   10:27
-
```

The dash (—), which is called the EXECUTIVE prompt, indicates that the user is in the EXECUTIVE and can enter any valid EXECUTIVE command. If another terminal is already logged in under the same user name, the message

ALREADY ENTERED

is printed, followed by

PROJ CODE:

If there are typing errors while logging in, the system replies with

ERROR, TYPE

followed by the name of the item in error. In the following example, the user types an unacceptable user name, corrects his error, and continues to log in.

[1] Line editing with control characters is discussed on page 9.

```
please log in: DELEIN:2↵

error, type user name:   DELEON:2↵

password:↵
PROJ CODE: K-123-D↵

TYMSHARE   C2 3/12/76  10:30
-
```

Once the user is thoroughly familiar with this log in procedure, an alternate and faster method can be used as follows:

Nonprinting password is entered here.

```
please log in:  DELEON:2;;K-123-D↵

TYMSHARE   C2 3/12/76  10:32
-
```

A semicolon (;) must be typed between the user name and the password, and between the password and the project code. If no project code is needed, the user types a carriage return after the second semicolon.

The error diagnostics are the same regardless of which log in procedure is used. When the system indicates an error, the user can correct the error and type the rest of the log in information in the normal way.

The user is allowed two minutes to log in. If the log in is not completed within the time limit, the system prints a disconnect message and disconnects the terminal.

Logging Out

To leave the system or to log out, the user types

–LOGOUT ↵

or

–EXIT ↵

or simply

–LOG ↵

or

–EXI ↵

If the user types LOG or LOGOUT, the system replies with

CPU TIME: n SECS. *Number of computing seconds used.*
TERMINAL TIME: hours:minutes:seconds *Connect time.*

PLEASE LOG IN:

If the user types EXIT or EXI, the system prints

PLEASE LOG IN:

The user may either disconnect the terminal or log in again. It is unnecessary to retype the terminal identifying character to log in again. If the terminal remains connected to the computer and another log in is not made within two minutes, the system prints the disconnect message and disconnects the terminal.

Changing the Project Code

The user can change project codes without logging in again by using the PROJECT command as follows:

```
-PROJECT↩

ENTER NEW PROJ CODE: T-6↩

CPU TIME: 0 SECS.
TERMINAL TIME: 0:0:57

-
```

The system prints the time charged to the previous project code. Terminal time is in hours:minutes:seconds.

Typing errors in the new project code can be edited with control A (see the discussion about control characters, below).

A shorter method is to provide the new project code without being prompted, as follows:

```
–PROJECT T-3↩
```

The system responds with computing time and connect time as above.

Entering EXECUTIVE Commands

When the EXECUTIVE prints a prompt (–), it is ready to begin accepting commands. When the user has typed a line of input, that is, a command, including the carriage return, the system checks the first word against its list of internal EXECUTIVE commands, subsystem names, and list of library programs until a match is found. If a match is not found, EXECUTIVE prints a question mark (?) and another prompt. The user can then retype the command.

If the user detects an error while typing a command, he can abort the command by pressing the alt mode/escape key or can correct the error using one of the editing characters discussed below.

If part of the command is in error, the EXECUTIVE will print an error message explaining the error. Whenever the EXECUTIVE responds to a command with an error message, the user can type alt mode/escape, then WHY. The EXECUTIVE further explains the error and gives a new prompt. For example, the user tries to copy a binary file to the terminal:

```
-COPY SHRINK TO T ↵
ERROR, TYPE: TO ⊕
-WHY ↵                         The user aborts the command with an alt mode/escape, then types WHY.

FILE TYPE WRONG.

-
```

The user can obtain the same answer and still stay in the command by typing a line feed instead of alt mode/escape:

```
-COPY SHRINK TO T ↵
ERROR, TYPE: TO ↴

FILE TYPE WRONG.
ERROR, TYPE: TO SHRINK1 ↵
 NEW FILE ↵


-
```

Many of the EXECUTIVE commands documented in this manual can be shortened to their first three characters.

EDITING WITH CONTROL CHARACTERS

Three control characters can be used for line editing. On most terminals, they are typed by depressing and holding the CTRL key and then typing the desired letter. They do not print on the terminal, but they are shown in the examples in this manual so that their use will be clear. They are represented by a superscript c following the letter.

Control A

Control A (A^c) is used for deleting a single character or several characters one at a time. When it is typed, a back arrow (←) or underscore (_) prints on the terminal and the preceding character is deleted. For example, if the user is typing the TIME command but strikes the N key instead of the M key, he can delete the N by immediately typing A^c, then typing the correct character M:

—TINA^c←ME ↵

The entry is accepted by the computer as TIME.

The user can use control A several times to delete several characters in a row. For example,

—TUPEA^c←A^c←A^c←YPE ↵

is accepted as TYPE. Characters can be deleted only in the current line; that is, once the carriage return or line feed has been typed, the material in that line is no longer available for editing.

Control W

If the user wants to delete a whole word, he can do it most conveniently with control W. This control character deletes back to but not including the first preceding space. A backslash (\) prints on the terminal when W^c is entered. For example,

–TYPE FINEW^c\FILE ↵

is accepted as TYPE FILE.

The user can delete several words in a row by typing several control W's in succession. For example,

–DELETE FIOE1, FIOE2,W^c\\W^c\FILE1, FILE2 ↵

is accepted as DELETE FILE1, FILE2. Words can only be deleted in the current line, not in preceding lines.

Control Q

Control Q is used to delete the entire line. It prints an up arrow (↑) and returns the carriage. For example,

–DELETE FIOE1, FIOE2,Q^c↑
DELETE FILE1, FILE2 ↵

is accepted as DELETE FILE1, FILE2.

The user cannot delete more than the current line with control Q.

USING HALF DUPLEX TERMINALS

Two programs, HDX and FDX, facilitate use of half duplex terminals. If half duplex terminals are used with a full duplex system, each character typed by the user appears twice on the terminal: once when the key is struck, and again when the computer echoes the character. The HDX program, called by typing

–HDX ↵

in the EXECUTIVE, solves this problem and allows the user to employ his half duplex terminal with Tymshare's full duplex communications network (TYMNET). Once HDX is called, each character of user input appears only once as the key is struck. For example,

```
-HHDDXX ↵

-COPY ↵
FROM FILE: T ↵
TO ANY2 ↵
 NEW FILE ↵

THIS IS A TEST OF HDX ↵
D^c
-
```

To return to full duplex operation, the user types

```
-FDX↵
```

in the EXECUTIVE, resetting the terminal to full duplex operation.

USING THE FAILSAFE FEATURE

SETFAILSAFE and FAILSAFE are Tymshare programs which protect the user from losing time and effort due to line disconnect or terminal failure. If the line is disconnected before the LOGOUT or EXIT command is given, the system saves the program, variable values, and subsystem references on a file. When the user logs in later, his work can be resumed at the exact point of interruption. This means that if the line should be prematurely disconnected, he can be confident that his work has been saved. The work is saved on a special file called the fail-safe file.

The user creates a fail-safe file by typing SETFAILSAFE in the EXECUTIVE. The system responds by printing the OLD FILE/NEW FILE message (see page 25). For example:

```
-SETFAILSAFE↵
 NEW FILE↵


-
```

It is necessary to execute this program only once under each user name. The FAILSAFE feature is then set until it is removed.

NOTE: FAILSAFE will not protect new material being appended to a file in EDITOR,[1] *nor will it always protect work in progress in the event of a system crash. Because of the proprietary nature of the Tymshare Library programs, they cannot be saved on the fail-safe file.*

The SETFAILSAFE command creates an empty file named /$/ in the user's directory.[2] If a premature disconnect occurs, the user's work in process is dumped on the /$/ file. This file can be used for storage like any other file. Of course, the previous contents of /$/ are erased if a premature disconnect occurs.

The contents of the /$/ file are recovered by the FAILSAFE program. For example:

```
-FAILSAFE↵
 OLD FILE
 HAS BEEN CLEARED. YOU MAY PROCEED
```

The computer loads all the data stored on the /$/ file into the user's working area and erases the contents of the /$/ file. The FAILSAFE command returns the user to the subsystem he was in at the time of the disconnect. The name of the subsystem is printed as a reminder. For example, the user was working in SUPER BASIC when he was disconnected:

```
-FAILSAFE↵
 OLD FILE
 HAS BEEN CLEARED. YOU MAY PROCEED

SBASIC
```

[1] – See the *Tymshare EDITOR Reference Manual.*

[2] – The /$/ file can be created directly by the user with the COPY or RENAME command. It is then unnecessary to use the SETFAILSAFE command. The COPY and RENAME commands are discussed on pages 25 and 30, respectively.

The FAILSAFE command is equivalent to RECOVER and CONTINUE.[1] It resumes execution at the program location where disconnect occurred. However, all files are closed by the disconnect.

To remove the failsafe feature, the user types

```
-DELETE /$/↵
```

The fail-safe feature does not work after the DELETE /$/ command is used until the SETFAILSAFE command is entered again or the /$/ file is created directly.

As another fail-safe feature, every night Tymshare stores on magnetic tape the contents of all files that have been created or altered that day. Furthermore, at least every other week the entire contents of all files are written on magnetic tape. Therefore, if a file is accidentally erased or destroyed, the user should call his Tymshare representative to have it restored to his file directory as it existed at the end of the previous day.

USING UPPERCASE AND MIXED-CASE TERMINALS

The Tymshare system accepts input from terminals with only uppercase and with both uppercase and lowercase characters. A user with a mixed-case terminal has a choice between either using the full 126-character set or restricting the system to recognize only 64 different characters, with no lowercase.[2] The command ONLC activates the lowercase (full character set), and OFFLC deactivates the lowercase. For example:

```
-ONLC↵

-"I can use Lower Case."↵
```

The system accepts the comment in both upper- and lowercase.

or

```
-OFFLC↵

-"I CANNOT USE LOWER CASE."↵
```

The user can type in both upper- and lowercase, but the system accepts the comment in uppercase only; that is, it converts any lowercase letters to uppercase.

The normal mode of operation on the TYMCOM-IX, that is, the mode when the user first logs in, is uppercase only. In this mode, lowercase letters entered at the terminal are converted to uppercase. To use lowercase, the user gives the ONLC command.

All EXECUTIVE commands are in uppercase characters. When the command OFFLC is in effect, commands can be entered at the terminal in either uppercase or lowercase, since all characters entered are accepted by the system as uppercase; when the ONLC command is in effect, commands must be entered in uppercase only.

CONTROLLING SYSTEM INTERRUPTIONS

Two commands, ONESC and OFFESC, enable and disable system interruptions. These commands are especially valuable when paper tape is being used or if voice-grade communications lines must be used.

[1] – RECOVER and CONTINUE are discussed on page 20.

[2] – With the full character set, two characters have been reserved for system use. These characters have internal codes 91 (133 octal) and 93 (135 octal) and correspond to the left and right braces.

The OFFESC command inhibits all interruptions, including the alt mode/escape and emergency exit. If such an escape character is typed at any time—in the EXECUTIVE, in a language, or in a program—while OFFESC is in effect, it is ignored by the computer. When OFFESC is in effect, the only way to interrupt the action of the computer is to disconnect the telephone communication line by removing the receiver from the modem. The OFFESC command is given by typing

```
-OFFESC ↩
```

in the EXECUTIVE.

The ONESC command deactivates the OFFESC command. Subsequent alt mode/escapes and emergency exits are treated normally. The ONESC command is given by typing

```
-ONESC ↩
```

in the EXECUTIVE. If the OFFESC or ONESC command is not specified, the system may be interrupted by any of the usual methods; for example, by typing an alt mode/escape.

CHANGING THE PASSWORD

If authorized, a user may change his password as often as he wishes, to ensure the security of his files, with the PASSWORD program.[1]

The PASSWORD program is called by typing

```
-PASSWORD ↩
```

The computer responds with

```
ENTER PASSWORD:
ENTER IT AGAIN:
```

After each colon (:), the user enters his new password followed by a carriage return, but the password is not printed on the terminal. It is entered twice to ensure accuracy. The new password must be identical in two successive entries before it is accepted by the program.

For security, the password is not stored directly, but is ciphered before being stored in the computer. Thus, the password is known only to the user and those persons he chooses to notify. After the password has been successfully entered, the program prints the message

```
END OF JOB
```

and returns control to the EXECUTIVE. When the user changes his password on any system, it is automatically updated on all systems to which the user has access. Although the updating is not always instantaneous, it is seldom delayed more than 15 minutes.

[1] – The Account Supervisor must specify that a user may change his password. The method of authorization is described in the *Tymshare Account Supervisor Reference Manual.*

Section 3
TRANSFERRING INTO AND OUT OF THE EXECUTIVE

A user is automatically in the EXECUTIVE when he logs in; he can then call any language or subsystem from the EXECUTIVE. He cannot go from one subsystem to another without first returning to the EXECUTIVE.

CALLING LANGUAGES AND LIBRARY PROGRAMS

All Tymshare languages and Tymshare applications programs are called from the EXECUTIVE. In addition, some User Program Library (UPL) programs (discussed on page 17) are called directly from the EXECUTIVE.

The Tymshare Languages

Tymshare offers a variety of languages. For example, SUPER BASIC and SUPER FORTRAN are powerful, user-oriented computation languages with complex numbers, matrix commands, and string functions; BATCH FORTRAN IV is a fast, efficient batch type language; EDITOR is a text editing language.

All languages are called from the EXECUTIVE by typing the name and a carriage return. For example:

```
–BFORTRAN4↩
```

Many of the Tymshare languages display a special character, called a prompt, to indicate that the user may enter a command. The table below lists some of the more popular Tymshare languages and their prompts. Most of the applications programs display a colon prompt.

Language	Abbreviation	Prompt
SUPER BASIC	SBA	>
SUPER FORTRAN	SFO	>
BATCH FORTRAN IV	BFO	+
XFORTRAN	XFO	+
EDITOR	EDI	*
CAL	CAL	>
XCAL	XCAL	>
DCAL	DCAL	>

To exit from any language, the user types QUIT or Q.

The Tymshare Library

The Tymshare Library includes many large applications programs, each documented in its own manual. Among the most widely used applications programs are those listed below.

Program	Function
Information Management Library (IML)	Data management package, including capabilities for sorting, merging, selecting, purging, and replacing data
RETRIEVE	Information retrieval
STATPAK	Statistical analysis
SURVEY	Population survey analysis
TYMTAB	Financial modeling package

Several library programs are documented in this manual. They are:

CIPHER

COMPARE

DIRIT

FDM

PASSWORD

PERP

SCOMPARE

TAPE

TELECOPY

VERIF

The User Program Library

The User Program Library (UPL) contains programs that have been donated by Tymshare users and accepted by Tymshare. Tymshare examines the programs for general usefulness and adequacy of documentation before accepting them for the library.

The UPL programs are called by typing a crosshatch in front of the name. For example:

```
–#MULREG↵
```

The user may refer to the *Tymshare TYMCOM-IX User Program Index* for a complete listing of the UPL and Tymshare Library Programs.

RETURNING TO A LANGUAGE

Occasionally a user is working in a language and needs to use an EXECUTIVE command. For example, he finds that he needs to rename a file or to remove a prohibition against writing on a file. He can return to the EXECUTIVE with QUIT, use the appropriate EXECUTIVE commands, and then return to the language with the REENTER command. REENTER restores the work in process. For example:

```
-SBA↵                                 The user calls SUPER BASIC.

>LOAD BUDGETA↵                        He loads and runs a program.
>RUN↵

ERROR IN STEP 250:

FILE NAME NOT IN FILE DIRECTORY       The program tries to open a file which does not exist.

>QUIT↵                                The user transfers to the EXECUTIVE.

-RENAME DEC AS BUDG↵                  He renames his data file to correspond to the name used in
                                      the program, and reenters SUPER BASIC.
-REENTER↵
SBASIC                                All of his variable values have been restored.
>GO TO 250↵
```

In the example above, if the user calls SUPER BASIC by typing

```
–SBA↵
```

instead of REENTER, he has to reload his program and restart execution.

When the user leaves a language, any open files are automatically closed. After REENTER, the files are still closed. Therefore, occasionally, it is necessary to reopen the data files before continuing at the point of interruption. Also, it is not possible to reenter in the middle of a SUPER BASIC FOR loop.

Calling another language or library program or giving certain commands makes it impossible to use REENTER. The following commands do not interfere with use of REENTER:

COMMAND
CONTINUE
COPY
DATE
DECLARE
DEINIT
DELETE
DIRECTORY
DSC
DUMP
FDC
FILES
INIT
GFD
LAST
LIMIT
LIST
MEMORY
PFDC
PMT
PROJECT
REENTER
REMOVE
RENAME
STATUS
SYSNO
TIME
TOUT
TYPE
WHO
WHY

No other commands permit using REENTER.

It is impossible to use REENTER for an applications program.

EMERGENCY TERMINATION AND RECOVERY

If the user is working in a language, he can abort any command with the alt mode/escape, and he is returned to the language command level. If he uses the emergency exit key, however, he is returned immediately to EXECUTIVE command level. It is impossible to use REENTER after an emergency exit, but the DUMP command can be used.

If it is necessary to interrupt a job before completion, the DUMP and RUN commands can be used to save all the data and the work done so that the user may continue at a later time. The DUMP command saves the necessary information on a file and the RUN command recovers it.

The DUMP Command

To save work done in a language, the user uses QUIT to return to the EXECUTIVE and then uses the DUMP command. With this command, the user stores on a file the language he was working in, his program and data, and whatever work he had done thus far.

The DUMP command has the form

```
—DUMP file name↵
```

Any of the commands that do not erase user memory can be used between the exit from the language and the DUMP command. These commands are listed above.

The DUMP command cannot be used to save work in Tymshare Library programs or proprietary GO files since the user's work done by these programs is no longer accessible when the user returns to the EXECUTIVE.

CAUTION: Tymshare languages are under continual development and improved versions are frequently released. Since DUMP files may not be compatible from one version of a language to another, they should not be used for long term storage.

The RUN Command

When the user next logs in, he gives the command

```
—RUN file name ↵
```

to load the contents of the DUMP file into the system.[1] He is returned automatically to the language he was in before he gave the DUMP command and may continue from where he left off, as shown in the following example.

```
-SBA↵

>LOAD SBASAVINTRST↵
```

The user calls SUPER BASIC and initiates a program to calculate new savings account balances.

```
>RUN↵
ORIGINAL BALANCE IS $ ? 0↵
INTEREST RATE IS ? .07↵
MONTHLY DEPOSIT IS $ ? 35↵
NUMBER OF MONTHS IS ? 60↵

MONTH      BALANCE
    1           35.00
    2           70.20
    3          105.61
    4          141.23
    5          177.05
    6          213.09
    7          249.33
    8          285.78
    9 ⊕
```

While program is in process, he enters an alt mode/escape.

```
INTERRUPTED BEFORE 150
>Q↵

-DUMP HOPEFUL↵
 NEW FILE↵
```

He saves the work in process on the file HOPEFUL.

```
-EXI↵

please log in: DELEON;;↵
```

When he logs in again, he calls the DUMP file and picks up execution where he left off.

```
TYMSHARE   C2 3/10/76  15:55
-RUN HOPEFUL↵
SBASIC
>GO TO 140↵
   11           396.43
   12           433.74
   13           471.27
    •
    •
    •
```

Program continues. All work in the program is intact, but output that was in TYMNET at the time of disconnect is lost.

NOTE: The RUN command does not reopen files. If a SUPER BASIC program is interrupted while in a FOR loop, the RUN command will not work.

[1] – The RUN command, when used with a dump file, is equivalent to a RECOVER command followed by CONTINUE (see next page). When used with a symbolic file, the RUN command is equivalent to the COMMAND command (see page 45).

The RECOVER, REENTER, and CONTINUE Commands

The RECOVER command also restores a DUMP file. A RECOVER command followed by a REENTER command returns the user to command level.[1] For example:

```
please log in: DELEON;;↵

TYMSHARE   C2 3/10/76  15:57
-RECOVER↵
FROM FILE: HOPEFUL↵

-REENTER↵
SBASIC
>GO TO 140↵
   18          662.23
   19          701.09
   20          740.18
   21          779.50
```

A RECOVER command followed by a CONTINUE command will return to execution of the program if DUMP was used after using the emergency exit key to leave the program.

```
   22          819.05
   23          858.83 ☆
```

Running the same program, the user strikes the emergency exit key and saves his work on HOPEFUL.

```
-DUMP HOPEFUL↵
 OLD FILE↵

-EXI↵

please log in: DELEON;;↵
```

He logs in and recovers HOPEFUL.

```
TYMSHARE   C2 3/10/76  16:03
-RECOVER↵
FROM FILE: HOPEFUL↵

-CONTINUE↵
SBASIC061.22
   29         1102.41
   30         1143.84
   31         1185.52
   32         1227.43
```

The CONTINUE command begins execution immediately at the point of interruption without a GO TO.

RUNNING PROGRAMS FROM THE EXECUTIVE

A GO file is a compiled program executable from the EXECUTIVE. Programs on GO files can be run directly from the EXECUTIVE by giving the GO command and the file name in the form

—GO file name↵

[1] – See page 17 for another use of the REENTER command.

For example:

```
-GO ANALYSIS↵                    ANALYSIS is a GO file.
```

GO files can be created in several Tymshare languages, as shown in the table below.

Language	Procedure
SUPER BASIC	The user creates both binary and GO files from completed symbolic files by the single command `>SAVE BINARY file name↵`

BATCH FORTRAN IV	The command
	`+WRITE file name↵`
	saves a single program or overlay as a GO file, whereas the command
	`+DUMP file name↵`
	saves the entire overlay structure and shared memory.

SUPER FORTRAN	The user must save both binary and symbolic files in SFO, then return to EXECUTIVE to create the GO file, as follows:

```
-SFO↵

>LOAD SFOPROG↵
OK.
>SAVE SFPRG↵
TEXT ONLY?N↵
 NEW FILE↵
OK.
>Q↵

-MAKEGO↵
LINK FILE: SFPRG↵
GO FILE: FRISCO↵
 NEW FILE↵

-GO FRISCO↵
```

GO files can be identified with the DIRECTORY command, which is described on page 39. The file type of GO appears in the TYP column of the directory listing.

Section 4
CREATING AND DELETING FILES

A file is information stored on a disk.[1] Each Tymshare language has commands to create files. The additional wide range of commands for naming, printing, limiting, comparing, and manipulating files provides maximum flexibility for the user.

NAMING FILES

A file is identified by the name the user assigns when he creates it. A series of characters which satisfies the following restrictions is an acceptable file name.

Rule 1 A file name may contain any combination of the characters 0 through 9, A through Z, and @.

Rule 2 A file name may contain protected strings, that is, a series of characters enclosed in slashes or single quotation marks. Protected strings can contain any characters except line feed (J^c), carriage return (M^c), and the delimiting character itself (/ or '). *NOTE: To include a control A, control D, control Q, control V, or control W in a file name in the EXECUTIVE, the A^c, D^c, Q^c, V^c, or W^c must be preceded by control V (V^c) to specify that the particular character is to be accepted literally as input.*

Rule 3 Certain reserved file names cannot be used.

TERMINAL
TERMINA
TERMIN
TERMI
TERM
TER
TE
T

TELETYPE
TELETYP
TELETY
TELET
TELE
TEL
TE
T

These names always designate the terminal.

NOTHING
NOTHIN
NOTHI
NOTH
NOT
NO
N

These names designate a null file (see page 25).

[1] – Maximum file size is about 6 million characters.

Examples of acceptable file names are:

TEST	*Unprotected string.*
/DO NO^{c}T^{c}K^{c}NO^{c}W F^{c}IL^{c}D/	*Protected string.*
THIS/IS A/WILD‘ FILE!# ?’	*Combination of protected and unprotected strings.*
N@IL	

Examples of unacceptable file names are:

TEST#1	*# must be protected.*
STUDENT SURVEY	*Space must be protected.*
N	*Reserved file name.*
//JJ^{c}LP/	*J^c and second slash are illegal.*

Appending Comments to File Names

Comments can be appended to file names. The comments feature is especially useful because it allows the user to have brief, easy-to-use file names and still have the contents of the file documented. The file name and the comment are separated by a hyphen. The rules for comments are the same as the rules for naming files. For example, the file names above might appear like this when comments are appended to the names:

TEST-TUBE
/DO NO^{c}T^{c}K^{c}NO^{c}W F^{c}IL^{c}E/-‘CONTAINS SPECIAL INFO’

The comment does not change the name of the file and is not needed to call the file, but will appear in the directory listings. A comment can be appended when a file is first created, when a file is duplicated under a new name by the COPY command, or when a file name is changed by the RENAME command.[1] To append the comment TUBE to the existing name TEST, the user enters:

```
-RENAME TEST AS 1↵               Changes the name of the file.
-RENAME 1 AS TEST-TUBE↵          Changes the new name back to the old
                                 name with the comment appended.
```

The command

```
-RENAME TEST AS TEST-TUBE↵
```

is invalid since it tries to rename a file as itself.

Using Reserved File Names

Any of the reserved file names which are a subset of TELETYPE or TERMINAL can be used with the EXECUTIVE commands to indicate the terminal. For example, to enter text from the terminal to be stored on the file RHDA, the user types

```
-COPY T TO RHDA-/HIGGIN'S DATA/↵
```

and then enters information. When finished, he types a control D. Discussion of the COPY command with a full description of this procedure is presented on page 25.

[1] – The COPY and RENAME commands are discussed on pages 25 and 30, respectively.

Any of the reserved file names which are a left subset of NOTHING can be used with the EXECUTIVE commands to indicate a null file. To create a completely empty file, the user types:

```
-COPY NOTHING TO file name↵
```

The command

```
-COPY file name TO NOTHING↵
```

causes no action.

When a command is given in a language to write on NOTHING (or its subset), the WRITE command is executed but there is no file output. For example, if a SUPER BASIC program allows the user to specify an output file, the user can specify NOTHING to suppress file output. Likewise, he can specify TERMINAL or TELETYPE (or a subset) to print the output on the terminal instead of a file if the output device is specified for sequential rather than random output.[1]

CREATING AND LISTING FILES

The user may create, copy, or print out files while in EXECUTIVE with the COPY and TYPE commands.

The COPY Command

The COPY command is one of the most frequently used EXECUTIVE commands. It can be used to create files, to list the contents of files, and to copy the contents of one file to another.[2] The general form of the command is

```
-COPY source TO destination↵
```

The source and destination may be file names or the reserved names T, TE, etc., and N, NO, etc. The COPY command causes the contents of the source to be duplicated on the destination but does not affect the contents of the source.

If the destination is a file, a check is made to see if a file by that name already exists. The system prints either

```
OLD FILE
```

if the file name already exists, or

```
NEW FILE
```

if a new file name is being created. In either case, the system waits for the user either to confirm or to abort the command. The command is confirmed simply by typing a carriage return or a line feed. If the command is confirmed, the contents of an old file are completely erased and replaced by the contents of the source. To save the contents of the old file, the command can be aborted by typing an alt mode/escape.

In the following example, the user wants a copy of a file stored in the directory of user name ASHBY.[3] The EXECUTIVE warns that this command erases the present contents of ACCT, so the user types an alt mode/escape to abort the command and copies ACCT3 to a different file, ACC.

[1] – Refer to the *Tymshare SUPER BASIC Reference Manual* for a discussion of sequential and random files.

[2] – To copy files to another user's directory, refer to the discussion of the DIRIT program on page 87.

[3] – Copying files from another directory is discussed on page 28.

```
-COPY (ASHBY)ACCT3 TO ACCT↩
 OLD FILE⊕
-COPY (ASHBY)ACCT3 TO ACC↩
 NEW FILE↩

-
```

If the user types an N and a carriage return after OLD FILE/NEW FILE, the system asks for another file. For example:

```
-COPY DEMO TO MOD↩
 OLD FILEN↩
ERROR, TYPE: TO MOD2↩
 NEW FILE↩

-
```

If the user types a line feed after ERROR, TYPE: TO, the system prints an error message followed by another ERROR, TYPE:TO.

COPY can be shortened to the first three letters, and the TO can be replaced by a comma (,). For example, all the following forms of the COPY command perform the same function.

```
–COPY JEAN TO JN↩

–COP JEAN TO JN↩

–COPY JEAN,JN↩

–COP JEAN,JN↩
```

Creating a New File with the COPY Command

To create a new file by entering information from the terminal, the form of the command is

```
–COPY T TO file name↩
```

The EXECUTIVE prints OLD FILE or NEW FILE. The user gives the appropriate response. If the file name is acceptable, the user then enters the text or data. After the entire text is entered, the user types a control D. For example,

```
-COPY TERMINAL TO MOON↩
 NEW FILE↩

23456↩
345678↩
12345↩
Dc
```

```
-COP T,SUN↵
 NEW FILE↵

98765↵
Dc
-
```

As discussed under "Setting File Security Controls" on page 33, it is possible to protect a file from being erased. If a protected file is given as a destination in a COPY command, the error message

```
ERROR, TYPE: TO
```

is printed. The user simply enters a new destination.

A user can also create a file in a programming language or in Tymshare's EDITOR. In EDITOR, more than 20 different control characters are available to make creating files easier. Any Tymshare representative has copies of the *Tymshare EDITOR Reference Manual.*

Listing a File

The command forms used to print the contents of a file at the terminal are

```
–COPY file name TO T↵
```

or

```
–TYPE file name↵
```

Examples

```
–COPY (RUBIN)RP TO T↵

–COP MTEST,T↵

–TYPE ALPH2↵
```

NOTE: The printing of a file can be interrupted at any time by typing an alt mode/escape.

The TYPE command displays the contents of symbolic files at the terminal. The form of the command is

```
–TYPE file name↵
```

The TYPE command is equivalent to the COPY command with T as the destination file. For example, if ABC is a symbolic file, the following are equivalent:

```
–TYPE ABC↵

–COPY ABC TO T↵
```

Accessing Files

The user has complete control over the files in his directory. In addition, he can access another user's files if the other user permits. Therefore, files can be called from the user's own directory, from another directory in the same account, or from a directory in another account on the same system.

A file in the user's own directory can be used in a command by typing just the name of the file with or without any comment. For example, if the file is MTDB-AUGUST, it can be designated in an EXECUTIVE command as either MTDB or MTDB-AUGUST. In fact, any comment can be given after the file name. The comments are ignored. For example, MTDB-'RANDOM COMMENT' also designates the file MTDB. In other words, a file is designated by its file name. Therefore, in any directory there cannot be two files with the same name. For example, MTDB-AUGUST and MTDB-JULY cannot exist in the same directory.

The contents of a file in another user name in the same account can be accessed if it has been declared public.[1] The contents of a file in another account but on the same system can be accessed if the file name contains the character @, any legal control character (protected), or any lowercase character (unprotected). The file name is designated by showing in parentheses the user name of the directory in which the file exists, followed by the file name. For example:

(MILLER)@NEWS

(MILLER)SH@RING

(MILLER)/SHO^{c}R/

If the user named SMITH has a public file named NORTH, the other users in his account can obtain the contents of the file by typing

```
–COPY (SMITH)NORTH TO NORTH↵
```

The contents of the file named NORTH in user name SMITH are copied to a file named NORTH in this user name.

NOTE: A user can change the contents of another user's file only if it has been declared PUBLIC and WRITE ACCESS. Therefore, a file can be shared, but its integrity is maintained.

LIMITING NEW FILES

There is no limit to the number of files in a user's directory. However, for the user's protection, there is a limit to the number of new files that may be created after each log in. This limit has been imposed because, in most Tymshare languages, it is possible to write programs which create files. Through a programming error, a user could create many, many files without being aware of it. If a file directory becomes full, the user can obtain the maximum room for new files by logging out and logging in again. The user can also create more room by deleting files.

When the directory is full, the system continues to print the error message

ERROR, TYPE: TO

in response to any new file name entered. The file the user is attempting to write must therefore be saved by overwriting an old file. It is advantageous to have a scratch file that can be used as temporary storage. If this scratch file is given the name /$/, it can double as the fail-safe file in

[1] – Refer to the discussion of the DECLARE command on page 33.

case of premature disconnect.[1] Likewise, if the fail-safe controls have been set, the file /$/ is available as a scratch file.

The number of new files that can be created after each log in depends on the length of the file name, not the size of the files. If the file names are short (three characters or less), the maximum number of new files is more than 200.

DELETING FILES

The user has two EXECUTIVE commands for deleting files: DELETE, for removing files by name, and REMOVE, for deleting files by number.[2] It is economical to delete files when they are no longer needed.

The DELETE Command

The simplest and most direct way to delete files is with the DELETE command. The form of the command is

```
-DELETE file name↵
```

There must be a space after the word DELETE. To delete several files with one command, the file names must be separated by commas or spaces. For example:

```
-DELETE TEXT,NEWS↵
```

```
-DELETE TEXT NEWS↵
```

The list of files to be deleted can be extended to the next line by typing only a line feed or both a space and a carriage return.

The files are not deleted until a carriage return or line feed is typed. Therefore, the command can be aborted with an alt mode/escape any time before the carriage return or line feed. However, if the command uses several lines, the files on each line are deleted when the line feed or carriage return ending that line is typed.

If several names are listed and there is one which cannot be deleted, all the files up to that one will be deleted, but none of the files after it. For example:

```
-DELETE A,B,1,C,2↵

ERROR ON NAME:1?
```

Files A and B are deleted but 1, C, and 2 are not.

[1] – Refer to page 11 for a discussion of a fail-safe file.

[2] – The DIRIT program, described in Section 10, provides many ways of deleting files singly or by groups.

The REMOVE Command

The REMOVE command is most frequently used to delete files whose names contain forgotten nonprinting control characters. The REMOVE command deletes files by number rather than by name. The number is the file's position in the file directory. The numbers are obtained with the DIRECTORY command, described on page 39. For example:

```
-DIRECTORY,26↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
26 R/W NO  BIN   3-10  2  1536  SFPRG
27 R/W NO  SYM   3-10  1  1536  MOON
28 R/W NO  SYM   3-10  1  1536  SUN

-REMOVE 27↵

-DIRECTORY,26↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
26 R/W NO  BIN   3-10  2  1536  SFPRG
27 R/W NO  SYM   3-10  1  1536  SUN

-
```

NOTE: When file 27 is removed, file 28 becomes file 27.

The REMOVE command is more restricted than the DELETE command. Only one file can be removed at a time. If a user has used the GFD command to access another file directory, he cannot use REMOVE in that directory.[1]

RENAMING FILES

The RENAME command is used to change the name of a file in the form

```
–RENAME old file name AS new file name↵
```

A common use of this command is to rename files so they may be shared with other users. For example:

```
–RENAME FOR2 AS @FOR2↵

–
```

The command is also useful to create file names with comments attached. For example, to attach a comment to a file named XB, the user types

```
–RENAME XB AS 1↵

–RENAME 1 AS XB-'UPDATE PROGRAM'↵
```

RENAME cannot change a file name to a name that is already in use. For example, a file named JUNEDATA exists in the file directory.

[1] – GFD is described on page 38.

```
-RENAME JDATA AS JUNEDATA↵
ERROR, TYPE: NEW NAME: JULYDATA↵      The name JUNEDATA is already in use.
```

The RENAME command has a shortened form: The AS can be replaced by a comma, and RENAME can be shortened to REN.

Section 5
FILES AND THE FILE DIRECTORY

The EXECUTIVE has commands to set the security controls on individual files and on the entire file directory. They determine who can access the files and the directory, who can read files, and who can write on them. In combination with the CIPHER command, which is used to encode files, these controls provide a maximum level of security. The EXECUTIVE also provides a variety of commands by which the user can print out file names and information about files in his own directory, as well as in other accessible directories.

SETTING FILE SECURITY CONTROLS

The DECLARE command is used to set the security controls on specific files.[1] DECLARE asks two sets of questions: PRIVATE and PUBLIC. The private controls limit what the user can or cannot do to his own files. The public access controls restrict use of the files by other users in the same account.[2] The form of the DECLARE command is

```
–DECLARE↵

FILE(S): file names↵
```

or

```
–DECLARE file names↵
```

The user specifies the file or group of files which he wishes to declare. After the last file name, he types a line feed or carriage return. EXECUTIVE then asks questions which can be answered yes or no, Y or N, followed by a line feed. One question, WRITE ACCESS?, can also be answered with an A. These questions and the effect of the user's responses are listed in the following table.

[1] – See page 36 for the procedure for setting controls on the whole directory.
[2] – See page 36 for the procedure for sharing files with users in other accounts.

DECLARING FILE SECURITY STATUS

Question	Response	Effect
PRIVATE:		
WRITE ACCESS?	*Y	The user can write on the file, rename, or delete it.
	N	The file can only be opened for input; the user can neither write on it, rename it, nor delete it. The next question is bypassed.
	A	The user can add to the file, but cannot write over existing information. Append-only files cannot be deleted or renamed.
READ ACCESS?	*Y	The user can read the file.
	N	The user cannot read the file. The file cannot be opened for input or loaded into a language.
PUBLIC:		
READ ACCESS?	Y	Other users in the same account can access the file.
	*N	Other users cannot copy or use the file unless the file name contains a control character or @. All subsequent questions are bypassed.
WRITE ACCESS?	Y	Other users in the same account can write on the file. If the file is private append only, other users can append to the end of the file.
	*N	Other users in the account cannot write on it or delete it.
PROPRIETARY?	Y	If accessible, the file can be executed by other users but cannot be listed or copied. It can be accessed only by the GO command if it is a GO file, or the RUN command if it is a DUMP file. Memory is cleared whenever control is returned to the EXECUTIVE.
	*N	No further limitation is placed on copying or using.

*A newly created file has the security status marked by asterisks.

In the following example, SORT and CATALOG are GO files belonging to user Jones (user name JONES). He declares these files to be proprietary as follows:

```
-DECLARE↵
FILE(S): SORT,CATALOG↵
PRIVATE:
 WRITE ACCESS? Y↴
 READ ACCESS? Y↴
PUBLIC:
  READ ACCESS? Y↴
 WRITE ACCESS? N↴
PROPRIETARY? Y↵

-
```

The user follows each response with a line feed to continue with the next question.

The files SORT and CATALOG can be read, written on, or deleted by Jones. In addition, all other users in the same account can use either file by typing

```
-GO (JONES)file name↵
```

No user can write on or delete either file.

If the Y or N is followed by a carriage return rather than a line feed, DECLARE skips the remaining questions and makes no further changes in the condition of the file or files. For example, Jones can control his file SEEB so that even he cannot erase it; he can append data to the end of the file only.

```
-DECLARE SEEB↵
PRIVATE:
  WRITE ACCESS?A↵


-
```

Append-only data files simplify programming when information is repeatedly added to a file.

The inclusive ALL may be used to refer to every file in the user's directory. This allows a company to set aside one user name as an "account library" by declaring all files under that name to be public to the account. For example:

```
-DECLARE ALL↵
PRIVATE:
 WRITE ACCESS? N↴     The N reply causes EXECUTIVE to bypass the READ ACCESS question. The
PUBLIC:               private READ ACCESS status remains unchanged.
  READ ACCESS? Y↴
 WRITE ACCESS? Y↵     The user types a carriage return after his last reply to indicate that the remaining
                      question need not be asked; that is, he does not wish to change the proprietary
                      status of his files.
-
```

The user can determine the security controls on a file by using the

```
-DIRECTORY file name↵
```

command, which is discussed on page 39.

THE CIPHER PROGRAM

With the CIPHER program, a user can encode a file into a form which cannot be decoded without the key word. An encoded file can be stored on magnetic tape, paper tape,[1] or in a public file with complete security. When the user wishes to use the file again, he simply calls CIPHER and gives the key word that was used to encode the file. For example:

```
-CIPHER↵        The user calls CIPHER.

ENTER KEY:↵     The user types a nonprinting key word. The key does not print so there is no written record.
AGAIN:↵         He types the key again.

TYPE C TO CIPHER, U TO UNCIPHER, C.R.: C↵     The user wants to encode.

INPUT FILE: CBIN↵       The file CBIN is to be encoded.

OUTPUT FILE: ABIN↵
 NEW FILE↵
```

[1] – The library program BINTAPE can be used to punch a binary tape. For more information on BINTAPE, refer to the *Tymshare TYMCOM-IX Paper Tape Package Reference Manual.*

```
END JOB

-DELETE CBIN↵                 The user deletes the original version of the file.

-
```

To decode the file, the procedure is as follows:

```
-CIPHER↵

ENTER KEY:↵                   The user enters the same key word.
AGAIN:↵                       He enters it again.

TYPE C TO CIPHER, U TO UNCIPHER, C.R.: U↵

INPUT FILE: ABIN↵

OUTPUT FILE: CBIN↵
 NEW FILE↵

END JOB

-
```

MAKING A FILE PUBLIC

A file with an @, any control character, or any lowercase character in its name is always public. It can be accessed by any user on the system, if he knows the full name of the file, by typing

—COPY (user name)file name TO file name↵

Therefore, the user can provide an extra measure of security for a file that must be shared by including nonprinting control characters in the file name and informing only those users who are allowed to access the file. Since the control characters do not print, there is never any written record of the complete file name to jeopardize its security.

SETTING FILE DIRECTORY SECURITY CONTROLS

In addition to the file sharing option, the Tymshare system has powerful file directory sharing options for users in the same account.[1] These options are exercised with the three commands FDC (File Directory Controls), PFDC (Print File Directory Controls), and GFD (Get File Directory). The FDC and PFDC commands set the directory security status and print the present status. The GFD command accesses another user's file directory.

The file directory controls allow several user names to share the same directory. These controls also allow several user names to add files to a directory such as the account library without having access to the directory.

[1] – The Account Supervisor automatically has access to all directories in his account. See the *Tymshare Account Supervisor Manual* for a discussion of the Supervisor's options.

The File Directory Controls (FDC) Command

Directory access by users in the same account is controlled with the File Directory Controls command, FDC.

The following options are available with the FDC command:

- SHARABLE? Other users within the same account can access the directory and files within the directory, with the GFD command.
- LISTABLE? Other users can list the file directory.
- CONTROLS? Other users can use the DECLARE command to change the security status of files.
- NEW FILES? Other users can create files in the directory.

The FDC command asks for each of the above options, and they must be answered with Y or N followed by a line feed or carriage return. A line feed asks for the next question; a carriage return bypasses the remaining questions and makes no further changes in the file directory controls. These questions and the effect of the responses are listed in the table below.

EFFECT OF RESPONSES TO FDC COMMAND OPTIONS

Question	Y (for YES)	N (for NO)
SHARABLE?	All users in the account may access the files in this directory (run programs, list files, etc.). Access is subject to the private controls on the files set by the owner using the DECLARE command.	Only the owner and the Account Supervisor have access to the files in this directory. *NOTE: If the answer is N, the LISTABLE and CONTROLS options are bypassed.*
LISTABLE?	All users who can access this file directory can use the FILES and DIRECtory commands.	Other users cannot use the FILE or DIRECTORY command to list the directory contents.
CONTROLS?	All users who can access this file directory can reset file controls on files in this directory with the DECLARE command.	Other users cannot use the DECLARE command on any of the files in this directory.
NEW FILES?	Files can be put into this directory by any user in the account.	Files can be put into the directory only by the owner of this directory.

Example

```
-FDC↵
SHARABLE? Y↴
  LISTABLE? Y↴
  CONTROLS? N↴
NEW FILES? N↵

-
```

Other users can run the programs and list the files and file directory, but they cannot change the public and private controls or create new files.

```
-FDC↩
SHARABLE? N↴
NEW FILES? Y↩

-
```

Other users can add new files but cannot use the directory. This is a convenient status for an account library.

When a user name is first created, its directory is declared LISTABLE only; no other user except the Account Supervisor can access it.

The Print File Directory Controls (PFDC) Command

If a user wishes to know what controls are in effect for his directory, he types PFDC for Print File Directory Controls. All options that were answered with a Y in the FDC command are listed. If the option is not listed, it has been answered with N and is not in effect. For example:

```
-FDC↩
SHARABLE? Y↴
  LISTABLE? Y↴
  CONTROLS? N↴
NEW FILES? Y↩

-PFDC↩
SHARABLE  LISTABLE
NEW FILES
-
```

Since CONTROLS? was answered with N, it is not shown in the PFDC command.

The Get File Directory (GFD) Command

The GFD command can be used to access other user directories in the same account if the SHARABLE option in FDC has been answered with Y by the user whose directory is to be accessed. The form of the command is

–GFD user name↩

The user now has access to all files in the directory of that user name. To re-access his own files, he must issue a GFD command specifying his own user name.

If the user specifies a directory in a GFD command that is sharable but not listable, issuing the FILES or DIRECTORY command results in an error message and a question mark being printed. If the directory has protected control, issuing the DECLARE command will result in the printing of a question mark. For example:

```
-GFD DOC↵

-PFDC↵
SHARABLE

-FILES↵

NOT LISTABLE
 ?

-DIRECTORY↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
NOT LISTABLE
 ?

-DECLARE↵
 ?

-
```

The GFD command allows more than one user to access the same file directory simultaneously, thus avoiding the excess storage that would result from maintaining duplicate file directories.

LISTING FILE INFORMATION

The EXECUTIVE has several commands to list file information. The LIST command lists the names of the files, DIRECTORY prints several items of information about the files, and the FILES command lists selected information about the files. In addition, the LAST command gives the number of files in the directory. The DIRIT program, described on page 87, also includes file listing capabilities.

The LIST Command

The user types

```
–LIST↵
```

to list all the file names in the file directory beginning with the file most recently created.

The DIRECTORY Command

All the characteristics of the user's files may be listed with the DIRECTORY command. Most recently created files are listed last. The listing can be stopped at any time by typing an alt mode/escape. The DIRECTORY command lists the following information for each file:

- File number
- Private controls
- Public controls

- File type
- Date of last write
- Number of times accessed since created
- File size in characters
- File name

For example:

```
-DIRECTORY↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
 1  R  NO  SYM   2-12 53  1536  ADDITIONS
 2  W  NO  SYM   2-12 37  1536  UPDATE
 3  R  NO  SYM   3-9  43  1536  LOOP
 4 R/W YES ,WT SYM   1-12 21  1536  SBASAVINTRST
 5  R  NO  SYM   2-10  6  3072  BUDGET
 6 R/W YES ,WT BIN   3-9   6  1536  SFOPROG
 7 R/W YES ,WT GO    2-12  3  1536  ACCTS
 8 R/W PRP ,WT GO    2-12  1  1536  SORT
 9 R/W YES ,WT GO    2-12  1   768  REPT1
10 R/W YES ,WT GO    2-12  1   768  REPT2

-
```

The file number is simply the position of the file in the directory. As files are deleted from the directory, the numbers of files farther down the list are adjusted. The file number is used with the REMOVE command, described on page 30.

The private controls and public controls are set by DECLARE. In the PVT column:

R = Read access only.
W = Write access only.
R/W = Read and write access.
AP = Appendable only.
R/A = Appendable with read access.

In the PUB column:

NO = Not public.
YES = Public.
YES,WT = Public with public write access.
PRP = Proprietary.
PRP,WT = Proprietary with public write access.
INT = Initialized.[1]

[1] – Initialized files are discussed starting on page 48.

The file type is one of the following:

SYM Symbolic files contain information in the standard alphanumeric character representation. They may be used as program files or data files.

BIN Binary files are written in machine code. Compiled programs and data may be stored on binary files for security and economy, since they usually take less storage space than symbolic files and are faster to load.

GO A GO file is a program file which may be executed directly from the EXECUTIVE by typing GO and the file name. See page 20 for an explanation of GO files.

DUM A DUMP file is a machine code file created by the DUMP command as discussed on page 18. A DUMP file can be executed with the RUN command.

BAD If a file is ever listed as a bad file, some unusual error has rendered the file unusable. The user should call the Tymshare representative to have a good version of the file restored from the backup tapes.

The date (month-day) is the last time the file was written on. If the data is more than a year old, the year appears in the parentheses after the day.

The use is the number of times since it was last created that the file has been accessed for either reading or writing.

The size is the file size given in characters.

The full name of the file is listed with any appended comments.

By typing a line feed instead of a carriage return after the DIRECTORY command, the user can specify one file or a group of files for which he wants all the characteristics to be listed. For example:

```
-DIRECTORY↴
FILE(S): UPDATE,LOOP,BUDGET↩
PVT PUB TYP  DATE USE  SIZE  NAME
 W  NO  SYM   2-12 37  1536  UPDATE
 R  NO  SYM   3-9  43  1536  LOOP
 R  NO  SYM   2-10  6  3072  BUDGET

-
```

File numbers are not printed if the user specifies the file names.

Alternatively, the DIRECTORY command and the file list can be given on one line:

```
—DIRECTORY PGM,BL↩
```

or

```
—DIR PGM BL↩
```

Another form of DIRECTORY allows the user to list part of the directory. Its form

```
—DIR,file number↩
```

lists the files beginning with the number specified. For example,

```
-DIR,24↵
```

lists all the files from number 24 to the end of the directory. If the DIRECTORY command and the file list or file number are given on one line and followed by a line feed, instead of a carriage return, the column headings are not printed. For example:

```
-DIR UPDATE,LOOP,BUDGET↵
PVT PUB TYP  DATE USE  SIZE  NAME
 W  NO  SYM   2-12 37  1536  UPDATE
 R  NO  SYM   3-9  43  1536  LOOP
 R  NO  SYM   2-10  6  3072  BUDGET
```

The user types the DIRECTORY command followed by a carriage return.

```
-DIR UPDATE,LOOP,BUDGET↴

 W  NO  SYM   2-12 37  1536  UPDATE
 R  NO  SYM   3-9  43  1536  LOOP
 R  NO  SYM   2-10  6  3072  BUDGET

-
```

The user types the DIRECTORY command followed by a line feed.

The FILES Command

The FILES command can be used to list particular directory information.[1] For a quick listing composed only of file names and file types, the FILES command is given as shown below.

```
-FILES↵

SYM   ADDITIONS
SYM   UPDATE
SYM   LOOP
SYM   SBASAVINTRST
SYM   BUDGET
BIN   SFOPROG
GO    ACCTS
GO    SORT
GO    REPT1
GO    REPT2
GO    REP
BIN   NAUT ⊕
```

By typing a line feed after the FILES command, the user can specify the quick listing of one file or a group of files.

```
-FILES↴
FILE(S): CON2,BITTY,COM1↵

SYM   CON2
BIN   BITTY
SYM   COM1

-
```

[1] – Particular file information can also be listed with the DIRIT program described in Section 10, page 87.

Alternatively, the file names can be typed on the same line as the FILES command.

```
-FILES CON2,BITTY,COM1↵

SYM   CON2
BIN   BITTY
SYM   COM1

-
```

In addition, by typing a line feed after the specified file names, the user may select which other characteristics besides the name and type he wants to see. For example:

```
-FILES↴
FILE(S): ACCTS,SUN↴
NUMBER? N↴
SIZE? Y↴
DATE? Y↴
CONTROLS? Y↴
PVT PUB TYP  DATE USE  SIZE  NAME
R/W YES ,WT GO    2-12  3  1536  ACCTS
R/W YES ,WT SYM   3-10  1  1536  SUN

-
```

As is shown in the preceding example, the user types Y for YES if he wants to list the information in question; otherwise, he replies with N for NO. Either a line feed or a carriage return may be typed after a Y or N reply. A line feed causes FILES to ask the next question; a carriage return causes it to skip the remaining questions. For example:

```
-FILES LOOP,SORT↴
NUMBER? N↴
SIZE? N↴
DATE? Y↵
TYP  DATE USE  NAME
SYM   3-9  43  LOOP
GO   2-12   1  SORT

-
```

The Y reply causes both DATE and USE to be listed. The carriage return causes the next question to be skipped.

The name ALL lists all the files in the user's directory. Therefore, the user can select information to be given about all of the files by answering

FILE(S): ALL↵

If the FILES command is followed by a comma and a file number, information is listed for that file and all subsequent files. For example:

```
-FIL,25⤵
NUMBER? Y⤵
SIZE? Y↵
 # TYP   SIZE  NAME
25 SYM   1536  @CATLG
26 SYM   1536  @SANDEN
27 SYM   1536  @LATER
28 SYM   1536  @DUMMY

-
```

This feature is especially useful with large file directories.

The LAST Command

For a large file directory, the user can easily find the number of the last file in the directory with the LAST command, then request a file listing or complete directory information about only the most recent files. For example:

```
-LAST↵
28
-FIL,25↵
```

or

```
-DIR,25↵
```

The SUMMARY Command

The SUMMARY command prints the user's total current file storage in number of characters and total number of files in the user's directory. For example:

```
-SUMMARY↵

3/10/76  17:31

TOTAL STORAGE:      89856

NO. OF FILES:          28

-
```

Section 6
AUTOMATIC FILE FEATURES

The TYMCOM-IX system has several automatic file features to facilitate data processing: use of command files, terminal output files, and file initialization.

COMMAND FILES

Under normal operation, commands are entered into the system from the terminal. Occasionally, however, it may be advantageous to store a particular sequence of commands on a file using the command file feature.

The command

```
–COMMAND file name↵
```

causes the system to accept commands from the specified file as if they had been entered from the terminal.[1] The command file feature is convenient whenever:

- An elaborate system of programs needs to be linked into one easy to use package.
- Clerical functions are performed on the system by employees with no need or desire to learn the Tymshare commands.
- A sequence of commands must be performed repeatedly, such as each time a program is run.

For example, a company keeps a catalog of its publications on a file. The SUPER BASIC program UPDATE adds new entries to the file, but for safety, a copy is made of the data file each time before the program is run. Therefore, the command sequence entered from the terminal is the following:

```
-COPY CATLG TO CATDUP↵
 NEW FILE↵


-SBA↵

>LOAD UPDATE↵
>RUN↵
```

[1] – The DO command is equivalent to the COMMAND command.

To perform the same operations, a command file named ADDITIONS contains the following commands as text:

```
COPY CATLG TO CATDUP

```

The space represents a carriage return entered to confirm NEW FILE.

```
SBA
LOAD UPDATE
RUN
COMMAND T
```

After the program UPDATE is completed, control is returned to the terminal by the command COMMAND T.

Then the program can be run by typing

–COMMAND ADDITIONS↵

The commands in the file are not printed on the terminal but any messages from the system are printed, such as OLD FILE. The command file may be created either in the EXECUTIVE, with the COPY command, or in EDITOR.[1]

The commands in a command file may be from any Tymshare language and many library programs. Users can also write programs which can be run from command files. Note, however, that input files for SBA programs cannot be specified in command files.

The system accepts its commands from the file specified in the COMMAND command until one of the following items is reached:

- A COMMAND T command, which causes the system to return to accepting commands from the terminal.
- The end of the command file, which has the same effect as the item above except that a question mark is printed.
- Another COMMAND command, which enables the user to link as many command files as he wishes. Command files can also link to themselves; that is, the last command in the file can be a command to take commands from itself. See the example below.
- An error.

Example

A user wants to delete all but his first 29 files. He creates a command file which removes file number 30 and calls itself to remove file number 30 again. All the files are deleted until only the first 29 remain. Control returns to the terminal when the REMOVE 30 command can no longer be executed.

[1] – Refer to the *Tymshare EDITOR Reference Manual.*

```
-COPY T,FILDEL↵
 NEW FILE↵
```

The user uses the COPY command to create the file.

```
REMOVE 30↵
COMMAND FILDEL↵
```

There are only two statements in the command file.

```
Dc
```

D^c terminates the COPY command.

```
-COMMAND FILDEL↵
 ?
```

Control is transferred to FILDEL.

The question mark prints when there is no longer a 30th file to be deleted.

```
-LAST↵
29
-
```

Only 29 files remain.

DIRECTING TERMINAL OUTPUT TO A FILE

It is frequently useful to direct terminal output to a file instead of to the terminal. The TOUT command provides this capability. For example, a user may wish to store on a file the output of a program as it is running, or it might be convenient to document terminal sessions by directing output from EXECUTIVE commands to a file. The form of the command is

–TOUT file name↵

The TOUT file is a sequential output file. As long as the TOUT file is open, new information is added to the end of the file. If a TOUT file is closed, then opened again, the first contents of the file are erased, and the new information replaces the former contents of the file. The user can avoid erasing a TOUT file by specifying a different file name in the subsequent TOUT command.

To return to the normal mode in which EXECUTIVE output is written on the terminal, the TOUT command is reexecuted with T as the file name. For example:

–TOUT T↵

Terminal output from programs or languages is directed to the terminal unless the user specifies a TOUT file. Calling a program or a language does not deactivate the TOUT file. The TOUT file remains open until the user redirects terminal output to the terminal. The TOUT file cannot be copied to the terminal, read into EDITOR, or opened as a data file in a language program until it is closed by a TOUT T command or a program.

```
-TOUT↵
 TO FILE: TRIAL↵
 NEW FILE↵
```

The user calls TOUT and specifies the file name TRIAL.

```
-DATE↵
```

The user requests the date. The system writes the date on the TOUT file TRIAL and gives the prompt.

```
-TOUT T↵

-TYPE TRIAL↵
```

The user leaves the TOUT mode and prints the contents of the file TRIAL.

```
3/10/76  17:55
-
```

THE " (COMMENTS) COMMAND

The double quotation mark (") lets the user type comments on the work he is doing. The user enters a double quotation mark followed by the desired comments, and terminates the entry with a control D or another double quotation mark and a carriage return. The form of the command is

—"**comments**D^c

or

—"**comments**" ↵

For example:

```
-"WE WILL NOW CALL SBA AND LOAD A PROGRAM THAT CALCULATES↵
THE AREA OF A TRIANGLE.Dᶜ
-
```

or

```
-"WE WILL NOW CALL SBA AND LOAD A PROGRAM THAT CALCULATES↵
THE AREA OF A TRIANGLE."↵

-
```

The EXECUTIVE ignores all text that is contained in comment form, but the text will be printed on the terminal if the comment is contained in a COMMAND file. For this reason, comments are particularly useful in command files to notify the user of progress when the file is running.

INITIALIZED FILES

When a command file is initialized, control is transferred to the file immediately after the user logs in. The command to initialize a command file is

—**INIT file name** ↵

For example, a company has a user name containing its inventory and inventory control programs. This user name is designed so that the control programs are loaded automatically by a command file called CONTROL shown below:

```
-COPY T TO CONTROL↵
 NEW FILE↵
```

The user uses the COPY command to create the file.

```
SBA↵
5 PRINT↵
10 PRINT "UPDATE OR RETRIEVE":↵
20 INPUT A↵
30 IF LEFT(A,1)="U"THEN A="UPDT"↴
   ELSE A="RETR"↵
```

The file calls SUPER BASIC and runs a program that asks what the user wants to do, and then writes a second command file which loads the appropriate program.

```
40 OPEN "CON2",OUTPUT,1↩
50 PRINT ON 1:"GO "+A↩
55 PRINT ON 1:"COM T"↩
60 CLOSE 1↩
```

The file runs the short program and transfers control to a second command file.

```
RUN↩
QUIT↩
COMMAND CON2↩
D^c
-INIT CONTROL↩
```

The user initializes the file.

```
-
```

Then, when a user logs in, the following occurs:

```
please log in: DELEON;;↩

TYMSHARE   C2 3/10/76  18:10

UPDATE OR RETRIEVE? R↩

INVENTORY INFORMATION RETRIEVAL PROGRAM
```

An initialized file continues to execute automatically each time the user logs in or until the file is deinitialized by giving the DEINIT command described below. If an initialized file is deleted without deinitializing it, and another file is not initialized, a question mark occurs every time the user logs in. The user may avoid this question mark by giving the DEINIT command. Initialized command files are listed as INT under the PUB heading in the file directory listing.

The user can also initialize a Tymshare Library program or a file in another user name. The correct form is

–INIT (user name)file name↩

or

–INIT #lp↩

where *lp* stands for any Tymshare Library program.

When a Tymshare Library command file or a file from another user name is initialized, an empty dummy file is set up in the user's directory with the same name as the file. These dummy files cannot be removed from the directory except by use of another INIT or the DEINIT command.

GO and DUMP files, as well as command files, can be initialized. If a GO file is initialized, the program starts running immediately after logging in. If a DUMP file is initialized, the file is automatically recovered and transfer is made to the language, but the program does not execute automatically.

Normally it is possible to stop initialized files by typing an alt mode/escape. However, the alt mode/escape can be disabled so that it will not interrupt a program. To disable alt mode/escape, the file is initialized by giving the INIT command followed by a line feed and then the file name. For example:

```
–INIT↴

FILE CONTROL↵

–
```

Files may be deinitialized by issuing the DEINIT command or by initializing another file in the directory. Dummy file names will be removed from the directory by the DEINIT command or a subsequent INIT command, but ordinary initialized files will remain. The DEINIT command does not take a file name. Its form is simply

```
–DEINIT↵
```

NOTE: Files that have been initialized by an Account Supervisor can be deinitialized only by the Account Supervisor.

Section 7
UTILITY COMMANDS

There are several utility commands available on the TYMCOM-IX system that provide system information to document a session at the terminal: DATE, TIME, SYSNO, and DSC.

THE DATE COMMAND

The DATE command prints the current date and the time of day. For example:

```
-DATE↵
3/10/76  18:13
-
```

THE TIME COMMAND

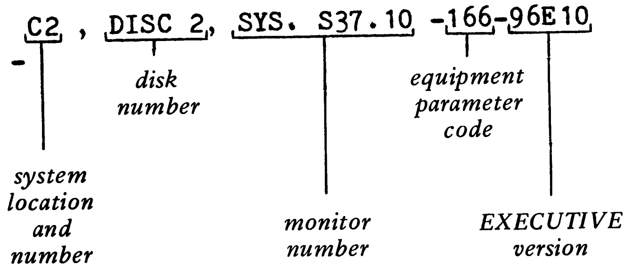
The TIME command prints the computer time used and the terminal connect time since the user logged in. For example:

```
-TIME↵

CPU TIME: 4 SECS.
TERMINAL TIME: 0:2:49

-
```

THE SYSNO COMMAND

The SYSNO command prints the computer location and number, the disk number, the system monitor number, and the EXECUTIVE version. For example:

```
-SYSNO↵
```



```
C2 , DISC 2, SYS. S37.10 -166-96E10
-
```

Location Code	Location
C	Cupertino, California
H	Houston, Texas
P	Paris, France
V	Valley Forge, Pennsylvania

The Tymshare analyst may use this information to help users with problems.

THE DSC COMMAND

The DSC command prints the storage currently being used and the maximum storage used (in blocks of 768 characters) since the last storage measurement. For example:

```
-DSC↵
DISC STORAGE (BLOCKS):
BASE = 2, MAX = 16
```

Section 8
FEATURES FOR NARP AND XDDT USERS

RUNNING A DEBUGGED PROGRAM

The SAVE command creates a GO file. The form of the command is

—SAVE first loc TO last loc ON file name↩

SAVE responds with the OLD FILE or NEW FILE message.

If either the command or the OLD FILE/NEW FILE is terminated by a line feed rather than a carriage return, SAVE asks

STARTING LOCATION

The user enters the starting location of the program when it is again placed in core. The starting location must be followed by a carriage return.

GO files can be saved so that they remain open when they are loaded. This allows the user to append data to the end of the program file.

To save a GO file so that it remains open, the user sets the sign bit when he enters the starting address. This is done by preceding the starting address with 4 and enough zeros to fill the number out to eight digits.

```
-SAVE 240 TO 307 ON PRESNI↩
 NEW FILE↴
STARTING LOCATION 40000240↩
```

To restore a program that has been saved on a file without calling XDDT, the user types

—PLACE file name↩

If a starting location was specified when the program was saved, the program can be run without calling XDDT by using the BRANCH command.

—BRANCH starting location↩

COMMANDS TO DETERMINE MEMORY ALLOCATION

Command	Information
MEMORY	Number of words of unused memory.
STATUS	Status of used bytes.
PMT	Status of all bytes allocated to user.

COMMANDS TO RELEASE MEMORY

Command	Action
RELEASE	Releases the subsystem.
KILL	Kills program relabeling.
RESET	Returns all of user's memory.

Section 9
PROGRAMS FOR FILE MANIPULATION

This section describes several Tymshare Library programs. The File Directory Management (FDM) program reduces the size of files to save storage charges and puts any number of them on one master file. CHECKSUM, VERIF, COMPARE, and SCOMPARE can be used to determine if a file has been changed.

THE FILE DIRECTORY MANAGEMENT (FDM) PROGRAM

The FDM program crunches symbolic files into smaller binary files and places them on one master file which can later be uncrunched back into individual files.[1] FDM is useful for grouping files with similar functions, or files of one program, for convenient storage. It is also handy for reducing storage charges for symbolic files that are not used often. The binary files created by FDM cannot be understood by the system; they must be uncrunched before they can be used. The program is called by typing

```
-FDM↵
```

The Basic Commands

The basic operation of FDM is as follows: Symbolic files are crunched into binary files and placed in a buffer. They are then written onto a master FDM file, any number of files to one master file. Files are restored to symbolic type in two steps also. First the master file is opened for reading, and one file at a time is read into the buffer. Each file is then uncrunched and written to the user's directory as a symbolic file with its old name, or with a new name, at the user's discretion.

In this section, the basic operation is discussed first, then the advanced features are discussed in detail.

The CRUNCH and MASTER Commands

The CRUNCH command crunches the specified file and places it in the FDM file buffer. The MASTER command takes the file from the FDM buffer and writes it on an FDM master file. For example, to crunch a file named TEXT and create the binary file SMALTEXT, the user proceeds as follows:

[1] – Binary or GO files can also be placed on a master file but their size remains unchanged; see page 63.

```
-FDM↵


:CRUNCH↵
FROM:TEXT↵
COMPRESSED TO 75%

:MASTER↵
TO:SMALTEXT↵
 NEW FILE↵
28 WORDS

:
```

The short forms of the commands are as follows:

```
-FDM↵


:CRUNCH TEXT↵
COMPRESSED TO 75%

:MASTER SMALTEXT↵
 NEW FILE↵
28 WORDS

:
```

NOTE: After a file has been crunched and stored, the symbolic version of the file can be deleted with the EXECUTIVE DELETE command.

Several symbolic files may be crunched and written into the same binary file by repeating the MASTER command without a file name. The commands below crunch files ALPH2, ALPH3, and ALPH4, and sequentially write the crunched files on the master file SHRINK.

```
-FDM↵


:CRUNCH ALPH2↵
COMPRESSED TO 76%

:MASTER SHRINK↵
 OLD FILE↵
24 WORDS

:CRUNCH ALPH3↵
COMPRESSED TO 77%

:MASTER↵
27 WORDS
```

```
:CRUNCH ALPH4↩
COMPRESSED TO 76%

:MASTER↩
31 WORDS

:
```

NOTE: If the user wishes to save a crunched file on paper tape, he should crunch the symbolic file onto a master FDM file and then use the EXECUTIVE Tape Package (see page 153) to save the master file on tape.

The READ and UNCRUNCH Commands

To uncrunch a crunched file, the master file is accessed with the READ command, then the individual file is restored with the UNCRUNCH command. For example:

```
-FDM↩


:READ SMALTRIAL↩        The user reads the file SMALTRIAL.
FILE: TRIAL
(LAST FILE)             TRIAL is the last file on SMALTRIAL.
3/10/76  17:55          FDM prints the first line of TRIAL, which contains a date and time.

:UNCRUNCH TRIAL↩        This command uncrunches the binary file and writes it on the symbolic file TRIAL.
 OLD FILE↩
14 CHARACTERS

:Q↩                     Q returns control to the EXECUTIVE.
```

Each time the READ command is given without a file name, it reads the next file from the current open master file. For example:

```
-FDM↩


:READ SHRINK↩
FILE: ANAL2
THIS IS A TEST

:READ↩
FILE: SUN
98765

:READ↩
FILE: CON2
(LAST FILE)
COM T

:
```

The UNCRUNCH command uncrunches the last file read in from the master file and writes the symbolic text on the specified file. If no write file is specified, the system asks TO:, and the user responds with the desired file name. The user may terminate the UNCRUNCH command or respond to the request TO: with a line feed which writes the uncrunched text to the file name originally associated with it. For example:

```
-FDM↵

:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST
```

The user reads but does not uncrunch the first file on master file SHRINK.

```
:READ↵
FILE: SUN
98765
```

The user reads the second file on master file SHRINK, uncrunches it, and writes the uncrunched file on BIGGER.

```
:UNCRUNCH BIGGER↵
 NEW FILE↵
6 CHARACTERS
```

The user reads the third file.

```
:READ↵
FILE: CON2
COM T
```

The system prompts for the name of the write file.

```
:UNCRUNCH↵
TO:BIGGEST↵
 NEW FILE↵
6 CHARACTERS

:READ↵
FILE: ADDITIONS
COPY CATALOG TO EXTRA
```

The user terminates the UNCRUNCH command with a line feed, and so the file ADDITIONS is uncrunched and written with its original file name.

```
:UNCRUNCH↴
FILE: ADDITIONS
 OLD FILE↵
55 CHARACTERS

:READ↵
FILE: UPDATE
100 S=0
```

The user responds to TO: with a line feed, and so UPDATE is written with its old file name.

```
:UNCRUNCH↵
TO:↴
FILE: UPDATE↵
 OLD FILE↵
294 CHARACTERS

:Q↵

-
```

NOTE: When files are uncrunched and restored to the directory as symbolic files, they are accepted as new files. That is, their creation date becomes the date they are uncrunched, and the usage record starts over at 1.

It is also possible to uncrunch a file and write it to its old file name, from any position in the open read file, simply by typing

```
:UNCRUNCH file name↵
```

For example, the master file SHRINK contains the following files:

```
:FILES↵

  1 SYM ANAL2
  2 SYM SUN
  3 SYM CON2
  4 SYM ADDITIONS
  5 SYM UPDATE
  6 SYM LOOP

:
```

Suppose that the user has uncrunched the first file, and is now at the second file in the directory. However, he wishes to skip files 2, 3, and 4, then uncrunch the fifth file.

```
:UNCRUNCH UPDATE↵
 OLD FILE↵
15 CHARACTERS

:Q↵
```

NOTE: To read and uncrunch a crunched file that has been stored on paper tape, the user should use the EXECUTIVE Tape Package (see page 153). The user can then restore the file to its original form using the FDM UNCRUNCH command.

Creating Master Files

The advanced features of FDM allow the user much greater flexibility in crunching and storing files. Files can be crunched and written to the master file one at a time, the whole directory or any range of files can be crunched and written at one time, selected files can be conveniently written with one simple command, or files can be crunched by type.

CRUNCH with MASTER or APPEND

After calling FDM, the user crunches a file with the CRUNCH command, then writes it to a file with the MASTER command.

```
-FDM⮐


:CRUNCH TRIAL⮐
COMPRESSED TO 76%

:MASTER LITTLE⮐
 NEW FILE⮐
31 WORDS

 :
```

The user writes a crunched file onto a new master file, which remains open for subsequent output.

The first use of the MASTER command opens the new master file and leaves it open for subsequent MASTER commands. More files can be added to the master by crunching and writing without specifying the file name. At this stage, the MASTER and APPEND commands are equivalent. Leaving FDM and returning to EXECUTIVE by using the QUIT command closes the master FDM file.

```
:CRUNCH CATLG⮐
COMPRESSED TO 76%

:APPEND⮐
65 WORDS

 :
```

The user appends a second crunched file to the already open master file LITTLE.

CAUTION: After the initial use of the MASTER command with the master file name, subsequent MASTER commands should be given without the file name. Use of the file name then causes the master file to be overwritten instead of appended.

Crunching All Files or a Range of Files

A new master FDM file can also be created by using the CRUNCH ALL command. This command crunches all symbolic files in a directory or part of a directory. FDM prompts for the starting position and the name of the master file.

```
-FDM⮐


:CRUNCH ALL⮐
START POS. 1⮐
TO:TINY⮐
 NEW FILE⮐
FILE: TEXT
COMPRESSED TO 79%
547 WORDS
FILE: SHRINK **WRONG FILE TYPE**
FILE: HALP2
COMPRESSED TO 75%
```

The user indicates that he wants all files crunched, beginning with the first.

The user writes the crunched master file onto TINY.

The CRUNCH ALL command gives the name and size of each file that can be crunched. It rejects binary, DUMP, and GO files.

```
434 WORDS
FILE: HALP1
COMPRESSED TO 77%
213 WORDS
FILE: HALP4
COMPRESSED TO 78%
73 WORDS
FILE: SMALTEXT **WRONG FILE TYPE**
FILE: LESSER
COMPRESSED TO 81%
1111 WORDS
FILE: ALPH4
COMPRESSED TO 81%
2219 WORDS

FILE BUSY
FILE: TINY **CANNOT OPEN FILE**
```

FDM cannot crunch the file on which it is writing.

```
FILES CRUNCHED: 6
FILES REJECTED: 3

:
```

To crunch only the last files in the user's directory, the user types the position of the first file to be crunched in response to the prompt. For example,

```
:CRUNCH ALL↵
START POS. 35↵
```

crunches files 35 through the end of the directory.

The user can also crunch a range of files not at the end of the directory by following the starting file number with a line feed The system then prompts for the number of the last file to be crunched.

```
:CRUNCH ALL↵
START POS. 22↴
```

A line feed here gets the system to prompt for an ending file number.

```
FINISH POS. 23↵
FILE: @BUDGET
COMPRESSED TO 79%
541 WORDS
FILE: @BAL
COMPRESSED TO 74%
80 WORDS

FILES CRUNCHED: 2
FILES REJECTED: 0

:
```

If a master file is not open, the system will prompt for the name of a master file.

Crunching Multiple Files

If the user wishes to crunch several files which are not in sequence in his directory, he can do this easily by writing the names on a file and crunching them all by the command

:CRUNCH ↑file name↵

```
-COPY T TO DATAS↵
 OLD FILE↵

DATA1↵
DATA2↵
DATA3↵
DATA4↵
Dc
```

In the EXECUTIVE, the user creates a file DATAS that contains the names of the files he wishes to crunch.

```
-FDM↵
```

He calls FDM.

```
:CRUNCH ^DATAS↵
TO:CDATAS↵
 NEW FILE↵
```

He crunches and writes the files listed in DATAS. The system prompts for the name of a master crunched file.

```
FILE: DATA1
COMPRESSED TO 85%
7 WORDS
FILE: DATA2
COMPRESSED TO 85%
7 WORDS
FILE: DATA3
COMPRESSED TO 77%
64 WORDS
FILE: DATA4
COMPRESSED TO 74%
80 WORDS


:
```

The system lists each of the files as they are crunched and written to the master file.

If a master file is already open for writing, the

:CRUNCH ↑file name↵

command will automatically crunch and append the files to the open master file without the prompt TO:.

Crunching Files by Type

The user can have all related files written on one master file, even if some of them are binary or GO files, by using the ACCEPT command. The correct form is

```
:ACCEPT file type list↵
```

where *file type list* may contain any or all of the following, separated by commas:

B	binary files
S	symbolic files
G	GO files

The two following CRUNCH procedures are identical except that the second is done with the ACCEPT command.

```
-FDM↵


:CRUNCH ALL↵
START POS. 75↴
```

The user enters a line feed so that he will be able to specify an ending file number.

```
FINISH POS. 78↵
TO:LEAST↵
 NEW FILE↵
FILE: MFIL
COMPRESSED TO 61%
118 WORDS
FILE: LITTLE **WRONG FILE TYPE**
FILE: SMALTEXT **WRONG FILE TYPE**
```

Two files are rejected because they are already binary files.

```
FILE: LYNFILE
COMPRESSED TO 72%
79 WORDS

FILES CRUNCHED: 2
FILES REJECTED: 2

:Q↵

-FDM↵


:ACCEPT B,S,G↵
OK
```

The user instructs the program to include binary, symbolic, and GO files on the master crunched file, and the system acknowledges the message with OK.

```
:CRUNCH ALL↵
START POS. 75↴
FINISH POS. 78↵
TO:LEAST↵
 OLD FILE↵
FILE: MFIL
COMPRESSED TO 61%
118 WORDS
FILE: LITTLE
```

```
36 WORDS READ
37 WORDS
FILE: SMALTEXT
33 WORDS READ
34 WORDS
FILE: LYNFILE
COMPRESSED TO 72%
79 WORDS

FILES CRUNCHED: 4
FILES REJECTED: 0

:
```

This time all files are accepted for inclusion on the master file. Note, however, that the binary files are not further compressed.

It is also possible to select file types by refusing certain types. In the following example, the user wants to group some GO and binary files on one master, but to exclude symbolic files:

```
-DIR,75↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
75 R/W NO  SYM   3-16  4  1536  MFIL
76 R/W NO  BIN   3-16  3  1536  LITTLE
77 R/W NO  BIN   3-16  3  1536  SMALTEXT
78 R/W NO  SYM   3-16  3  1536  LYNFILE
79 R/W NO  GO    3-16  1  1536  RISKS

-FDM↵


:ACCEPT↵
FILE TYPES: B,G↵
OK

:REFUSE↵
FILE TYPES: S↵
OK

:CRUNCH↵
FROM:ALL↵
START POS. 75↴
FINISH POS. 79↵
TO:BINGO↵
 NEW FILE↵
FILE: MFIL **WRONG FILE TYPE**
FILE: LITTLE
36 WORDS READ
37 WORDS
FILE: SMALTEXT
33 WORDS READ
34 WORDS
FILE: LYNFILE **WRONG FILE TYPE**
FILE: RISKS
23 WORDS READ
24 WORDS
```

The user instructs the system to accept binary and GO files, but to reject symbolic files—the reverse of the normal procedure.

```
FILES CRUNCHED: 3
FILES REJECTED: 2

:
```

Appending to Existing Master Files

At subsequent FDM sessions, the user can add more files to an existing FDM master file by using the APPEND command.

```
-FDM↵


:CRUNCH @SANDEN↵
COMPRESSED TO 73%

:APPEND BITTY↵
 OLD FILE↵
82 WORDS
```

An existing FDM master file. In this case, a carriage return in response to OLD FILE does not *delete the old file, but appends to it.*

```
:TERMINATE OUTPUT↵
OK
```

The master file is closed, then opened for reading.

```
:READ BITTY↵
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:FILES↵
```

The FILES command (see below) is used to verify that the file @SANDEN has been added to the end of BITTY.

```
  1 SYM CATLG
  2 SYM @LATER
  3 SYM @SANDEN

:
```

Uncrunching Files

The FILES Command

The FILES command lists all the files on the master crunched file currently open for reading. For example:

```
:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST

:FILES↵

  1 SYM ANAL2
  2 SYM SUN
  3 SYM CON2
  4 SYM ADDITIONS
  5 SYM UPDATE
  6 SYM LOOP
  7 SYM LOOP
  8 SYM @BUDGET
  9 SYM @BAL

:
```

The user may specify that the list start with the *n*th file on the master file by entering the command:

```
:FILES n↵
```

For example:

```
:FILES 7↵

  7 SYM LOOP
  8 SYM @BUDGET
  9 SYM @BAL

:
```

The DIRECTORY Command

The DIRECTORY command lists all the files on the master file currently open for reading, as well as their size and first line of text. For example:

```
:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST

:DIRECTORY↵
  # TYP   SIZE NAME               IDENT
  1 SYM      4 ANAL2              THIS IS A TEST
  2 SYM      3 SUN                9876$ST
  3 SYM      2 CON2               COM T76$ST
  4 SYM     13 ADDITIONS          COPY CATALOG TO EXTRA
  5 SYM     75 UPDATE             100 S=0
  6 SYM     50 LOOP               :ESC
  7 SYM      50 LOOP                :ESC
  8 SYM     540 @BUDGET             100 STRING Y(24),S4
  9 SYM      79 @BAL


:
```

If the user specifies that the directory listing begin with the *n*th file, the same information is listed but without column headings:

```
:DIRECTORY 7↵

  7 SYM      50 LOOP                :ESC
  8 SYM     540 @BUDGET             100 STRING Y(24),S4
  9 SYM      79 @BAL


:
```

The LIST Command

The LIST command is a useful way to check the whole directory without leaving FDM command level. It has the same function in FDM that it has in the EXECUTIVE. For example:

```
-FDM↵


:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST
```

The user opens the master file SHRINK and prints a list of its contents with the FILES command.

```
:FILES↵

  1 SYM ANAL2
  2 SYM SUN
  3 SYM CON2
  4 SYM ADDITIONS
  5 SYM UPDATE
  6 SYM LOOP
  7 SYM LOOP
  8 SYM @BUDGET
  9 SYM @BAL

:LIST↵
```

He then gives the LIST command, which prints his whole directory, beginning with the most recent files created.

```
BITTY END BDATAS DATAS DATA4 DATA3 DATA2 DATA1 LITTLE BIGGEST BIGGER
SHRINK SMALTRIAL 17 CON2 TRIAL CATLG @DUMMY @LATER @SANDEN @CATLG @EOM

:
```

He is returned to FDM command level.

If the LIST command is given at FDM command level while an FDM file is open for reading, the open file is not affected. It remains open and the pointer remains at the same place.

Using EXECUTIVE Commands from FDM Command Level

Any EXECUTIVE command that does not run another program or change memory (see list on page 18) can be executed from FDM command level, provided that the command does not require arguments. The first three letters of the command are used, followed by an exclamation point. In the following example, the user opens a master file that contains three crunched files. The first use of the DIRECTORY command lists the directory of the currently open master FDM file. The user then wishes to check his overall directory and does so by typing the command

:DIR!⮐

He then aborts the EXECUTIVE DIRECTORY command and returns to his still open master file. Use of an EXECUTIVE command by this procedure does not affect any file open in FDM.

```
:READ BITTY⮐
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:DIR⮐
  # TYP   SIZE NAME               IDENT
  1 SYM     64 CATLG              ACME LABS,100,AMERICAL INSTRUMENT,100
  2 SYM     13 @LATER             :PERPOUT
  3 SYM     81 @SANDEN            /\/\/\\/III

:DIR!⮐
 # PVT  PUB TYP  DATE USE  SIZE  NAME
 1  R   NO  SYM   3-11 55  1536  ADDITIONS
 2  R   NO  SYM   3-11 42  1536  UPDATE
 3  R   NO  SYM   3-9  45  1536  LOOP
 4 R/W YES ,WT SYM   1-12 21  1536  SBASAVINTRST
 5  R   NO  SYM   2-10⊕

:GET 2⮐
FILE: @LATER
:PERPOUT

:
```

Finding Files within the FDM Master File

The user has three convenient commands to help him find files within a large FDM master file: FIND, SEARCH, and GET. The SEARCH command finds the file sought, then prints its position number, type, and name. The FIND command finds the file, then prints the file name and first line. For example:

```
:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST

:SEARCH UPDATE↵

  5 SYM UPDATE

:FIND SUN↵
FILE: SUN
98765

:
```

If the user knows the position number, he can use the GET command to locate the file:

```
:GET 2↵
FILE: SUN
98765

:
```

The RESTORE Command

With the RESTORE command, the user can select a file for uncrunching by using its position number in the master file.

The master file must be opened for reading first. For example:

```
-FDM↵


:READ SHRINK↵          The user opens the master file SHRINK.
FILE: ANAL2
THIS IS A TEST

:SEARCH UPDATE↵        He finds the file number of his file UPDATE.

  5 SYM UPDATE

:RESTORE 5↵            He uncrunches UPDATE and writes it as a symbolic file under its old name.
FILE: UPDATE
100 S=0
294 CHARACTERS

:Q↵    He quits FDM, then uses the DIRECTORY command to look at the end of his directory, verifying that
       UPDATE has been restored.
```

```
-DIR,44↩
 # PVT PUB TYP  DATE USE  SIZE  NAME
44 R/W NO  BIN   3-11  6  1536  BITTY
45 R/W NO  SYM   3-11  1  1536  UPDATE

-
```

The UNCRUNCH ALL Command

An entire master file can be uncrunched using the UNCRUNCH ALL command. This command supplies the name, first line (for symbolic files), and size of each file on the master file. The master file BITTY is uncrunched below:

```
-FDM↩


:READ BITTY↩
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:UNCRUNCH ALL↩
OK? Y↩
FILE: CATLG
249 CHARACTERS
FILE: @LATER
:PERPOUT
50 CHARACTERS
FILE: @SANDEN
(LAST FILE)
/\/\/\\/III
330 CHARACTERS

FILES UNCRUNCHED: 3
NO. UNCRUNCHABLE: 0

:
```

The user can uncrunch only the last part of a master file by using the GET command. The GET command locates the first file desired, then that file and the remaining files are uncrunched. For example:

```
:READ SHRINK↩
FILE: ANAL2
THIS IS A TEST

:GET 8↩
FILE: @BUDGET                      The eighth file on SHRINK is located.
100 STRING Y(24),S4
```

```
:UNCRUNCH ALL↵
OK? Y↵
FILE: @BUDGET
2025 CHARACTERS
FILE: @BAL                   The eighth and ninth files are uncrunched.
(LAST FILE)


316 CHARACTERS

FILES UNCRUNCHED: 2
NO. UNCRUNCHABLE: 0

:
```

The user can also uncrunch a range of files ending at any position in the master file. The procedure is the same as the above, except that a line feed is used after the UNCRUNCH ALL command. The system then asks for the number of the last file the user wants uncrunched.

```
:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST

:GET 3↵
FILE: CON2
COM T

:UNCRUNCH ALL↴           The user types a line feed, then supplies the number of the last file to be uncrunched.
FINISH POS. 4↵
OK? Y↵
FILE: CON2
6 CHARACTERS                     The third and fourth files are uncrunched.
FILE: ADDITIONS
COPY CATALOG TO EXTRA
55 CHARACTERS

FILES UNCRUNCHED: 2
NO. UNCRUNCHABLE: 0

:
```

FDM Command Files

FDM commands can be included in ordinary command files, which are run from the EXECUTIVE (see page 45), or in FDM command files, which are run from FDM command level. For example, the EXECUTIVE command file SHRUNK is as follows:

```
FDM
READ SHRINK
UNCRUNCH ANAL2

QUIT
COMMAND T
```

The space reflects a carriage return, which must be entered in the command file to respond to the NEW FILE/OLD FILE message.

A comparable FDM command file is identical except that FDM is not called. To execute an FDM command file, the DO command is given at FDM command level. For example, the FDM command file SHRUNK has the commands

```
READ SHRINK
UNCRUNCH ANAL2

QUIT
COMMAND T
```

and is executed as follows:

```
-FDM↵

:DO SHRUNK↵

:READ SHRINK
FILE: ANAL2
THIS IS A TEST

:UNCRUNCH ANAL2
 OLD FILE15 CHARACTERS

:QUIT
```

The commands are printed on the terminal as they are executed.

The user can suppress terminal output by having all conversation written on a terminal output (TOUT) file.[1] For example, the above command file has been altered to include an output file as follows:

```
TOUT NOISE
```

Terminal output will now go to the file NOISE.

```
READ SHRINK
UNCRUNCH ANAL2

QUIT
TOUT T
COMMAND T
```

The EXECUTIVE command TOUT T closes the TOUT file and directs output back to the terminal.

[1] – See page 47 for a discussion of EXECUTIVE TOUT files.

When this command file is run, all terminal output is suppressed except for the TOUT command and the confirming message NEW FILE which applies to the TOUT file itself.

```
:DO SHRUNK↵


:TOUT NOISE
 NEW FILE
-
```

The TOUT file NOISE now contains the following output:

```
-TYPE NOISE↵



:
FILE: ANAL2
THIS IS A TEST

:
 OLD FILE15 CHARACTERS

:

-
```

If the user chooses to record terminal output on a TOUT file but would still like information on the progress of the command file execution, he can include in the command file text which will override the TOUT command and print on the terminal. To do this, he uses the OUTPUT command. All terminal output will still go to the TOUT file except that specified in the OUTPUT command. For example, the user modifies his command file by adding two lines that he wishes printed out at the terminal:

```
TOUT NOISE

READ SHRINK
UNCRUNCH ANAL2

OUTPUT ..............THE FILE IS NOW UNCRUNCHED.
OUTPUT YOU ARE BEING RETURNED TO EXEC COMMAND LEVEL.....
QUIT
TOUT T
COMMAND T

-
```

The file is now executed as follows:

```
:DO SHRUNK↵


:TOUT NOISE
 OLD FILE..............THE FILE IS NOW UNCRUNCHED.
YOU ARE BEING RETURNED TO EXEC COMMAND LEVEL.....


-
```

Controlling Terminal Output

The EXPERT command reduces FDM conversation to a minimum for experienced users. To return to the conversation mode, the NOVICE command is used. For example:

```
:EXPERT↵                          The EXPERT mode
OK

:CRUNCH UPDATE↵
OK

:MASTER BUPDATE↵
 NEW FILE↵
76 WORDS

:TERMINATE OUTPUT↵
OK

:NOVICE↵                          The NOVICE mode
OK

:CRUNCH UPDATE↵
COMPRESSED TO 76%

:MASTER BUPDATE↵
 OLD FILE↵
76 WORDS

:TERMINATE OUTPUT↵
OK

 :
```

The TOUT command can be used to suppress *all* terminal output when entering commands from the terminal; however, the user must remember to answer all unprinted NEW FILE/OLD FILE messages with carriage returns. If NOTHING is specified as the TOUT file, the conversation is not saved. Note that even FDM prompts are printed on the TOUT file instead of at the terminal.

```
-FDM↵


:READ SHRINK↵                    The user reads and uncrunches a crunched file in the normal manner.
FILE: ANAL2
THIS IS A TEST

:UNCRUNCH ANAL2↵
 OLD FILE↵
15 CHARACTERS

:Q↵


-FDM↵


:TOUT↵
TO:SAVE↵        The user reads and uncrunches a crunched file, writing all conversation on the file SAVE.
 NEW FILE↵
READ SHRINK↵
UNCRUNCH ANAL2↵
↵
Q↵          The user types a carriage return to answer the unprinted question NEW FILE.

-TOUT T↵        The user closes the TOUT file by returning output to the terminal.

-
```

The HELP Command

The HELP command lists all FDM commands in alphabetical order. For example:

```
:HELP↵
COMMANDS:

APPEND
ACCEPT
CRUNCH
COMMENT
CLEAR
CONVERSATION
CREDITS
CHARGES
DIRECTORY
DO
E⊕

:
```

Multiple Blanks in Files

In the normal mode of operation, when files with multiple blanks are uncrunched, all the blanks are written. If the user wishes to write his uncrunched symbolic file with multiple blanks compressed, he can activate the MB ON option before uncrunching his file. For example, in the normal mode of operation a structured file is uncrunched as follows:

```
:READ SMALTEXT↵
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:GET 5↵
FILE: COLOR                 COLOR is a RETRIEVE file with multiple blanks.
(LAST FILE)
XEXEC DATA SHEET                    40GRAY

:UNCRUNCH SCOLOR↵
 NEW FILE↵
5000 CHARACTERS             With all blanks written out, it is 5000 characters.
```

With the MB ON option in use, the same file is uncrunched as follows:

```
:READ SMALTEXT↵
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:GET 5↵
FILE: COLOR
(LAST FILE)
XEXEC DATA SHEET                    40GRAY

:MB ON↵                     The user activates the multiblank compression option.
OK

:UNCRUNCH BCOLOR↵
 NEW FILE↵
3066 CHARACTERS             With blanks compressed, the file is 3066 characters long.

:
```

NOTE: Command files will not work properly if they are written with multiple blanks compressed, and RETRIEVE files cannot be used in RETRIEVE with multiple blanks compressed.

The user can return to normal mode (multiple blanks written out) by typing

```
:MB OFF↵
```

before uncrunching a file.

The PRINT Command

The PRINT command prints the first line of the current crunched file in symbolic form. For example:

```
:CRUNCH @EOM↵
COMPRESSED TO 75%

:PRINT↵
:BAL

:
```

The RENAME Command

The RENAME command is used to rename files as they are being crunched and stored on a master file. For example:

```
:CRUNCH COLOR↵
COMPRESSED TO 43%

:RENAME DULL↵              COLOR is renamed as DULL.
OK

:APPEND SMALTEXT↵          It is appended to master file SMALTEXT.
 OLD FILE↵
732 WORDS

:TERMINATE OUTPUT↵
OK

:READ SMALTEXT↵            SMALTEXT is opened for reading.
FILE: CATLG
ACME LABS,100,AMERICAL INSTRUMENT,100

:FILES 6↵                  The last file on the master file SMALTEXT is DULL.

  6 SYM DULL

:
```

The RESET Command

The RESET command is used to clear all text from the FDM buffer, close all files, and reset all options (for example, ACCEPT file type, MB ON, etc.) to normal status. To illustrate:

```
:UNCRUNCH BITTY↵
 OLD FILE↵
249 CHARACTERS

:PRINT↵          The user prints the first line of the first file in BITTY.
ACME LABS,100,AMERICAL INSTRUMENT,100

:RESET↵          He clears the buffer, resets all options, and closes all files.
OK

:PRINT↵          This time he gets an error message in response to the PRINT command.
BUFFER EMPTY?

:
```

The STATISTICS Command

The STATISTICS command prints the following data for a specified file: name, position on the master file, file size, and the remaining core buffer space. For example:

```
:READ SHRINK↵
FILE: ANAL2
THIS IS A TEST

:STATISTICS↵
SYM ANAL2  #1-MASTER
SIZE: 4/49152 WORDS
49148 (46K) WORDS FREE

:GET 2↵
FILE: SUN
98765

:STATISTICS↵
SYM SUN  #2-MASTER
SIZE: 3/49152 WORDS
49149 (46K) WORDS FREE

:
```

The TERMINATE Command

To close the input and/or output file, the TERMINATE command is used. The command

:TERMINATE INPUT↵

closes the input (or read) file. The command

:TERMINATE OUTPUT↵

closes the output file opened by the MASTER or APPEND command. The command

```
:TERMINATE ALL↵
```

closes all input/output files.

NOTE: The QUIT command closes all files before returning to the EXECUTIVE.

The VERSION Command

The VERSION command types the current FDM version number. For example:

```
:VERSION↵
B03.00

:
```

THE CHECKSUM AND VERIF PROGRAMS

A checksum is a unique number associated with a particular file content. The checksum is calculated by adding the binary equivalent of all the words on the file. If a file is copied, renamed, or transferred from one user to another, the checksum should remain the same. If the contents of a file are changed in any way, the checksum changes. Therefore, checksums are useful for keeping track of various versions of files and for ensuring that a file has not been altered in any way.

NOTE: The checksum is not related to the size of the file.

File checksums can be obtained with the CHECKSUM or VERIF command.[1] With the CHECKSUM command, the checksum for an individual file can be listed. The VERIF command gives the checksums for a range of files.

The CHECKSUM command has the following form:

```
-CHECKSUM↵              The user calls CHECKSUM.

INPUT: SHRINK↵          He specifies a file name.
4062341                 CHECKSUM prints the checksum.
INPUT: .↵               A period returns the user to the EXECUTIVE.

-
```

The VERIF command is used to obtain checksums for a range of files. The user calls VERIF, then specifies the number of the first file in the range followed by a line feed, then enters the number of the last file in the range. If the beginning number is followed by a carriage return, the ending number is assumed to be the last file number. VERIF prints the date and time, then the number, type, name, and checksum for each file.

[1] – The DIRIT program, described on page 87, can also be used to obtain checksums of selected files.

```
-VERIF↵


BEG.NO.: 23↴
END.NO.: 25↵


03/11/76  17:36

23  SYM  @EOM-50162247
24  SYM  @CATLG-77307717
25  SYM  @SANDEN-27455511




-
```

The user may create a command file to perform the functions of the CHECKSUM and VERIF programs.[1] When creating a CHECKSUM command file the user specifies the files he wishes to check by placing a period (.) before the first file name and after the last file name. For example, the following files are in a user's directory:

```
-FIL↵

SYM   ADDITIONS
SYM   LOOP
SYM   BUDGET
BIN   SFOPROG
GO    ACCTS
GO    SORT ⊕
-
```

The user wishes to perform a CHECKSUM on files ADDITIONS, BUDGET, and LOOP. He creates a command file in EDITOR.[2]

```
-EDI↵
*APPEND↵
CHECKSUM↵
.ADDITIONS↵
BUDGET↵
LOOP↵
.↵
COMMAND T↵
D^c
*WRITE CKFIL↵
 NEW FILE↵
44 CHARS
*Q↵
```

The user places a period before the first file he wants to check and after the last.

[1] – Refer to the discussion of command files on page 45 for information on creating command files.

[2] – See the *Tymshare EDITOR Reference Manual.*

```
-COMMAND CKFIL↩
```

He executes the command file CKFIL.

```
ADDITIONS-33504
BUDGET-33236166
LOOP-23401637

-
```

The program responds with the checksums of the requested files.

When creating a VERIF command file the period is unnecessary. The user creates the file as follows:

```
-EDITOR↩
*APPEND↩
VERIF↩
20↓
24↩
```

The user specifies the beginning number of the range of files followed by a line feed and the ending number followed by a carriage return.

```
COMMAND T↩
Dc
*WRITE CFIL↩
 OLD FILE↩
22 CHARS
*QUIT↩

-COMMAND CFIL↩


03/16/76  15:22

20  SYM  @BAL-15327604
21  SYM  @CATLG-77307717
22  SYM  @SANDEN-27455511
23  SYM  @LATER-47726701
24  SYM  @DUMMY-53121027
```

COMPARING TWO FILES

Tymshare provides two different programs for comparing the contents of files. SCOMPARE, which compares two files line by line, is an efficient method of comparing symbolic files, either programs or data files. COMPARE, which compares two files word by word, is especially useful, for example, in finding changes made in a GO file or a NARP object file.

The SCOMPARE Program

The SCOMPARE program is called by typing SCOMPARE and a carriage return in the EXECUTIVE. The program requests the names of the two files to be compared, then the name of the output file. The user may enter the letter T and a carriage return to have the results printed on the terminal. If he enters a file name, the OLD FILE/NEW FILE message is printed and must be confirmed with a carriage return or aborted with an alt mode/escape. The following example illustrates the SCOMPARE program.

```
-TYPE DATA1↵

ABC
DEF
GHI
JKL
MNO

-TYPE DATA2↵

ABC
GHI
MNO
STU
VWY

-SCOMPARE↵
INPUT FILE 1: DATA1↵      The contents of DATA1 are compared with the contents of DATA2.
INPUT FILE 2: DATA2↵
OUTPUT TO: DATACOMP↵      The output is written on a new file DATACOMP.
 NEW FILE↵
```

SCOMPARE next requests the number of lines to match in the files. This number specifies how many lines apart two unmatched lines can be and still be considered a single discrepancy. The number may range from one to the total number of lines in the file; if a carriage return is entered in response to the prompt, the number used is three.

```
NUMBER OF LINES TO MATCH = 1↵
```

The contents of DATACOMP are shown below.

```
-TYPE DATACOMP↵

03/11/76  17:50
1)=DATA1
2)=DATA2

1) DEF
1) GHI
1) JKL
1) MNO
****2
2) GHI
2) MNO
```

```
2) STU
2) VWY
********2

-
```

For each file, the lines in which differences occur are printed, followed by a row of asterisks and the line number of the first line in each group. The following examples demonstrate the SCOMPARE program. The four sample files are displayed:

```
-TYPE ALPH1↵

AAA
BBB
CCC
DDD
EEE
FFF
GGG
HHH

-TYPE ALPH2↵

AAA
BBB
CCC
EEE
FFF
GGG
HHH

-TYPE ALPH3↵

AAA
BBB
CCC
DDD
EEE
FFFF
GGG
HHH

-TYPE ALPH4↵

AAA
BBB
CCC
EEE
FFF
HHH

-
```

SCOMPARE works as follows:

```
-SCOMPARE↩
INPUT FILE 1: ALPH1↩        ALPH1 and ALPH2 are the files being compared.
INPUT FILE 2: ALPH2↩
OUTPUT TO: T↩
NUMBER OF LINES TO MATCH = 1↩

03/11/76  17:56
1)=ALPH1
2)=ALPH2

1) DDD          1) indicates that these are lines from file 1 (ALPH1).
1) EEE
****4           Line number of first discrepant line in file 1.
2) EEE
********4       Line number of first discrepant line in file 2.


-
```

The difference between the two files is that DDD is omitted in file ALPH2. Therefore, line 4 in file ALPH2 is EEE. The short row of asterisks signifies the first discrepant line, 4, in file 1, ALPH1. The longer row of asterisks signifies the first discrepant line in file 2, ALPH2.

Note that the program also prints the first matching pair of lines that occur after a discrepancy.

```
-SCOMPARE↩
INPUT FILE 1: ALPH1↩        ALPH1 and ALPH3 are files being compared.
INPUT FILE 2: ALPH3↩
OUTPUT TO: T↩
NUMBER OF LINES TO MATCH = 1↩

03/11/76  17:58
1)=ALPH1
2)=ALPH3

1) FFF          Line 6 in ALPH1 is FFF; line 6 in file ALPH3 is FFFF. Lines 7 in the two files,
1) GGG          GGG, match.
****6
2) FFFF
2) GGG
********6


-
```

```
-SCOMPARE↵
INPUT FILE 1: ALPH2↵
INPUT FILE 2: ALPH4↵
OUTPUT TO: COMP↵
 NEW FILE↵
NUMBER OF LINES TO MATCH = 1↵
```

ALPH2 and ALPH 4 are the files being compared. The output is written on a new file named COMP.

```
-TYPE COMP↵
```

The file is displayed.

```
03/11/76  17:58
1)=ALPH2
2)=ALPH4

1) GGG
1) HHH
****6
2) HHH
********6

-
```

The significance of the number of lines matched is illustrated below. The files ALPH1 and ALPH4 are compared twice. The first comparison uses 1 as the number of lines to match and discovers two discrepancies. The second method uses 4 as the number of lines to match and treats the mismatches as a single discrepancy. In this case the four lines containing the mismatches are considered as one unit.

```
-SCOMPARE↵
INPUT FILE 1: ALPH1↵
INPUT FILE 2: ALPH4↵
OUTPUT TO: T↵
NUMBER OF LINES TO MATCH = 1↵
```

The number of lines to match is 1; therefore the discrepancies are treated separately.

```
03/11/76  18:00
1)=ALPH1
2)=ALPH4

1) DDD
1) EEE
****4
2) EEE
********4
1) GGG
1) HHH
****7
2) HHH
********6

-SCOMPARE↵
INPUT FILE 1: ALPH1↵
INPUT FILE 2: ALPH4↵
```

```
OUTPUT TO: T↵
NUMBER OF LINES TO MATCH = 4↵

03/11/76  18:00
1)=ALPH1
2)=ALPH4

1) DDD
1) EEE
1) FFF
1) GGG
1) HHH
****4
2) EEE
2) FFF
2) HHH
********4


-
```

The number of lines to match is 4. The differences in the files are less than four lines apart and are therefore treated as a single discrepancy.

If the question "number of lines to match" is answered with a number followed by a line feed instead of a carriage return, SCOMPARE will ask

SUPPRESS BLANKS?

If this question is answered with Y, blank differences will be ignored when the lines are compared. Any other single-character answer will force the default mode of comparing blanks.

The COMPARE Program

The COMPARE program compares two files word for word and lists the content and location of any words which are not exactly the same. For each discrepancy encountered, COMPARE prints the letter D, the number of the discrepant word, and the contents of the word in each file. When COMPARE encounters the ends of the two files, it prints E and the number of words compared. If it encounters the end of one file before the other, it prints the file identifier (A or B) of the shorter file and the number of words in that file. For example,

```
-COMPARE↵
AALPH1↵
+0↵

BALPH4↵
+0↵

D 4 11022044 11222445
D 5 33222445 33223046
D 6 11266446 11466450
D 7 11423155 12024155
D 10 11623447 27657537
B 10
-
```

The user gives the first file name, ALPH1, and starting location 0; then the second file name, ALPH4, and starting location 0.

D 4 means there is a discrepancy in the fourth word processed.

B 10 means that the end of the second file was encountered after 10 lines.

Section 10
THE DIRIT PROGRAM

DIRIT is a utility program for file management that can list files, delete files, or copy files to another user's directory. The program is particularly valuable to users with large directories because it provides a wide range of options for sorting and selecting files and for selecting information about files.

The examples on the following pages show several ways to combine the sort, select, and information options. These examples only suggest the wide variety of possible combinations. By examining the lists of options in this section, the user will find the combinations of forms most suited to his own unique needs.

The DIRIT program can operate in one of two modes: the once-only mode or the command dispatcher mode. In the once-only mode, DIRIT executes any valid command and returns to the EXECUTIVE upon completion of the task. In the command dispatcher mode, DIRIT prompts with a colon, accepts a command, executes it, and prompts again for another command. The command dispatcher mode is terminated by typing a carriage return immediately after the colon prompt.

To initialize DIRIT in the once-only mode, the user types DIRIT followed by the DIRIT command form on the same line. The form is

```
        ⎧file identifier(s)!command list⎫
−DIRIT  ⎨file identifier(s)             ⎬ ↵
        ⎩!command list                  ⎭
```

where the file identifier(s) are file names or any of the forms listed on pages 91–97 and the command list consists of any of the commands discussed in this section. The file identifiers can be separated by spaces, commas, or semicolons. Each command is preceded by an exclamation point but no spaces. A space is optional to separate file identifiers from command forms. For example, at the EXECUTIVE command level,

```
−DIRIT DA###!CHECKSUMS↵
```

obtains the checksums of all files whose names contain five characters, the first two of which are DA.

```
CHECKSUM FILE NAME
22513123 DATA1
61032665 DATA2
25145004 DATA3
15327604 DATA4
52204373 DATAS

-
```

Control is returned to the EXECUTIVE after the command is executed.

To obtain the same information and stay at DIRIT command level, the following form is used:

```
-DIRIT↵
:DA###!CHECKSUMS↵
```

The carriage return causes the DIRIT prompt.
The user asks for checksums on the same files.

```
CHECKSUM FILE NAME
22513123 DATA1
61032665 DATA2
25145004 DATA3
15327604 DATA4
52204373 DATAS

:
```

This time control is returned to DIRIT command level and the user can now enter another DIRIT command.

Controls A, W, and Q can be used when entering DIRIT commands.

Each DIRIT command can be shortened to the least number of characters that make it unique.

LISTING FILES AND RELATED INFORMATION

When listing information about files, the user can sort the list alphabetically, chronologically, numerically, by type, or by number. All these sorts can also be done in reverse order.

The user can select files about which he wants information by date, by directory number or range of numbers, by name, or by any characters within the name.

He can list any or all of the information about files that is provided by the DIRECTORY command (see page 39), plus checksum (see page 79) and status information (see page 98).

Sorting Files

When listing file information, any of the following commands can be used to sort the files:

Command	Sort
!ALPHABETICAL	Alphabetical.
!CHRONOLOGICAL	By date of creation or last change.
!ACS	By the number of times the file has been accessed.
!PSS	By file number, that is, position in the directory.

Command	Sort
!RCHRONOLOGICAL	Reverse chronological order, that is, the order used when the DIRECTORY command is given.
!REVERSE	When combined with another sort command, reverses that order. If no other sort command is given, the order is reverse file number.
!TPS	By file type: SYM, BIN, GO, DUM, BAD.

Any of the sort commands can be given without a file identifier. In that case, the program prints all file names in the order specified by the command. For example:

```
-DIRIT↵                  The user calls the command dispatcher mode of DIRIT.
:!ALPHABETICAL↵          He asks for an alphabetical printout of file names in his directory.

FILE NAME
/$/
//TEMP
17
@BAL
ABIN
ACCTS
ADDITIONS
ALPH1
ALPH2
AL⊕                      When he strikes the alt/mode escape, DIRIT replies with (ESC) and the
(ESC)                    DIRIT prompt.

:
```

To reverse any order, the user gives the REVERSE command in combination with the desired sort command. For example:

```
:!REVERSE!ALPHABETICAL↵       Each command is preceded by an exclamation point, but commands
                              are not separated by spaces.
FILE NAME
UPDATE
TRIAL
TINY
TEXT
SUN
SORT
SMALTRIAL
SMALTEXT
SKUNK⊕
(ESC)

:
```

The sort commands can be combined with any of the file identifiers discussed on pages 91–97. For example, the file identifier :BEAR will find all file names that end in BEAR. If the user combines this identifier with the command !ALPHABETICAL, all such files will be listed in alphabetical order.

```
::BEAR!ALPHA↵

BABYBEAR
MAMABEAR
PAPABEAR

:
```

Selecting Files

The user has available three simple command forms for selecting files by date. The other forms for selecting files—by file number, by file type, by name, or by partial name—are all specified as file identifiers.

By Creation (or Change) Date

The user may select files created or changed either before or after a certain date. He may also select files created during the current day. The command forms are:

```
!BEFORE date↵

!AFTER date↵

!TODAY↵
```

These commands can be combined with any of the sort commands or with the commands that select information about files. For example, the user can look at creation dates and file names of all files created before February 12, 1976, by using the following:

```
:!DATE!BEFORE 12-FEB-76↵

CREATION  FILE NAME
10-FEB-76 BUDGET
10-FEB-76 BUDG
30-DEC-75 @CATLG
12-JAN-76 SBASAVINTRST

:
```

The form for selecting files after a certain date is similar:

```
:!DATE!AFTER 12-FEB-76↵

CREATION  FILE NAME
12-MAR-76 ADDITIONS
 9-MAR-76 LOOP
 9-MAR-76 SFOPROG
12-MAR-76 NAUT
10-MAR-76 CATALOG
12-MAR-76 /$/
10-MAR-76 HOPEFUL
10-MAR-76 SFPRG
10-MAR-76 SUN
10-MAR-76 ABIN
10-MAR-76 CBIN
11-MAR-76 @BUDGET
11-MAR-76 @BAL
11-MAR-76 @SANDEN⊕
(ESC)

:
```

If the year is not specified in the !BEFORE or !AFTER command, the current year is assumed. If the month is not specified, the current month is assumed.

The user can list the files created today and check their size by using the following form:

```
:!SIZE!TODAY↵

 CHARS FILE NAME
   116 TRIAL
   378 SHRINK
    24 DATAS

:
```

By Directory Position (File Number)

The user can specify any file number, numbers, or range of numbers by including them in parentheses. These numbers are file identifiers, not commands, and so do not take an exclamation point. They precede any commands in the command form. The forms are:

(n)	Specifies a file number.
(n_1,n_2,n_3, . . .)	Specifies several file numbers.
(n_1–n_2)	Specifies any range of positions.

If the file identifiers are typed alone, the program lists only the file names for the files specified.

```
:(38)↩

FILE NAME
BDATAS

:
```

The file identifiers can also be used in combination with DIRIT commands. For example, to list the creation or change date of the thirty-eighth file in the directory, the user types

```
:(38)!DATE↩

CREATION  FILE NAME
11-MAR-76 BDATAS

:
```

To examine the creation dates and verify position numbers of several files, the user types

```
:(13,15,38)!DATE!POSITION↩

POS CREATION  FILE NAME
 13 10-FEB-76 BUDG
 15 10-MAR-76 SFPRG
 38 11-MAR-76 BDATAS

:
```

The user can list the number of times accessed for a range of files by using the form

```
:(37-40)!POSITION!ACCESS↩

POS  USE FILE NAME
 37    7 DATAS
 38    1 BDATAS
 39   17 BITTY
 40    8 UPDATE

:
```

By File Type

To select files by type, the user enters the form

.t

or

$.t_1,t_2, \ldots ,t_n$

where *t* can be SYM, BIN, GO, DUM, or BAD.

Again, the user can enter the file type alone to obtain a listing of only the file names for that file type:

The file type can be combined with any DIRIT commands:

```
:.DUM,BIN!TYPE↵

TYP  FILE NAME
BIN  SFOPROG
BIN  NAUT
DUM  /$/
DUM  HOPEFUL
BIN  SFPRG
BIN  ABIN

:
```

By Name

The user can simply list the file names, separated by spaces, commas, or semicolons. If a file name has a comment (see page 48), that comment must be included. For example, to obtain the file numbers of several files, the user types

```
:CBIN,UPDATE,CFIL,ADDITIONS-BASE,ANAL2!POSITION↵

POS FILE NAME
 17 CBIN
 39 UPDATE
 48 CFIL
 80 ANAL2
 85 ADDITIONS-BASE

:
```

If the comment is not included in the file name for ADDITIONS, the program will not recognize the name.

By Partial File Name

Files can be selected by specifying any number of characters in the file name with the following identifier forms:

aa## — where *a* can be any unprotected character that can be included in a TYMCOM-IX file name. This file identifier finds all file names having the specified characters in the specified position and having the same total number of characters as specified.

:aaa — File name must end with the specified characters.

a: — File name must begin with the specified characters.

:aa: — File name must contain the specified characters, but they can be located anywhere in the name.

If the character string to be sought contains any characters that need to be protected in file names (see page 23 for rules for naming files), that string should be enclosed in angle brackets (< >). The brackets are not considered part of the string.

The following examples demonstrate several ways in which partial file names can be used to specify files.

```
:AL###↵

FILE NAME
ALPH1
ALPH2
ALPH3
ALPH4
```

This form finds all 5-letter file names that begin with AL.

```
:AL:↵

FILE NAME
ALPH1
ALPH2
ALPH3
ALPH4
ALPHABET
ALP
```

This form finds all file names that begin with AL regardless of length.

```
::AL:↵

FILE NAME
CATALOG
@BAL
TRIAL
SMALTRIAL
ALPH1
ALPH2
ALPH3
ALPH4
SMALTEXT
ANAL2
```

This form finds all file names that contain AL at any position and are of any length.

```
ALPHABET
ALP
HALP
```

```
::PH2↵
```

This form finds all files that end with PH2, regardless of length.

```
FILE NAME
ALPH2
COPH2
```

```
::2↵
```

This form finds all files that end with 2, regardless of length.

```
FILE NAME
REPT2
CON2
DATA2
ALPH2
ANY2
ANAL2
COPH2
```

```
::<R L>:!CHECKSUM↵
```

This form finds all file names that contain R L. The character string must be enclosed in angle brackets because a space must be protected when it occurs in a file name.

```
CHECKSUM FILE NAME
13724560 'R L'
45745376 'AR L1'
05573230 'STAR LITE'

:
```

Combining File Identifiers

File identifiers can be combined by using any of the following file identifier switches:

!AND

!ANN *(means And Not)*

!OR

The !OR switch will list files satisfying either file identifier criterion:

```
:(47-50)!OR ALPH#↵

FILE NAME
CKFIL
CFIL
DATACOMP
ALPH1
ALPH2
ALPH3
ALPH4

:
```

The !AND switch will list only those files that satisfy both file identifier criteria simultaneously.

```
:(47-50)!AND ALPH#↵

FILE NAME
ALPH1

:
```

The !ANN switch picks those files that meet the first criterion but not the second:

```
:(47-50)!ANN ALPH#↵

FILE NAME
CKFIL
CFIL
DATACOMP

:
```

Negating File Identifiers

The user can pick all files that do not meet the specified criterion by preceding it with the !NOT switch and a space. For example:

```
:!NOT :A:↵

FILE NAME
LOOP
BUDGET
SFOPROG
SORT
REPT1
REPT2
REP
/$/
BUDG
HOPEFUL
SFPRG
SUN
CBIN
@BUDGET
@DUMMY
CON2
17
SHRINK
BIGGER
BIGGEST
⊕
(ESC)

:
```

In the following example, the user wants an alphabetical list, with file type, of all files in his directory that are not symbolic.

```
:!NOT  .SYM!TYPE!ALPHA↩

TYP  FILE NAME
BIN  @PRINT
GO   ACCTS
GO   CATALOG
DUM  HOPEFUL
GO   REP
GO   REPT1
GO   REPT2
BIN  SFOPROG
BIN  SFPRG
BIN  SHRINK1
BIN  SMALTEXT
BIN  SMALTRIAL
GO   SORT

:
```

Selecting Information about Files

The user can specify any or all of 10 specific kinds of information to be listed about the files in his directory by using the information selection commands.

For example, if he wants to know the checksum, file number, and usage of all his files containing the beginning characters CO, he combines the CO: file identifier with the appropriate commands:

```
:CO:!CHECKSUM!POSITION!ACCESS↩

POS CHECKSUM  USE FILE NAME
 22 45745376   18 CON2
 36 70421725    1 COLOR'STR.E'
 45 42457136    2 COMP
 54 45270553    2 COMMI
 57 05573230    2 COM1

:
```

The following items of information can be obtained:

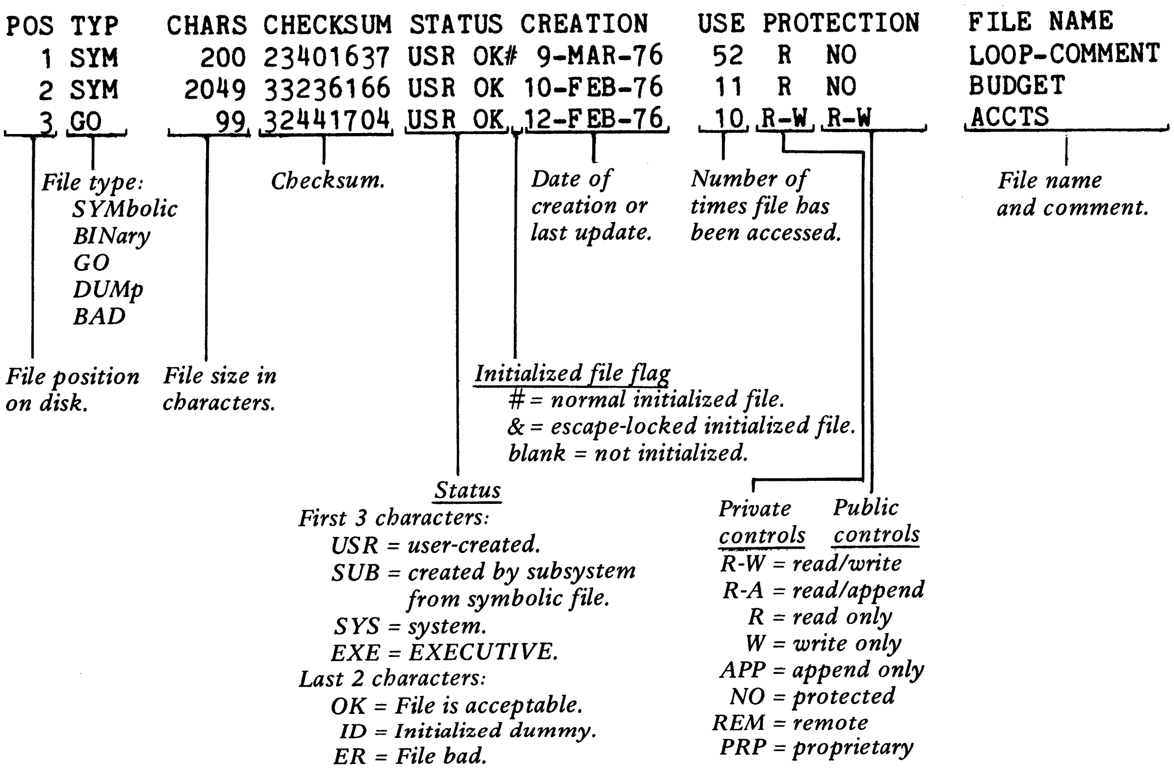
Command	Information
!ACCESS	The number of times the file has been used and the file name.
!CHECKSUM	The checksum and file name.
!CREATION	The date of creation or last update and file name.

(Continues)

Command	Information
!DIRECTORY	The position number, file type, size in characters, creation date, protection, and file name.
!EVERYTHING	All data available about the files. Concludes listing with total storage in words, characters, and files, for files listed.
!FAST	File type, creation date, and file name.
!LIST	File name only, several names per line; see !SHORT below.
!POSITION	File number and name.
!PROTECTION	File protection and name.
!SHORT	File names, one per line; see !LIST above.
!SIZE	File size, in characters, and file name.
!STATUS	Status and file name.
!SUMMARY	All file names followed by total characters, total words, total files.
!TYPE	File type and file name.

The format of the information printed in response to the above commands is illustrated below by showing the output format of the !EVERYTHING command.

```
-DIRIT
:!EVERYTHING
```



```
POS TYP   CHARS CHECKSUM STATUS CREATION   USE PROTECTION   FILE NAME
  1 SYM     200 23401637 USR OK# 9-MAR-76   52  R   NO      LOOP-COMMENT
  2 SYM    2049 33236166 USR OK 10-FEB-76   11  R   NO      BUDGET
  3 GO       99 32441704 USR OK 12-FEB-76   10 R-W R-W      ACCTS
```

File position on disk.

File type:
SYMbolic
BINary
GO
DUMp
BAD

File size in characters.

Checksum.

Date of creation or last update.

Number of times file has been accessed.

File name and comment.

Initialized file flag
# = normal initialized file.
& = escape-locked initialized file.
blank = not initialized.

Status
First 3 characters:
USR = user-created.
SUB = created by subsystem from symbolic file.
SYS = system.
EXE = EXECUTIVE.
Last 2 characters:
OK = File is acceptable.
ID = Initialized dummy.
ER = File bad.

Private controls / *Public controls*
R-W = read/write
R-A = read/append
R = read only
W = write only
APP = append only
NO = protected
REM = remote
PRP = proprietary

The !EVERYTHING Printout

DELETING FILES

Much of the flexibility available for listing files and related information is also available for deleting files. The basic command form is

```
: [file identifier] !DELETE! [command list] ! [DECIDE]↩
```

where the file identifier can be any of those listed on pages 91–97, the commands can be any of the sorting commands listed on pages 88–90 or the date commands described on pages 90–91.

The user can wipe out his whole directory with the simple DIRIT command

```
:!DELETE↩
```

As a safety measure, DIRIT prints

```
ARE YOU SURE?
```

The user can respond with any of the following options:

S Deletes all files, printing nothing at the terminal until the job is completed.

L Lists at the terminal all file names as the files are being deleted.

D Prints the name of each file and waits for confirmation before deleting (same as the !DECIDE switch described below).

Y In response to ARE YOU SURE?, begins deleting all specified files. In response to individual file names presented by the D option, deletes the file in question.

N In response to ARE YOU SURE?, aborts the command and returns to DIRIT control level. In response to individual file names, retains the file in question and goes on to the next file name.

Q In response to ARE YOU SURE?, aborts the command and returns to DIRIT control level. In response to an individual file name, retains the file in question and aborts the command.

CAUTION: Do not follow these responses with carriage returns. Also, a carriage return in response to ARE YOU SURE? begins deleting all files.

The following examples show the effect of the optional responses to the DELETE command.

S

```
:!DELETE↩
ARE YOU SURE? S          All files are deleted with no terminal output.
:
```

L

```
:!DELETE↵
ARE YOU SURE? L

FILES DELETED:
LASTHC
GENEX3
GENEX2
:
```

All files are deleted, and their names are listed as they are deleted.

D, Y, N, and Q

```
:!DELETE↵
ARE YOU SURE? D

FILES DELETED:
LYNNFILE  ? N **NOT DELETED**
DATA2     ? Y
DATA3     ? Q **NOT DELETED**

:
```

Note that the user does not type a carriage return after any of his responses.

The user can limit the !DELETE command by including the switch !DECIDE. This is equivalent to answering D to the ARE YOU SURE? question. The program then responds by presenting each file name and waiting for a response before proceeding. The valid responses are N, Y, or Q, all explained in the list above. For example:

```
:!DELETE!DEC↵

FILES DELETED:
LYNFILE   ? Y
SMALTEXT  ? Y
MFIL      ? N **NOT DELETED**
@PRINT    ? Y

:
```

The DIRIT !DELETE command will not delete files that have been write protected.

During the execution of the !DELETE command, the file count is updated to provide the correct count for the !SUMMARY command (see page 98).

The user can select the files to be deleted by using any of the following file identifiers with the !DELETE command: file number, file type, file name, or partial file name. The correct forms of these identifiers are shown on pages 91–95. Files can also be selected by using the date commands presented on pages 90–91.

If the user wishes to delete files by file number, the files are presented in reverse order so that the numbers will not change as files are deleted. For example, the user has as the last three files in his directory:

```
-DIR,57↵
 # PVT PUB TYP  DATE USE  SIZE  NAME
57 R/W NO  SYM   3-16  6  1536  MFIL
58 R/W YES ,WT BIN   3-9   9  1536  SFOPROG-ACCTG
59 R/W NO  SYM   3-16  5  1536  FAKE

-
```

He wants to delete these files, and so the program presents them in reverse order:

```
:(57-59)!DELETE!DECIDE↵

FILES DELETED:
FAKE          ? Y
SFOPROG-ACCTG  ? Y
MFIL          ? N **NOT DELETED**

:
```

If the user wants to look at the names of all files created today and decide about their deletion, he types

```
:!DELETE!TODAY↵
ARE YOU SURE? D

FILES DELETED:
COM1    ? Y
TEXT    ? Y
MFIL    ? N **NOT DELETED**

:
```

In the following example, the user wants to delete the seventh through the ninth files, but as the names are presented he decides to delete none of them.

```
:(7-9)!DELETE↵
ARE YOU SURE? D

FILES DELETED:
SFPRG    ? N **NOT DELETED**
CATALOG   ? N **NOT DELETED**
REP      ? N **NOT DELETED**

:
```

In the following example, the user knows he has only one dump file in his directory, and he chooses to delete it without having the name presented for decision.

```
:.DUM!DELETE↩
ARE YOU SURE? Y

FILES DELETED:
/$/

:
```

The user can also sort the files to be deleted by any of the sort commands listed on pages 88 and 89. For example, the user wants to decide about deleting all the files in his directory that begin with AL, and he wants the files presented in alphabetical order. He uses the following command:

```
:AL:!DELETE!ALPHA!DECIDE↩

FILES DELETED:
ALPH1     ? Y
ALPH2     ? Y
ALPH3     ? Y
ALPHABET  ? N **NOT DELETED**

:
```

The user can also have the list of deleted files presented in any chosen order even though he does not want to decide about deleting individual files.

```
:AL:!DELETE!ALPHA↩
ARE YOU SURE? L

FILES DELETED:
ALP
ALPHABET
ALTER

:
```

The user specifies L so that the deleted files will be listed at the terminal.

Because the command ALPHA had been included in the command form, the list is alphabetical.

COPYING FILES TO ANOTHER DIRECTORY

The user can copy files from his directory to another directory in the same account provided that the receiving directory has public write access (see page 37). Once this condition is met, the user can move any or all of the files from his directory to the receiving directory. The basic form is

$:!\left\{\begin{array}{l}\text{TO}\\ \text{RENAME}\end{array}\right\}$(user name)[file identifier(s)]![DECIDE]↩

where the user name must meet the requirements presented above and the file identifier(s) can be either file name(s) or file number(s). If no file identifiers are specified, the whole directory can be copied.

The TO command copies the specified files to the new directory, but does not remove them from the originating directory. The !RENAME command copies the specified files to the new directory and deletes them from the originating directory.

As a precaution, if the !DECIDE switch is not used, DIRIT responds to a !TO or !RENAME command with

ARE YOU SURE?

Valid responses are the same as those for the !DELETE command described on page 99, with one exception. To prevent the loss of a file of the same name in the receiving directory, DIRIT responds with the EXECUTIVE OLD FILE/NEW FILE message when the !DECIDE option is in effect with the TO command. If the user wants the file copied, he responds with a carriage return. If he does not want the file copied, he responds with N. The following examples demonstrate the !TO and !RENAME commands:

```
:!TO (DELEON) NEWFILE↵
ARE YOU SURE? Y
:


:!TO (DELEON) (7:9)↵
ARE YOU SURE? Y
:


:!TO (MJL)(1-3)↵
ARE YOU SURE? D

FILES COPIED:
UPDATE  NEW FILE↵
ADDITIONS  NEW FILE N **NOT COPIED**
CON2  OLD FILE↵

:


:!TO (MJL) BAL,EOM,ANY↵
ARE YOU SURE? D

FILES COPIED:
BAL  NEW FILE↵
EOM  NEW FILE↵
ANY  NEW FILE N **NOT COPIED**

:
```

The !RENAME command does not overwrite an old file in the receiving directory. It responds with the message

CANNOT RENAME

```
:!RENAME (DELEON) HASH,ALPH1,CMDS,COM1↵
ARE YOU SURE? L

FILES RENAMED:
COM1
CMDS
ALPH1 **CANNOT RENAME**
HASH

:

:!RENAME (DELEON)(1-3)!D↵

FILES RENAMED:
GENEX2 ? Y
GENEX3 ? N **NOT RENAMED**
LASTHC ? Y

:


:!RENAME (DOMINSKI) LASTHC,GENEX2↵
ARE YOU SURE? D

FILES RENAMED:
LASTHC ? Y
GENEX2 ? Y

:
```

Section 11
DEFERRED AND PERIODIC PROCESSING

Tymshare's TYMCOM-IX system offers both deferred and periodic processing (PERP) capabilities.[1] Less urgent jobs can be processed most economically by entering them as deferred jobs and letting the system schedule their execution during minimum load periods. In deferred processing, the job is placed in a special queue to be executed, only once, on a deferred basis. In periodic processing, a deferred job is executed at user-specified intervals so that the user need not resubmit the job each time he wishes to execute the program.

The deferred and periodic processing commands allow the user to control the execution of his job as completely as if he were running it directly from the terminal. He may instruct the system to search for specific phrases in the output and to perform specified operations, including alteration of command sequence, based on that output. He may also control program pauses and interruptions.

DEFERRED PROCESSING

Deferred jobs are run during the night of the day they are entered, provided that they are entered before 10 p.m. local time. Any job entered after 10 p.m. will be run the following night.

Deferred processing requires the creation of a deferred processing file containing the commands to be executed. The commands in this file can be any combination of EXECUTIVE commands, language commands, and special deferred processing commands.

Deferred processing commands are identified by a preceding colon in the deferred processing file. Such commands may be abbreviated to the colon and three letters. For example, the :ESCAPE command can be given as :ESC.

The first line of the deferred processing file must contain a colon followed by the name of a terminal output file. During execution of the deferred processing command file, all output which would normally be printed at the terminal is written on this terminal output file. It will contain all error messages, normal output generated by the program or commands, and log out information.

It is not necessary to include the LOGOUT command in the deferred processing file, since the deferred processor automatically performs a log out after executing the last command on the file.

To insert the job into the deferred processing queue, the user types

```
—DEFER↵
```

at the EXECUTIVE command level. The system prints the current version number of the deferred processing program and then requests the name of the command file.

[1] – The same deferred and periodic processing capabilities are available on the TYMCOM-X.

—DEFER↵

VERS 1.0

COMMAND FILE:

The user enters the name of the deferred processing file and a carriage return. The program then builds the circuit to DEFER's home system and enters the job in the DEFER queue. For example:

```
-DEFER↵

VERS 1.0

COMMAND FILE: NITE↵

BUILDING CIRCUIT TO MASTER DEFER SYSTEM      It takes several seconds for the circuit to be
                                             built and the job to be entered.
DEFER JOB ENTERED

-
```

After the job has been processed by the system, a message is sent to the user's directory informing him of this fact. The next time he logs in, he is notified that a message is waiting. He can then display the terminal output file by using the TYPE or COPY command to obtain the program results, error messages, and processing time.

If an error causes the deferred job to be aborted, the user receives a message indicating the number of the line in which the error occurred. This line number corresponds to the EDITOR line number of the command file.[1] The user may read the command file into EDITOR and examine the improper line. The procedure for creating, running, and retrieving deferred jobs is illustrated in the example below.

```
-EDI↵                          The user creates the deferred file in EDITOR with the APPEND command.
*APPEND↵
:PERPOUT↵                      The output from the deferred job is written on the file PERPOUT.
DATE↵
COMMAND ADDITIONS↵             The program ADDITIONS updates and reports a small data base of
TIME↵                          stock holdings.
DELETE EXTRA↵                  The user terminates the APPEND mode with a control D and writes the
D^c                            contents on the file LATER.
*WRITE LATER↵
 NEW FILE↵
50 CHARS
*QUIT↵

-DEFER↵                        The user submits the job to the deferred processing queue.

VERS 1.0

COMMAND FILE: LATER↵
```

[1] – See the *Tymshare EDITOR Reference Manual.*

```
BUILDING CIRCUIT TO MASTER DEFER SYSTEM

DEFER JOB ENTERED

-
```

The user continues his other terminal duties and logs off. When he logs in the next day, he receives mail indicating that his job is complete. For example:

```
MAIL WAITING.

-MAIL↵

FROM DELEON +
3/18/76  8:27
```

The message is sent from the user's own directory when the deferred job has been completed.

```
FROM OPERATOR

YOUR JOB "LATER" SUBMITTED TO "DEFER"

HAS BEEN COMPLETED
-TYPE PERPOUT↵
```

The user types the output file to see the results of his job.

Note that the TYMCOM-IX commands and responses are printed on the terminal output file.

```
DATE
3/18/76  8:25
-COMMAND ADDITIONS
 NEW FILE


ACME LABS                    100
AMERICAL INSTRUMENT          100
CONTINENTAL BAKING           100
DIVERSIFIED CONTROL          100
EASTERN METALS               125
GOODSON & CO                 140
HOWARD, J.H.                 100
INT´L INDUSTRIES              12
LINCOLN PUBLISHING            50
NATIONAL PHARMACEUTICALS      64
ROYAL CHEMICAL                66
UNITED FURNITURE             135

TOTAL NUMBER OF SHARES IS 1092


-TIME

CPU TIME: 6 SECS.
TERMINAL TIME: 0:0:51

-DELETE EXTRA
```

```
-
```

The dashes assure the deferred processor that the job is in the EXECUTIVE mode.

```
-
```

```
LOG
```

The deferred processor automatically executes a LOG command at the end of each deferred job.

```
CPU TIME: 7 SECS.
TERMINAL TIME: 0:1:06

-
```

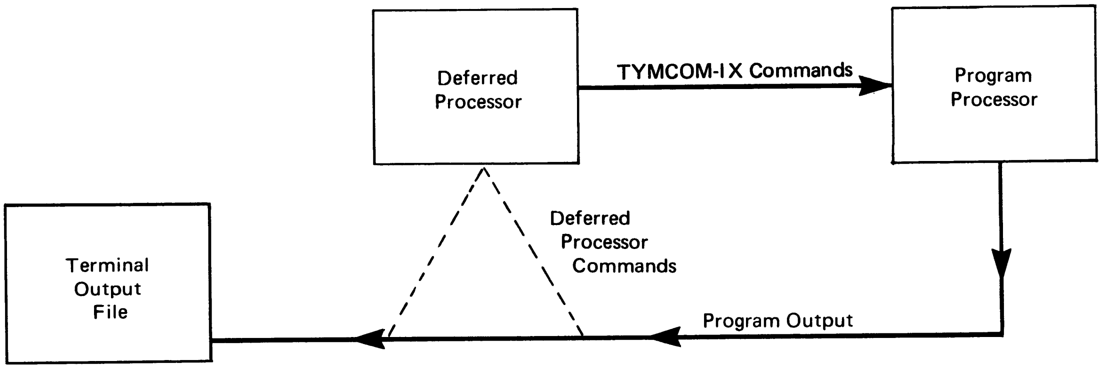
Control is returned to the EXECUTIVE command level after the terminal output file is completely printed.

The Processing Sequence

The deferred processing file contains two types of commands, TYMCOM-IX commands and deferred processing commands. TYMCOM-IX commands include EXECUTIVE commands, EDITOR commands, and all language commands. Any command which the user may execute from his terminal is a valid TYMCOM-IX command. TYMCOM-IX commands are sent directly to the program processor just as if the user had typed them. Program output which normally would appear at the terminal is written on the terminal output file.

Deferred processing commands, on the other hand, are not sent to the program processor. Their major functions are to analyze the characters being sent to the terminal output file and to create and process pauses in transmission of TYMCOM-IX commands.

The diagram below illustrates the relationship among the various elements in deferred processing.



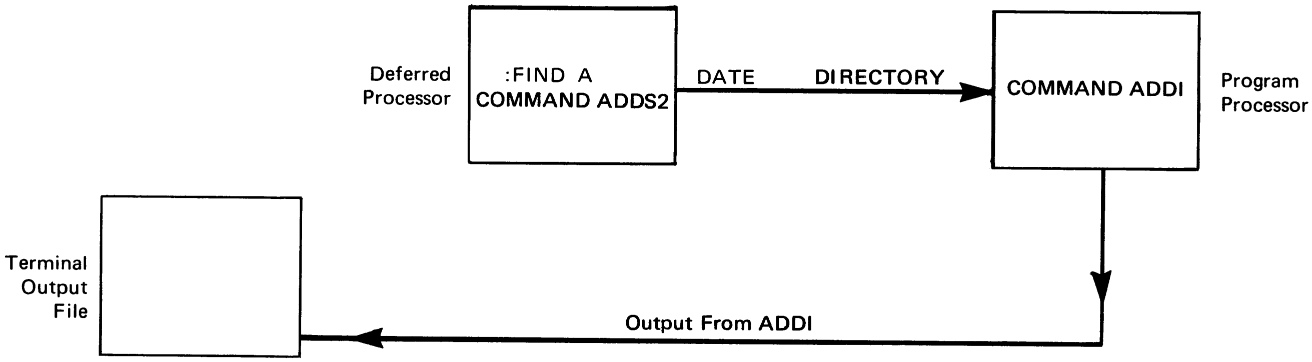
Before progressing to a discussion of the individual commands, the user should study the description of command processing below. He may find it convenient to refer to the diagrams as each pertinent command is discussed.

The normal sequence of operation for the deferred processor is to process a command, either a TYMCOM-IX command or a deferred processing command, if one is present, and then to check the stream of characters being transmitted to the terminal output file. Pauses occur when no command can be processed and all characters have been sent to the output file. The deferred processor determines when all characters have been sent to the terminal output file by sending a semicolon enclosed in quotation marks (";") through the program processor to the terminal output file and analyzing the stream of characters until the semicolon string has been transmitted.

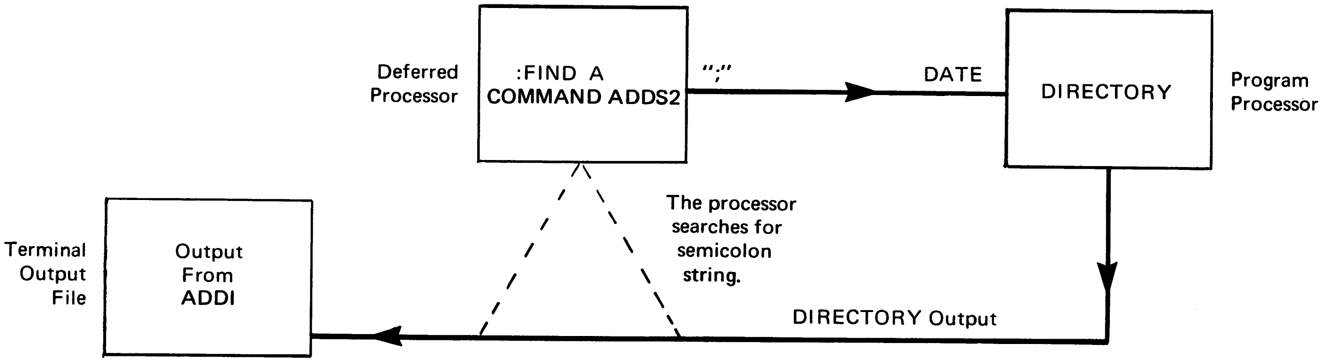
For example, the deferred processing command FIND causes a pause before it takes effect. Assume that the following commands are on the deferred processing file:

```
COMMAND ADDI
DIRECTORY
DATE
:FIND A
COMMAND ADDS2
```

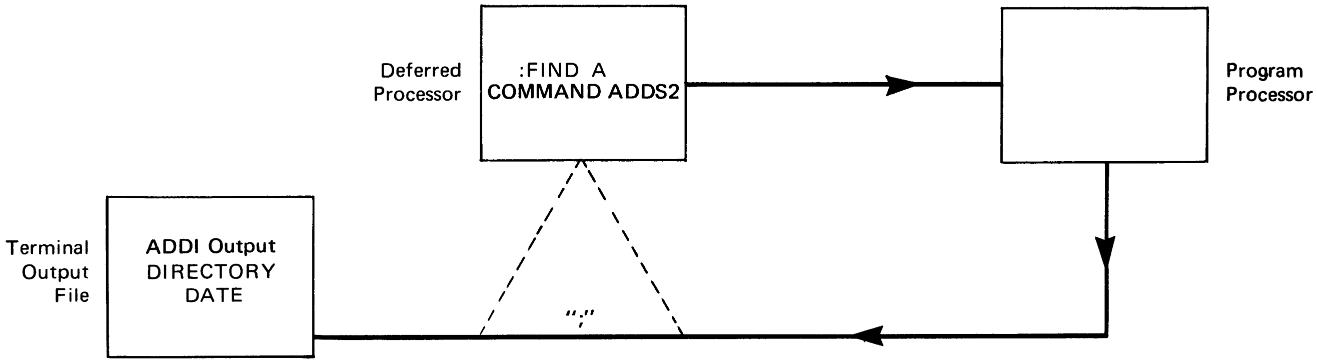
The diagrams below illustrate the sequence of operations. The program processor executes the file ADDI while the TYMCOM-IX DIRECTORY and DATE commands are waiting to be processed.



The :FIND A command is encountered by the deferred processor, which begins sending the semicolon string through the system and checking for its transmission to the terminal output file. Neither the FIND command nor any other commands which follow it in the deferred processing file are processed.



The deferred processor sends no additional commands until it discovers the semicolon string.



The system writes on the terminal output file all the output resulting from commands which precede FIND. The semicolon string is detected only after all this output is processed. As soon as it detects the semicolon string, the deferred processor processes the :FIND A command, which instructs it to begin searching for the string associated with A in the character output stream. The search applies only to output caused by commands following FIND. The deferred processor then sends the TYMCOM-IX COMMAND ADDS2 command to the program processor.

Searching for Phrases and Altering the Sequence of Commands

The user may instruct the system to search for specific phrases in the terminal output and to alter the normal sequence of commands within the command file based on the appearance of such phrases. There are three commands which initiate or control searches. These commands are DEFINE, FIND, and the assignment command. Labels and the commands GO, IF, and UNLESS control the continuity of execution.

The DEFINE Command

The DEFINE command identifies the characters for which the processor should search by associating the characters with a variable, named with a single letter from A to Z. As many as 26 such strings may be defined in a single command file. The variables may be considered as having an initial logical value of 0; the value is set to 1 by the processor when the string is found. Note that the variable names defined in this command and used in other deferred processor commands do not in any way correspond to variable names within a program which the user may be executing from the command file.

The general form of the DEFINE command is

:DEFINE string list

where the items in the string list are separated by commas, and each item is of the form

x='string'

where *x* is any letter of the alphabet and the string is a sequence of characters. The string may contain a maximum of 45 printing characters and may contain no line feeds or carriage returns. The user defines the beginning and end of the string with a character different from any character in the string. The string should be unique and as short as possible, because the fewer characters the processor must test the less computer time the search consumes.

For example, the following DEFINE command specifies strings for which the user wishes to search in an inventory program. In this case, the single quote mark (') defines the beginning and end of each string.

```
:DEFINE S='OUT OF STOCK',T='BACKORDERED'
```

Additional DEFINE commands may be used as long as the limit of 26 variables is not exceeded. A variable cannot be redefined.

NOTE: The DEFINE command simply identifies the characters for which the processor should search. It does not activate the search. The search itself is activated by the FIND or LOOK command. The DEFINE command must be used to identify strings for the FIND and LOOK commands.

The FIND Command

The FIND command activates a search for specific strings as output is written on the terminal output file. The form of the FIND command is

```
:FIND variable list
```

where the variables in the list must have been defined in a previous DEFINE command. For example:

```
:DEFINE K='COMPLETE',X='PROBLEM'
:FIND K,X
```

The string may appear as output from a program, an error message, or as the echo of a command contained in the user's command file. The search may be terminated in four ways—if the string is found, if the variable associated with the string is tested in an IF or UNLESS command, if the value of the associated variable is assigned with a direct assignment command, or if no string is found and the semicolon string is detected by the deferred processor.

The user will find it convenient to place the FIND command in the file just before he expects to encounter the string. The deferred processor pauses when the FIND command is encountered. The search activated by the FIND command applies only to output written on the terminal output file *after* the FIND command is processed.

The Assignment Command

The assignment command permits the user to set or reset the value of a variable. The form of the command is

```
:v=n
```

where *v* is a variable, with name from A to Z, and *n* is either 0 or 1. For example,

```
:J=1
:X=0
```

If the variable in the assignment command was being used for a search in a FIND command, the assignment command terminates the search for that variable. The variable need not be defined with a DEFINE statement.

NOTE: The variables in the assignment command do not in any way correspond to variable names in any program which the user may be executing from the command file.

Labels

Labels may be used within a deferred processing file to divide the file into segments. Labels consist of a series of characters preceded by two colons. Labels may contain any printing characters, but may not contain spaces or control characters. For example, the following are labels:

```
::REPORT
::ENDJOB
::ERROR1
```

The GO Command

The GO command changes the normal order of command execution within the deferred processing file, allowing the next command to be taken from the file immediately after a specified label.

The form of the GO command is

:GO m

where *m* is a label which is defined within the deferred processing file. The GO command may appear before or after the label to which it refers. For example:

:GO ERROR1

The IF and UNLESS Commands

The IF and UNLESS commands perform tests on a variable or group of variables and allow the user to execute a command conditionally, based on the value of the variables. The forms of these commands are

:IF e THEN c

and

:UNLESS e THEN c

where *e* is a variable expression, as defined below, and *c* is any deferred processing command except DEFINE. Note that the initial colon should not be included as part of the command *c*.

The IF command executes the command *c* if the variable expression *e* has the value 1. The UNLESS command executes the command *c* if the variable expression *e* equals 0. In each case, the command is ignored if the variable expression does not meet the test in the IF or UNLESS command; the next command in the file is processed instead.

Unless the command *c* transfers control to another part of the file or terminates execution, the next command executed is the succeeding line of the deferred processing command file.

Upon encountering the IF or UNLESS command, the processor tests to see whether the expression is already equal to the respective value. If such a value is present, no pause occurs. If the value is not yet present, the processor pauses to allow all previous output to be searched for the string. For example, if the processor encounters the command

```
:IF M THEN GO FINAL
```

it first tests to see whether the string associated with M has been located in the output. If so, the processor prepares to process the GO FINAL command. If the string associated with M has not yet appeared in the output, the processor causes a pause to search all output not yet written on the terminal output file. After all output has been searched, the test for M is made again; if it has not yet appeared, the GO FINAL command is not executed.

The variable expression *e* may be a single variable or a combination of variables using the symbols + and & to represent logical OR and AND, respectively.[1] The letters OR can be used instead of the plus sign; the letters AND can be used instead of the ampersand. If the word OR or AND is used, it must be preceded and followed by a space.

For example, the command

```
:IF Q+R THEN GO BEGIN
```

executes the GO command if either Q or R (or both) is equal to 1, indicating the associated string had been located in the terminal output or the user had assigned the value 1 to the variable.

The variable expression is evaluated from left to right; no parentheses are allowed. For example,

```
:IF X OR Y AND Z THEN QUIT
```

tests for the occurrence of Z and either X or Y or both.

Setting Time Limits

The timing commands are used to set limits on the amount of elapsed time the deferred processing job is allowed to run or on the amount of time a specific segment runs before control transfers to another part of the file.

If no timing command is set, the deferred processor allows each job a maximum of one hour in which to run. The user may set a lower maximum with the TIME command. In addition, he may use the TIME command to set a higher maximum to any length of time he desires. The form of the command is

```
:TIME n
```

where *n* is the number of minutes of elapsed time the job is allowed to run. If the time specified in the TIME command is less than the time already used by the job, the STOP command is executed immediately, and so no log out information is printed on the output file.

The form of the AFTER command is

```
:AFTER n THEN c
```

where *n* is an integer representing time in minutes, and *c* is any command except DEFINE. Note that the initial colon is not included in the command *c*. After the specified time has elapsed, the next command executed is that specified by *c*. Unless the command *c* is a GO command or a termination command, the command following the AFTER command is executed next.

[1] – The logical OR is an inclusive OR; the expression X OR Y is true if X is true, Y is true, or both X and Y are true.

For example,

```
:AFTER 2 THEN GO REPORT
```

transfers control to the label ::REPORT after two minutes have elapsed.

The command *c* may be omitted in the AFTER command. In this case, the form is

```
:AFTER n
```

and the commands following the AFTER command are executed only after the specified time has elapsed.

The TIME and AFTER commands operate dependently. When one of these commands is in effect, the other is postponed temporarily. For example, the following commands appear in the deferred file:

```
:TIME 5
.
.
.
:AFTER 20 THEN GO TEST
```

The TIME command sets the time limit to five minutes for the job. When the AFTER command is encountered, the 20-minute limit specified in the AFTER command supersedes the time specified in the TIME command limit. After the 20-minute limit expires, the 5-minute limit becomes effective again; because the job has exceeded five minutes (in fact it has exceeded 20 minutes), it is terminated with a STOP command.

Termination Commands

The QUIT and STOP commands terminate execution of the command file. These commands may be used individually or as part of an IF, UNLESS, or AFTER command. The individual forms are

```
:QUIT
```

and

```
:STOP
```

The QUIT command causes a pause to process all previous program output; QUIT then terminates the execution of the deferred processing job and writes the log out information on the terminal output file.

The STOP command immediately terminates the execution of the deferred processing job. No pause occurs and no further output is placed in the terminal output file. No log out information is written on the terminal output file if the STOP command is used.

Program Pauses

Four deferred processing commands discussed elsewhere in this section cause a program pause condition: FIND, QUIT, IF, and UNLESS. Note that IF and UNLESS do not cause pauses if the variable expression is equal to 1 or 0, respectively.

The user may create artificial pauses by having the program print a comment string and writing an IF command to test for the appearance of the string.

There are five additional deferred processing commands which affect program pauses; these commands are ESCAPE, PAUSE, CONTINUE, WAIT, and LOOK.

The ESCAPE Command

The ESCAPE command transfers control to the EXECUTIVE. If the processor is executing a program within a subsystem such as SUPER FORTRAN or EDITOR, the ESCAPE command terminates the execution of that program and executes the next command from the command file after a 10-second pause.

The PAUSE Command

The PAUSE command forces a pause, sending the semicolon string to test for the transmission of all output. PAUSE thus allows all output to be written on the terminal output file before processing the next command on the deferred processing file.

The CONTINUE and WAIT Commands

The semicolon string sent by the deferred processor during a pause may confuse a user's program if it is running within a subsystem such as SUPER FORTRAN or EDITOR. The CONTINUE command deactivates the automatic pauses associated with the commands FIND, IF, UNLESS, and QUIT. The CONTINUE command may appear at any point in the deferred processing file and applies to all FIND, IF, UNLESS, and QUIT commands which occur after the CONTINUE command is processed. If CONTINUE is in effect, these commands are processed immediately; no semicolons are sent. CONTINUE remains in effect until nullified by the WAIT command.

The WAIT command restores the normal mode of pausing before processing the commands FIND, IF, UNLESS, and QUIT.

The forms of these commands are:

```
:CONTINUE
```

and

```
:WAIT
```

The LOOK Command

When it is inconvenient to use the automatic pause or the PAUSE command, the user may give the LOOK command to force a pause in the program based on text generated by the user's program. The LOOK command combines the functions of the FIND and IF commands without the IF command arguments.

The form of the command is

```
:LOOK e
```

where *e* is a variable expression identical to that defined in the IF and UNLESS commands.

The LOOK command searches for the specified string in the output being written on the terminal output file. Since processing does not continue until the string is found, it is important to exercise care in choosing the string to be sought. If the string has been found previously by the FIND command, the condition is satisfied and the program continues.

Note that the LOOK command waits for output of the specified characters. The LOOK command never sends the special semicolon string to test for a pause.

Restarting Deferred Jobs

The user may issue the RESTART command to cause the processor to restart the job after a specified time. The form of the command is

```
:RESTART n
```

where *n* is the number of minutes of elapsed time before the job is started again.

The RESTART command is particularly useful when a file to be processed in the deferred job has not yet been created, or is not ready for processing. This gives the user an opportunity to delay job execution until the file is ready to be processed, whereupon the deferred job may begin again.

Multiple Deferred Jobs

The deferred processor can process several deferred jobs simultaneously. When a user submits several individual deferred jobs, they may be executed in an order different from that in which they were submitted. If order is important or if one deferred job must run to completion before another is begun, the user should include the jobs on the same deferred file.

Debugging Deferred Jobs

The program DDF is available to assist the user in debugging deferred jobs. DDF executes the deferred job immediately instead of submitting it as a deferred job. Thus, without waiting overnight, the user may determine if his deferred commands are accurately specified and arranged.

To access DDF, the user logs in and types DDF and a carriage return in the EXECUTIVE. The program requests the number of the computer on which the user is working and the name of the file containing the commands to be executed.

During the use of DDF, the user is using files on two systems simultaneously: the files needed for his deferred job, which are all in his directory on his system, and the DDF programs, which are all on the DDF home system. Thus the user is actually logged in to two computers simultaneously, and the computer charges are based on two machines being in use. When deferred jobs are run as scheduled by the deferred processor, however, this is not true; the user is charged only for the use of his home system.

DDF creates a second compacted copy of the deferred processing command file in the user's directory. The copied file, named //TEMP, is not deleted by DDF; the user must delete it.

After the temporary file is created, the deferred processing begins and DDF prints the message

STARTING JOB

on the user's terminal. DDF prints the message

END OF JOB

when the deferred processing is completed. The user can then examine the file //TOUT, which contains the terminal output generated by the command file and is a duplicate of the terminal output file named by the user if the job runs to completion without error.

During processing by DDF, the user may terminate the processing by typing an alt mode/escape. Processing ceases immediately and the user is returned to the EXECUTIVE. The //TOUT file and the //TEMP file remain in the user's directory for analysis.

The example below illustrates a sample DDF session.

```
-TYPE DAILY↵
```

The command file DAILY is in the user's directory on system 2.

```
:DAYLST
DATE
COMMAND DAY
TIME
DELETE BB
```

The output file is DAYLST.

```
-DDF↵
```

The user calls DDF and enters his job.

```
ENTER YOUR COMPUTER NO.:   2↵

TYPE YOUR COMMAND FILE NAME:  DAILY↵

STARTING JOB
  +
```

The job has been started.

```
COPYING TERMINAL OUTPUT
+
END OF JOB
```

The job has run to completion and the //TOUT file is being written.

```
MAIL WAITING

-MAIL↵
```

The user is notified that he has a message from the DDF operator. The user requests that the message be printed at the terminal.

```
FROM DELEON +
5/17/76  17:21
```

The message comes from the user's own directory, but is put there by the DDF operator.

```
FROM OPERATOR

YOUR JOB "DAILY" SUBMITTED TO "DEFER"

HAS BEEN COMPLETED
-TYPE //TOUT↵
```

The user prints out the //TOUT file, which in this case is identical to the output file DAYLST.

```
DATE
5/17/76  17:20
-COMMAND DAY
 NEW FILE
```

```
DAILY LIST OF PURCHASES           MAY 3, 1976

PPO   ITEM                                          COST
1162  BUS CARDS FOR F. IADIANO                      8.22
1163  LABELS FOR ER SHIPPING                       11.40
1164  STATIONERY FOR DOWNTOWN OFFICE               49.61
1165  BUS CARDS FOR C. MELLOR                       8.22
1166  STATIONERY FOR PRES. OFFICE                  16.64
1167  BUS CARDS FOR D. FARRER                      16.44
1168  LABELS FOR PRES. OFFICE                      32.86
1169  BUS CARDS FOR R. LUPINO                       8.22
1170  BUS CARDS FOR M. DELEON                       8.22
1171  BUS CARDS FOR M. DAY                          8.22

          TOTAL                                   168.05

-TIME
```

This is the TIME command that is part of the user's command file.

```
CPU TIME: 6 SECS.
TERMINAL TIME: 0:0:19

-DELETE BB

-


-
```

The user is logged out of the DDF home system.

```
LOG
```

This time message is the one normally printed by the system on log out.

```
CPU TIME: 7 SECS.
TERMINAL TIME: 0:0:23

-
```

Control is returned to EXECUTIVE command level.

Sample Deferred Jobs

This section illustrates two deferred jobs containing many of the commands described on the preceding pages.

Example 1

The following example illustrates labels and the commands DEFINE, TIME, AFTER, LOOK, ESCAPE, and GO. The user sets an overall time limit of 10 minutes on the whole job. He wishes to execute a SUPER BASIC program for only one minute. He uses the AFTER command to set the time and the GO command to transfer control to the label ::ENDJOB. The ESCAPE command interrupts the SUPER BASIC subsystem and returns control to the EXECUTIVE after a 10-second pause.

The first occurrence of the command LOOK A forces the processor to pause and not accept any further deferred processing commands, since it searches for a string which does not appear. The SUPER BASIC program continues to run, as the processor pauses do not affect execution of TYMCOM-IX commands or programs.

The second occurrence of LOOK A causes a pause until the string END OF JOB has been encountered before processing the final EXECUTIVE TIME command.

```
MAIL WAITING
```

When he logs in, the user is notified that he has a message waiting.

```
-MAIL↩
```

He requests that the message be printed on the terminal.

```
FROM DELEON +
4/29/76  7:44

FROM OPERATOR

YOUR JOB "LOOP" SUBMITTED TO "PERP"

HAS BEEN COMPLETED
-TYPE LOOP↩
```

The user displays the contents of the deferred processing file.

```
:ESC
:DEFINE A='END OF JOB'
:TIME 10
:AFTER 1 THEN GO ENDJOB
SBA
```

The user sets a maximum of 10 minutes for the command file. After one minute, the processor takes commands from the ENDJOB section.

```
10 FOR K=1 TO 100
20 FOR L=1 TO 1000
30 NEXT L
40 PRINT K
50 NEXT K
60 END
RUN
:LOOK A
```

The LOOK command halts the acceptance of deferred processing commands.

```
::ENDJOB
:ESCAPE
TIME
"END OF JOB"
:LOOK A
TIME
```

The quotation marks indicate that the comment END OF JOB is to be written on the terminal output file.

```
-TYPE ESC↩
```

The user displays the contents of the terminal output file. Note that the EXECUTIVE and SUPER BASIC commands are printed on the file, but that deferred processing commands are not printed.

```
SBA

>10 FOR K=1 TO 100
>20 FOR L=1 TO 1000
>30 NEXT L
>40 PRINT K
>50 NEXT K
>60 END
>RUN
```

```
1
2
3
4
5
6
7
8
9
10
11
•
•
•
26
27
```

The program executed the outer loop 27 times before the time limit expired.

```
-TIME

CPU TIME:  11 SECS.
TERMINAL TIME:  0:1:25

-"END OF JOB"
```

The END OF JOB comment is written on the file.

```
-TIME

CPU TIME:  11 SECS.
TERMINAL TIME:  0:1:47

-
```

The dashes assure the processor that the user is in EXECUTIVE before giving the LOG command.

```
-


LOG

CPU TIME:  12 SECS.
TERMINAL TIME:  0:1:59
```

The log out information is written on the output file unless termination occurs because of a STOP command or the exceeding of the time specified in a TIME command.

```
-
```

Example 2

This example illustrates the DEFINE, FIND, TIME, IF, GO, UNLESS, QUIT, and PAUSE commands. The user wishes to run his end-of-month accounting with the ACCTS program. If the ACCTS program runs properly and contains sorted data, he wishes to execute the REP program to generate a report. If the data is not sorted, he wants to run a special SORT program before running the REP program. The deferred processing commands allow him to run only the required programs based on output from the program ACCTS.

The deferred file EOM contains commands to print the report files produced and to delete certain scratch files from the user's directory.

```
MAIL WAITING

-MAIL↵

FROM DELEON +
4/30/76  22:18

FROM OPERATOR

YOUR JOB "EOM" SUBMITTED TO "PERP"

HAS BEEN COMPLETED
-TYPE EOM↵

:BAL
:DEFINE A='COMPLETED',B='ABORTED',C='REPORT ABORTED',D='SORTED'
:FIND A,B,D              The user does not need to begin the search for C until the ACCTS
:TIME 8                  program is complete. He sets a limit of eight minutes on his job.
GO ACCTS
:IF A&D THEN GO REPORT    Control transfers to the label ::REPORT only if both strings A and D
TIME                      are found.
:UNLESS D THEN GO SORT    Control transfers to the label ::SORT unless the string D is found.
::REPORT
:FIND C                  The search for string C begins just before the program REP is executed.
GO REP
TIME
:PAUSE
FILES REPT1,REPT2
:UNLESS C THEN GO DELETE  If the report aborts, no working files are deleted.
:QUIT
::SORT
GO SORT
:GO REPORT
::DELETE
DELETE SCR1,SCR2         The system deletes all the working files.
TIME

-




-TYPE BAL↵




-GO ACCTS

SORTED PROPERLY          Strings A and D are found.
ACCTS PROCESSED AND COMPLETED
```

```
-
```

The EXECUTIVE dashes are generated by the pause in the IF command.

```
-

-GO REP

REPORT COMPLETED
```

String C was not produced by the REP program.

```
-TIME

CPU TIME: 4 SECS.
TERMINAL TIME: 0:4:05

-

-FILES REPT1,REPT2

GO    REPT1
GO    REPT2

-

-DELETE SCR1,SCR2

-TIME

CPU TIME: 5 SECS.
TERMINAL TIME: 0:4:18

-
```

The dashes assure the processor that the user is in EXECUTIVE before giving the LOG command.

```
-


LOG

CPU TIME: 5 SECS.
TERMINAL TIME: 0:5:26

-
```

Summary of Deferred Processing Commands

The deferred processing commands are listed alphabetically in the table below. The parts of the commands in brackets are optional. Braces indicate that the user must include one of the options within the braces.

Command	Function
:file name	Specifies the name of the terminal output file.
::label	Begins a new segment of the deferred processing file.
:variable = $\left\{\begin{matrix}1\\0\end{matrix}\right\}$	Sets value of variable. Stops FIND search if one was in progress.
:AFTER n [THEN c]	Executes command *c* after *n* minutes elapse. If *c* is not specified, executes command following AFTER after *n* minutes elapse.
:CONTINUE	Deactivates the automatic pauses associated with the commands FIND, IF, UNLESS, and QUIT.
:DEFINE string list	Identifies the characters for which the processor is asked to search.
:ESCAPE	Transfers control to EXECUTIVE.
:FIND variable list	Initiates search for specified strings.
:GO label	Transfers control to the command immediately following the specified label.
:IF e THEN c	Executes command *c* if variable expression *e* has the value 1.
:LOOK e	Searches for the string corresponding to the variable expression and does not process the next command until it locates the string.
:PAUSE	Creates a pause in the processing of the user's job.
:QUIT	Causes a pause, then terminates the execution of the deferred processing job.
:STOP	Terminates the execution of the deferred processing job without causing a pause.
:TIME n	Sets limit of *n* minutes on elapsed time deferred job may run.
:UNLESS e THEN c	Executes command *c* if variable expression *e* has the value 0.
:WAIT	Restores the normal mode of pausing before processing the commands FIND, IF, UNLESS, and QUIT.
:RESTART n	Restarts the deferred job after *n* minutes.

PERIODIC PROCESSING

Periodic processing provides the capability to execute deferred jobs at specified intervals. The user may write his own expressions to indicate when a particular deferred job is to be executed; for example, he may specify that the job be executed every day, the fifteenth of every month, or every other Monday. He then gives simple commands to enter the deferred job into the periodic queue.

Periodic processing is especially useful for running repetitive jobs. Some examples of periodic processing applications are payrolls, quarterly reports, weekly forecasts, and daily analyses.

This section contains a description of the periodic processing commands, followed by a discussion of the specification of date expressions.

The user calls the periodic processing program from any TYMCOM-IX system by typing, in the EXECUTIVE, PERP and a carriage return. The program responds with a colon (:), indicating its readiness to accept commands. For example:

```
-PERP↵

:
```

The user may enter a PERP command after the colon prompt. He should wait until the colon appears and should not type ahead.

Control A, control W, and control Q can all be used for editing during command entry.[1]

There are 12 periodic processing commands, which perform the following functions:

- Enter date and job information into the processing queue
- Delete jobs from the periodic processing queue
- Print out date and job information at the user's terminal
- Aid the PERP user

The DATES Command

The DATES command allows the user to enter the date specification for a job. This specification may be as simple or as complex as the user needs to define the scheduling of the deferred job. See page 129 for a complete explanation of the possible date specification forms.

The date expression must be terminated with a period (.). Thus, no periods may appear in the date expression except at the end. The expression may be continued on additional lines, if necessary, by typing a carriage return at the end of the line. Only the final line may contain a period at the end of the line.

The example below illustrates the use of the DATES command to specify that the job be executed on the second day of every month.

```
:DATES↵

ENTER YOUR DATE PERIOD
THE 2ND DAY OF EVERY MONTH.↵

:
```

The INSERT Command

The INSERT command enters a job into the periodic processing queue. The job is executed according to the date specification in the most recent DATES command.

The INSERT command requests the hour that the job should run, the number of the computer on which the job should run, and the name of the file containing the commands to be executed.

The user enters the INSERT command followed by a carriage return. The system prints:

```
WHAT HOUR (0 TO 23)?
```

[1] – Editing with control characters is discussed on page 9.

The user may enter an integer from 0 to 23 specifying the time the job should run. The number 0 represents midnight; the number 23 represents 11 p.m. These times correspond to the user's local time, that is, the time displayed when the user logs in.

The user may enter the word ANY to specify that the job may be run at any hour during the day, or the word DEFER to indicate that the job should be run as a deferred job.

NOTE: The user should specify an execution time within the scheduled machine availability for the system on which the job is to be run. If the machine is not available at the requested time, the job is executed at the machine's next availability.

The system then requests the computer number, project code, and name of the command file by printing:

COMPUTER?

PROJECT CODE?

COMMAND FILE NAME?

The user's command file must be located in his file directory on the computer specified in the INSERT command. Note that the user name under which the user logged in to call the PERP program must be identical to the user name in the directory in which the file is located.

The example below illustrates the description and entry of a deferred periodic job LOOP to be executed every Monday on computer 2 at 5 a.m.

```
-PERP↵

:DATE↵

ENTER YOUR DATE PERIOD
EVERY MONDAY.↵

:INSERT↵

WHAT HOUR (0 TO 23)? 5↵

COMPUTER? 2↵

PROJECT CODE? ZIP↵

COMMAND FILE NAME? LOOP↵

BUILDING CIRCUIT TO MASTER PERP SYSTEM


:
```

The command file requested must be a deferred processing command file; see page 105.

Just as in deferred processing, PERP sends mail to the user after each execution of the deferred job.

The DELETE Command

The DELETE command removes a specified job from the periodic processing queue; the job is no longer executed at the previously specified intervals. The user may enter it into the queue again with the DATES and INSERT commands.

The DELETE command requests the computer number, project code, and the name of the command file. It then removes that file from the periodic processing queue; it does not, however, delete the file from the user's directory on that computer.

NOTE: The user name from which the DELETE command is given must be the same from which the corresponding INSERT command was given, and the project code must also be the same.

Example

The user removes job EOW from the periodic processing queue on computer 2.

```
:DELETE⤶

COMPUTER? 2⤶

PROJECT CODE? ZX⤶

COMMAND FILE NAME? EOW⤶
```

The EXPRESS Command

The EXPRESS command prints the date specification entered in the last DATES command. An expression printed by the EXPRESS command contains only the key words PERP used to form the actual dates for job execution. For example:

```
:DATES⤶

ENTER YOUR DATE PERIOD
THE FIFTH DAY OF THE MONTH.⤶

:EXPRESS⤶
THE 5TH DAY OF EVERY MONTH


:DATES⤶

ENTER YOUR DATE PERIOD
EVERY OTHER MONTH.⤶

:EXPRESS⤶
EVERY 2 MONTHS


:DATES⤶
```

```
ENTER YOUR DATE PERIOD
THE FIFTH DAY OF THE MONTH AND EVERY OTHER FRIDAY.↩

:EXPRESS↩
THE 5TH DAY OF EVERY MONTH

OR EVERY 2ND FRIDAY

:
```

The DISPLAY Command

The DISPLAY command prints a list of all dates on which the last requested job is scheduled to run. The user may use this list to verify that the job runs on the dates he wishes.

The dates are printed chronologically, showing the month, date, year, and day of the week.

The date list includes all dates for the next few months or until a user-specified termination date. To terminate the printing of the date list, the user types an alt mode/escape.

The example below illustrates the DISPLAY command with simple date expressions.[1] For examples of more complex expressions, see pages 135, 137, and 139.

```
:DATES↩

ENTER YOUR DATE PERIOD
THE 20TH DAY OF EVERY MONTH.↩

:DISPLAY↩

MAR 20, 1976 SATURDAY
APR 20, 1976 TUESDAY
MAY 20, 1976 THURSDAY
JUN 20, 1976 SUNDAY
JUL 20, 1976 TUESDAY
AUG 20, 1976 FRIDAY⊕
```

The user interrupts the printing of the date list by typing an alt mode/escape.

```
:
```

The TEST Command

The TEST command is equivalent to the EXPRESS command followed by the DISPLAY command. TEST prints the date specification, then a list of the dates corresponding to the specification. For example:

```
:DATES↩

ENTER YOUR DATE PERIOD
THE 1ST DAY OF THE MONTH.↩

:TEST↩
THE FIRST DAY OF EVERY MONTH
```

[1] – All examples in this section were run on March 12, 1976.

```
APR  1, 1976 THURSDAY
MAY  1, 1976 SATURDAY
JUN  1, 1976 TUESDAY
JUL  1, 1976 THURSDAY⊕
```

The user interrupts the date list.

```
:
```

The STATUS Command

The STATUS command prints the next date on which a specified job is scheduled to run. The STATUS command requests the number of the computer and the name of the corresponding command file. For example:

```
:STATUS↵

COMPUTER? 2↵

PROJECT CODE? ZX↵

COMMAND FILE NAME? LATER↵

JOB TO BE DONE ON: MAR 15, 1976 MONDAY     4:59
:
```

The user finds that the file LATER on computer 2 is scheduled to be executed next at 4:59 a.m. on March 15.

The GET Command

The GET command finds a job previously entered in the PERP queue so that the user can get information about its schedule. The GET command requests the number of the computer, the project code, and the name of the command file. When it locates the job, PERP prints a prompt, and the user can then enter the DISPLAY, EXPRESS, or TEST command. The GET command does not delete the information from the job queue. If the user wishes to change any information for the job, he must delete it, then insert it in the queue as a new entry. An example of the GET command follows.

```
:GET↵

COMPUTER? 2↵

PROJECT CODE? ZX↵

COMMAND FILE NAME? LATER↵

:TEST↵
EVERY MONDAY

MAR 15, 1976 MONDAY
MAR 22, 1976 MONDAY
MAR 29, 1976 MONDAY
APR  5, 1976 MONDAY
APR 12, 1976 MONDAY
APR 19, 1976 MONDAY
APR 26, 1976 MONDAY
MAY  3, ⊕

:
```

The LIST Command

The LIST command lists all jobs in the periodic processing queue entered under a particular user name, along with pertinent scheduling information. For example,

```
-PERP↩

:LIST↩

BUILDING CIRCUIT TO MASTER PERP SYSTEM
MAR 15, 1976 MONDAY     4:59, CPN: 2, FIADIANO;ZX:LATER
MAR 15, 1976 MONDAY    16:00, CPN: 2, FIADIANO;ZX:STATS
MAR 15, 1976 MONDAY    21:00, CPN: 2, FIADIANO;ZX:BYMO
END OF JOB
:
```

If the user has been working in PERP, this message will not appear.

Date and time the job is to be run

Computer number; this job is on system 2

User name

Project code

Name of PERP command file

Utility Commands

PERP contains three utility commands: QUIT, HELP, and VERSION. QUIT returns control to the EXECUTIVE after a PERP session. The HELP command prints a list of PERP commands and their descriptions. The VERSION command prints the number of the current version of PERP.

Date Specifications

The periodic processing user may select the exact execution dates for his job by using date specifications in the DATES command; these specifications may be as simple or as complex as required. Each time a periodic job is run, PERP computes the next date on which the job is scheduled to run. If the current date meets the criterion, the job is run on that date also.

The date specification must be terminated by a period. Thus, no periods may appear in the date specification except at the end. For completeness, the user may include the words OF and THE in the date specifications; PERP, however, does not require these words.

NOTE: The user should use the DISPLAY command to assure that his date specification produces the exact execution dates he requires.

Date specifications may consist of basic expressions alone or combined in groups and series. These units are discussed in detail after the definitions below.

The simplest part of a date specification is an expression. Examples of expressions are:

EVERY WEDNESDAY.

THIRD FRIDAY OF EVERY THIRD MONTH.

Expressions may be linked together with the words AFTER, BEFORE, AND, and OR to form a group. An example of a group with two expressions is:

10 DAYS AFTER THE FIRST MONDAY OF EVERY MONTH.

Several groups connected together comprise a series. A series begins with the word EARLIEST or LATEST or the phrase DO IN ORDER. For example:

DO IN ORDER MONDAY AND FEB 6, 1976 AND MAR 6, 1976 AND WED.

Basic Expressions

An expression may be one of six types. These types are listed in the table below. If an expression specifies a number of months, with no modifiers, the job runs on the first day of that month.

Months and days of the week may be shortened to the first three characters. No period may be used with such an abbreviation, because a period specifies the end of a date expression.

Type of Expression	Examples	
Day of week	EVERY OTHER MONDAY. EVERY WEDNESDAY.	
Number of days	EVERY 5 DAYS. EVERY 10 DAYS.	
Number of months	EVERY MONTH. EVERY 4 MONTHS.	
Day of month	THE 5TH DAY OF EVERY MONTH. THE LAST MONDAY OF EVERY MONTH.	
Specific dates	JULY 4, 1973. APR 28, 1973. The form is *month day, year* with no space before the comma.	
Special words	TODAY. TOMORROW.	WEEKDAYS. DAILY. MONTHLY.

These basic expressions are illustrated in the examples below:

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY MONDAY.↵
```

The date specification must end with a period.

```
:DISPLAY↵

MAR 15, 1976 MONDAY
MAR 22, 1976 MONDAY
MAR 29, 1976 MONDAY
APR  5, 1976 MONDAY
APR 12, 1976 MONDAY
APR 19, 1976 MONDAY
APR 26, 1976 MONDAY
MAY  3,⊕
```

An alt mode/escape terminates the date listing.

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY OTHER TUESDAY.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
MAR 30, 1976 TUESDAY
APR 13, 1976 TUESDAY
APR 27, 1976 TUESDAY
MAY 11, 1976 TUESDAY
MAY 25, 1976 TUESDAY
JUN ⊕

:DATES↵

ENTER YOUR DATE PERIOD
EVERY 5 DAYS↵
.↵
```

The user forgets to terminate the entry with a period, and so adds it on the next line.

```
:DISPLAY↵

MAR 16, 1976 TUESDAY
MAR 21, 1976 SUNDAY
MAR 26, 1976 FRIDAY
MAR 31, 1976 WEDNESDAY
APR  5, 1976 MONDAY
APR 10, 1976 SATURDAY
APR 15,⊕
```

In an expression such as EVERY 5 DAYS, the current date is not considered as meeting the criterion.

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY MONTH.↵
```

EVERY MONTH refers to the first day of every month.

```
:DISPLAY↩

APR  1, 1976 THURSDAY
MAY  1, 1976 SATURDAY
JUN  1, 1976 TUESDAY
JUL  1, 1976 THURSDAY
AUG  1, 1976 SUNDAY
S⊕

:DATES↩

ENTER YOUR DATE PERIOD
THE 5TH DAY OF EVERY OTHER MONTH.↩

:DISPLAY↩

MAY  5, 1976 WEDNESDAY
JUL  5, 1976 MONDAY
SEP  5, 1976 SUNDAY

NO FURTHER DATES
```

Dates are printed out only for the next few months.

```
:DATES↩

ENTER YOUR DATE PERIOD
EVERY 3RD MONDAY OF THE MONTH.↩

:DISPLAY↩

MAR 15, 1976 MONDAY
APR 19, 1976 MONDAY
MAY 17, 1976 MONDAY
JUN 21, 1976 MONDAY
JUL 19, 1976 MONDAY
AU⊕

:DATES↩

ENTER YOUR DATE PERIOD
THE LAST FRIDAY OF EVERY 3RD MONTH.↩

:DISPLAY↩

MAR 26, 1976 FRIDAY
JUN 25, 1976 FRIDAY
SEP 24, 1976 FRIDAY

NO FURTHER DATES

:DATES↩

ENTER YOUR DATE PERIOD
THE LAST DAY OF THE MONTH.↩
```

```
:DISPLAY↵

MAR 31, 1976 WEDNESDAY
APR 30, 1976 FRIDAY
MAY 31, 1976 MONDAY
JUN 30, 1976 WEDNESDAY
JUL 31, 1976 SATURDAY

:DATES↵

ENTER YOUR DATE PERIOD
APRIL 17, 1976.↵

:DISPLAY↵

APR 17, 1976 SATURDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
APRIL 23, 1976.↵

:DISPLAY↵

APR 23, 1976 FRIDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
TODAY.↵

:DISPLAY↵

MAR 12, 1976 FRIDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
TOMORROW.↵

:DISPLAY↵

MAR 13, 1976 SATURDAY

NO FURTHER DATES

:
```

Date specifications are not cumulative. Only the expression in the most recent DATES command is used to form the date list.

When a month expression appears, PERP calculates dates from the first day of the month. For example, the expresssion

THE 50TH DAY OF EVERY MONTH.

causes PERP to calculate from the beginning of each month. The fiftieth day of September, for example, occurs on October 20.

```
:DATES↵

ENTER YOUR DATE PERIOD
THE 45TH DAY OF EVERY MONTH.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
APR 14, 1976 WEDNESDAY
MAY 15, 1976 SATURDAY
JUN 14, 1976 MONDAY
JUL 15, 1976 THURSDAY
AUG 14, 1976 SATURDAY
SEP 14, 1976 TUESDAY

NO FURTHER DATES

:
```

The forty-fifth day of February is March 16.

Similarly, in an expression such as

EVERY 5TH WEDNESDAY OF THE MONTH.

each month's calculation includes a fifth Wednesday; for some months, however, it is the first Wednesday of the following month. The examples below illustrate this type of expression.

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY 5TH WEDNESDAY OF THE MONTH.↵

:DISPLAY↵

MAR 31, 1976 WEDNESDAY
MAY  5, 1976 WEDNESDAY
JUN  2, 1976 WEDNESDAY
JUN 30, 1976 WEDNESDAY
AUG  4, 1976 WEDNESDAY
SEP  1, 1976 WEDNESDAY
SEP 29, 1976 WEDNESDAY

NO FURTHER DATES

:
```

```
:DATES↩

ENTER YOUR DATE PERIOD
EVERY 5TH WEDNESDAY OF THE MONTH↩
EXCEPT THE 1ST WEDNESDAY OF THE MONTH.↩

:DISPLAY↩

MAR 31, 1976 WEDNESDAY
JUN 30, 1976 WEDNESDAY
SEP 29, 1976 WEDNESDAY

NO FURTHER DATES

:
```

This expression eliminates any Wednesdays which are not the fifth calendar Wednesday.

Groups of Basic Expressions

Simple or basic expressions may be combined with the words AFTER, BEFORE, AND, or OR to form expression groups. The connectors AND and OR are equivalent and cause the job to run on all dates which meet the conditions on either side of the connecting word.

When the date specification includes the word AND or OR, PERP computes the next execution date according to the earliest subsequent date in the group.

The examples below illustrate various expression groups:

```
:DATES↩

ENTER YOUR DATE PERIOD
THE FIRST WEDNESDAY AFTER THE 5TH DAY OF EVERY MONTH.↩

:DISPLAY↩

APR  7, 1976 WEDNESDAY
MAY 12, 1976 WEDNESDAY
JUN  9, 1976 WEDNESDAY
JUL  7, 1976 WEDNESDAY
AUG 11, 1976 WEDNESDAY
SEP  8, 1976 WEDNESDAY

NO FURTHER DATES

:DATES↩

ENTER YOUR DATE PERIOD
FRIDAY AFTER THE FIRST WEDNESDAY OF EVERY MONTH.↩

:DISPLAY↩

APR  9, 1976 FRIDAY
MAY  7, 1976 FRIDAY
JUN  4, 1976 FRIDAY
```

```
JUL  9, 1976 FRIDAY
AUG  6, 1976 FRIDAY
SEP  3, 1976 FRIDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
5 DAYS BEFORE THE END OF THE MONTH.↵

:DISPLAY↵

MAR 27, 1976 SATURDAY
APR 26, 1976 MONDAY
MAY 27, 1976 THURSDAY
JUN 26, 1976 SATURDAY
JUL 27, 1976 TUESDAY
AUG 27, 1976 FRIDAY⊕

:DATES↵

ENTER YOUR DATE PERIOD
EVERY FRIDAY BEFORE THE END OF THE MONTH.↵

:DISPLAY↵

MAR 26, 1976 FRIDAY
APR 30, 1976 FRIDAY
MAY 28, 1976 FRIDAY
JUN 25, 1976 FRIDAY
JUL 30, 1976⊕

:DATES↵

ENTER YOUR DATE PERIOD
EVERY MONDAY AND EVERY FRIDAY.↵

:DISPLAY↵

MAR 12, 1976 FRIDAY
MAR 15, 1976 MONDAY
MAR 19, 1976 FRIDAY
MAR 22, 1976 MONDAY
MAR 26, 1976 FRIDAY
MAR 29, 1976 MONDAY
APR  2, 1976 FRIDAY⊕

:DATES↵

ENTER YOUR DATE PERIOD
1ST DAY OF MONTH AND 20TH DAY OF MONTH.↵
```

```
:DISPLAY↵

MAR 20, 1976 SATURDAY
APR  1, 1976 THURSDAY
APR 20, 1976 TUESDAY
MAY  1, 1976 SATURDAY
MAY 20, 1976 THURSDAY
JUN  1, 1976 TUESDAY
JUN 20, 1976 ⊕

:DATES↵

ENTER YOUR DATE PERIOD
1ST DAY OF MONTH AND 10TH DAY OF MONTH↵
AND EVERY FRIDAY BEFORE THE END OF THE MONTH.↵

:DISPLAY↵

MAR 26, 1976 FRIDAY
APR  1, 1976 THURSDAY
APR 10, 1976 SATURDAY
APR 30, 1976 FRIDAY
MAY  1, 1976 SATURDAY
MAY 10, 1976 MONDAY
MAY 28, 1976 FRIDAY
JUN  1, 1976 TUESDAY
JUN 10, 1976 THURSDAY
⊕

:DATES↵

ENTER YOUR DATE PERIOD
APRIL 15, 1976 AND JUNE 1, 1976.↵

:DISPLAY↵

APR 15, 1976 THURSDAY
JUN  1, 1976 TUESDAY

NO FURTHER DATES

:
```

Expression Series

The PERP user may form series of expressions with the words DO IN ORDER, EARLIEST, and LATEST. DO IN ORDER causes the list of expressions to be processed once each in succession. The EARLIEST specification causes the expressions following it to be examined for the earliest of the remaining execution dates. This examination occurs repeatedly until the job is removed from the periodic processing queue. LATEST is similar to EARLIEST, except that LATEST determines the latest of the remaining dates.

NOTE: The EARLIEST, LATEST, and DO IN ORDER phrases begin a series. Thus, each phrase cancels the effect of any other such phrases in an expression.

The examples below illustrate expression series.

```
:DATES↵

ENTER YOUR DATE PERIOD
DO IN ORDER MARCH 25, 1976 AND MAY 17, 1976 AND MONDAY.↵

:DISPLAY↵

MAR 25, 1976 THURSDAY
MAY 17, 1976 MONDAY          The system includes the Monday that falls in order after May 17.
MAY 24, 1976 MONDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
MARCH 25, 1976 AND MAY 17, 1976 AND MONDAY.↵

:DISPLAY↵

MAR 15, 1976 MONDAY          Without DO IN ORDER, the system includes the first Monday after
MAR 25, 1976 THURSDAY        today's date, Mar 15.
MAY 17, 1976 MONDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
MARCH 25, 1976 AND APRIL 17, 1976 AND EVERY MONDAY.↵

:DISPLAY↵

MAR 15, 1976 MONDAY          Here the user has specified EVERY MONDAY.
MAR 22, 1976 MONDAY
MAR 25, 1976 THURSDAY
MAR 29, 1976 MONDAY
APR  5, 1976 MONDAY
APR 12, 1976 MONDAY
APR 17, 1976 SATURDAY
APR 19, 1976 MONDAY
APR 26, 1976 MONDAY
MAY  3, 1976 ⊕

:DATES↵

ENTER YOUR DATE PERIOD
DO IN ORDER MARCH 25, 1976 AND MONDAY AND APRIL 20, 1976.↵
```

```
:DISPLAY↵

MAR 25, 1976 THURSDAY
MAR 29, 1976 MONDAY
APR 20, 1976 TUESDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
THE LATEST OF THE FIRST THURSDAY OF THE MONTH AND THE FIRST WEDNESDAY↵
OF THE MONTH.↵

:DISPLAY↵

APR  7, 1976 WEDNESDAY
MAY  6, 1976 THURSDAY
JUN  3, 1976 THURSDAY
JUL  7, 1976 WEDNESDAY
AUG  5, 1976 THURSDAY⊕

:
```

Modifiers for Date Specifications

There are five modifiers which PERP accepts with date specifications. These modifiers are REFERENCE, STARTING, UNTIL, THROUGH, and EXCEPT. With such modifiers, the user has complete flexibility to request exact dates on which he wants the job to run.

The REFERENCE modifier specifies the reference dates from which all other dates are to be calculated. For example, if the job is entered on May 15 to run on the first day of every other month, the current date is used as the reference date, and the job runs on July 1, September 1, November 1, etc. Thus, the first month is skipped. If the reference date was April 7, the job would run on June 1, August 1, October 1, etc.

NOTE: The reference date may occur before, after, or on the date of job entry.

The examples below illustrate jobs entered on March 12, 1976, with and without reference dates:

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY 5 DAYS.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
MAR 21, 1976 SUNDAY
MAR 26, 1976 FRIDAY
MAR 31, 1976 WEDNESDAY
APR  5, 1976 MONDAY
APR 10, 1976 SATURDAY
APR 15, 1976 THURSDAY
AP⊕

:DATES⮐

ENTER YOUR DATE PERIOD
REFERENCE MARCH 15, 1976 EVERY 5 DAYS.⮐

:DISPLAY⮐

MAR 15, 1976 MONDAY
MAR 20, 1976 SATURDAY
MAR 25, 1976 THURSDAY
MAR 30, 1976 TUESDAY
APR  4, 1976 SUNDAY
⊕

:
```

The REFERENCE modifier is extremely powerful. PERP normally calculates each succeeding date based on the date on which the job runs. Some dates may never be reached unless a REFERENCE modifier is used. For example, if the user gives the date expression

MONDAY AND EVERY OTHER FRIDAY.

the job runs first on Monday. PERP then calculates the date of the following Monday and the second Friday from the current date. The date for Monday occurs first. This process is repeated each time the job is run; the job never runs on Friday. However, if the date expression includes a REFERENCE modifier, PERP calculates each date on the basis of the reference date, and so all specified dates are included.

In the following example, by including the REFERENCE modifier JANUARY 1, 1976, the user gets his job to run as requested on Monday and every other Friday.

```
:DATES⮐

ENTER YOUR DATE PERIOD
REFERENCE JANUARY 1, 1976 EVERY MONDAY AND EVERY OTHER FRIDAY.⮐

:DISPLAY⮐

MAR 12, 1976 FRIDAY
MAR 15, 1976 MONDAY
MAR 22, 1976 MONDAY
MAR 26, 1976 FRIDAY
MAR 29, 1976 MONDAY
APR  5, 1976 MONDAY
APR  9, 1976 FRIDAY
APR 12, 1976 MONDAY
A⊕

:
```

The STARTING modifier specifies the first date on which the job should run. If STARTING is used with the REFERENCE modifier, the REFERENCE date is used for calculation, but the STARTING date determines the first date used.

The examples below illustrate the effect of the STARTING date with and without the REFERENCE modifier.

```
:DATES↵

ENTER YOUR DATE PERIOD
STARTING MARCH 20, 1976 EVERY 30 DAYS.↵

:DISPLAY↵

APR 19, 1976 MONDAY
MAY 19, 1976 WEDNESDAY
JUN 18, 1976 FRIDAY
JUL 18, 1976 SUNDAY
AUG 17, 1976 TUESDAY
SEP 16, 1976 THURSDAY

NO FURTHER DATES
```

PERP calculates every 30 days from March 20, 1976. Note that March 20 is not included.

```
:DATES↵

ENTER YOUR DATE PERIOD
REFERENCE JANUARY 17, 1976 STARTING MAY 18, 1976 EVERY 30 DAYS.↵
```

The starting and reference dates are different.

```
:DISPLAY↵

JUN 15, 1976 TUESDAY
JUL 15, 1976 THURSDAY
AUG 14, 1976 SATURDAY
SEP 13, 1976 MONDAY

NO FURTHER DATES

:
```

PERP uses January 17 to calculate every thirtieth day, but ignores the dates preceding May 18.

The UNTIL and THROUGH modifiers specify the final date to which the expression applies. The UNTIL modifier refers to all dates up to but not including the date in the UNTIL phrase. The THROUGH modifier performs the same function as UNTIL except that THROUGH includes the specified date.

The UNTIL and THROUGH modifiers may appear several times in a single expression, modifying the expression immediately following the UNTIL or THROUGH date. Each modifier remains in effect until another is encountered.

```
:DATES↵

ENTER YOUR DATE PERIOD
THROUGH JULY 20, 1976 EVERY 6TH TUESDAY.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
APR 27, 1976 TUESDAY
JUN  8, 1976 TUESDAY
JUL 20, 1976 TUESDAY

NO FURTHER DATES

:DATES↵

ENTER YOUR DATE PERIOD
UNTIL JULY 20, 1976 EVERY 6TH TUESDAY.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
APR 27, 1976 TUESDAY
JUN  8, 1976 TUESDAY

NO FURTHER DATES

:
```

If two or more UNTIL modifiers are combined, a REFERENCE date should be included.

```
:DATES↵

ENTER YOUR DATE PERIOD
REFERENCE MARCH 1, 1976↵
UNTIL APRIL 15, 1976 EVERY WEDNESDAY↵
UNTIL MAY 30, 1976 EVERY OTHER FRIDAY.↵

:DISPLAY↵

MAR 17, 1976 WEDNESDAY
MAR 19, 1976 FRIDAY
MAR 24, 1976 WEDNESDAY
MAR 31, 1976 WEDNESDAY
APR  2, 1976 FRIDAY
APR  7, 1976 WEDNESDAY
APR 14, 1976 WEDNESDAY
APR 16, 1976 FRIDAY
APR 30, 1976 FRIDAY
MAY 14, 1976 FRIDAY
MAY 28, 1976 FRIDAY

NO FURTHER DATES

:
```

The user may exclude dates by using the EXCEPT modifier with a simple or complex expression. Note that EXCEPT divides the expression into two parts: The first portion selects dates; that portion after EXCEPT rejects dates.

```
:DATES↵

ENTER YOUR DATE PERIOD
EVERY TUESDAY EXCEPT MARCH 23, 1976.↵

:DISPLAY↵

MAR 16, 1976 TUESDAY
MAR 30, 1976 TUESDAY
APR  6, 1976 TUESDAY
APR 13, 1976 TUESDAY
APR 20, 1976 TUESDAY
APR 27, 1976 TUESDAY
MAY  4, 1976 TUESDAY
MAY 11, 1976 TUESDAY
MAY 18,⊕

:DATES↵

ENTER YOUR DATE PERIOD
EVERY MONDAY AND THURSDAY EXCEPT THE 3RD MONDAY OF THE MONTH.↵

:DISPLAY↵

MAR 18, 1976 THURSDAY
MAR 22, 1976 MONDAY
MAR 25, 1976 THURSDAY
MAR 29, 1976 MONDAY
APR  1, 1976 THURSDAY
APR  5, 1976 MONDAY
APR  8, 1976 THURSDAY
APR 12, 1976 MONDAY
APR 15, 1976 THURSDAY
APR 22, 1976 THURSDAY
APR 26, 1976 MONDAY
APR 29, 1976 THURSDAY
MAY  3, 1976 MONDAY
MAY  6, 1976 THURSDAY
MAY 10, 1976 MONDAY
MAY 13, 1976 THURSDAY
MAY 20, 1976 THURSDAY
MAY 24, 1976 MONDAY
MAY 27, 197⊕

:DATES↵

ENTER YOUR DATE PERIOD
EVERY WEDNESDAY EXCEPT THE 1ST WED OF THE MONTH AND↵
THE 2ND WED OF THE MONTH AND THE 3RD WED OF THE MONTH AND↵
THE 4TH WED OF THE MONTH.↵
```

Note that this specification defines the fifth calendar Wednesdays. See page 135 for another method of specifying the same dates.

```
:DISPLAY⏎

MAR 31, 1976 WEDNESDAY
JUN 30, 1976 WEDNESDAY
SEP 29, 1976 WEDNESDAY

NO FURTHER DATES

:
```

Summary of PERP Commands

Command	Function
DATES	Accepts the date expression.
INSERT	Accepts job information and enters the job into the queue. The job will be executed according to the expression in the last DATES command.
DISPLAY	Displays all dates associated with the last DATES command expression.
EXPRESS	Prints in PERP language the date specification entered by the user.
TEST	Performs the actions of both EXPRESS and DISPLAY.
STATUS	Prints the next scheduled execution date for a specified job.
DELETE	Deletes a specified job from the job queue.
QUIT	Returns the user to the EXECUTIVE.
HELP	Lists all PERP commands and a description of each.
VERSION	Prints the version number of the current version of PERP.
GET	Retrieves information from the job queue. This information is used in DISPLAY, EXPRESS, and TEST commands to state the date period used.
LIST	Lists all jobs in the periodic processing queue entered under the current user name.

Section 12
THE TELECOPY PROGRAM

TELECOPY is a file transfer program used to transfer files between two TYMSHARE computer systems. It can copy any type of file from one TYMCOM-IX computer to another TYMCOM-IX computer, or it can transfer any symbolic file between a TYMCOM-IX and either a TYMCOM-X or a TYMCOM-370 computer. In all cases file size restrictions are the same as for the EXECUTIVE COPY command. File names must meet the syntax requirements of the receiving system.[1]

To copy files between two systems, the same user name must be valid on the originating system and the destination system. That is, files cannot be telecopied from one user name on one system to a different user name on another system. Files can be copied either from or to the system the user is currently on, but he must be logged in to one of the two systems involved. For example, if the user is logged into system 2, he cannot telecopy from system 24 to 33.

TELECOPY COMMANDS

The TELECOPY program is called by typing

```
–TELECOPY↵
```

in the EXECUTIVE. The program responds with a colon (:) prompt. At this point, the user may enter any valid TELECOPY command. The commands are listed below.

Command	Function
HELP or ?	Lists all TELECOPY commands with a brief description of each command.
CAPABILITIES	Describes program capabilities.
INSTRUCTIONS	Explains how to execute the TELECOPY program.
PDP10	Prints additional instructions relative to file transfers between a TYMCOM-IX and a TYMCOM-X system.
SYS370	Prints additional instructions relative to file transfers between a TYMCOM-IX and a TYMCOM-370 system.

(Table continues)

[1] – See page 23 for a discussion of valid file names on the TYMCOM-IX. Refer to the *Tymshare TYMCOM-X XEXEC Reference Manual* and the *Tymshare TYMCOM-370 CMS Reference Manual* for discussions of valid file names on the respective systems.

Command	Function
ONLC	Enables the program to copy files in both uppercase and lowercase letters.
OFFLC	Enables the program to copy files in uppercase only; resets the ONLC switch. This is the default mode.
TRUNCATE	Sets switch to truncate records beginning with specified column numbers when transferring files from the TYMCOM-370.
NO TRUNCATE	Nullifies the TRUNCATE switch setting. This is the default mode.
SUPPRESS TRAILING BLANKS	Sets switch to suppress trailing blanks when transferring a file from the TYMCOM-370.
NO SUPPRESSION TRAILING BLANKS	Nullifies SUPPRESS TRAILING BLANKS switch. This is the default mode.
VERSION	Lists number and creation date of the current version of the TELECOPY program.
PROJECT	Enters project code. The system asks the user to ENTER PROJECT CODE: and the user responds by entering the appropriate code.
RUN	Begins execution of TELECOPY.
QUIT or Q	Terminates TELECOPY program and returns control to the EXECUTIVE.

Any of the above commands may be abbreviated to their first three letters, except NO TRUNCATE and NO SUPPRESSION TRAILING BLANKS, which may be shortened to the first six letters.

In the following example, the user is logged in to system 2, a TYMCOM-IX system, under the user name DELEON; he telecopies a data file to his own directory on system 40, which is a TYMCOM-370 system. He has to use a different file name on the destination system, as the name the file has on the TYMCOM-IX is not an acceptable name on the TYMCOM-370.

```
-TELECOPY↵

:RUN↵

COPY FROM FILE: (DELEON:2)DATA4↵

COPY TO FILE: (DELEON:40)DATA TEXT↵

COPY STARTING

DATA4 SYSTEM 2
COPIED TO
DATA TEXT SYSTEM 40

107 CHARS. TRANSMITTED

:
```

He copies a file from his directory on system 2 to his directory on system 40, changing the file name so that it is acceptable on the TYMCOM-370.

The program notifies him that the copy is starting.

The program tells him that the copy is complete, tells him how many characters in the file, and returns control to TELECOPY command level.

It is possible to omit prompting if the user chooses. For example, the telecopy run shown above could.have been entered this way:

```
-TELECOPY↵

:(DELEON:2)DATA4,(DELEON:40)MODEL TEXT↵

COPY STARTING

DATA4 SYSTEM 2
COPIED TO
MODEL TEXT SYSTEM 40

107 CHARS. TRANSMITTED

:
```

If the user is telecopying to a TYMCOM-X or to another TYMCOM-IX and wishes to use the same file name in the destination system, he need only give the number of the second system. For example:

```
:(DOMINSKI:2)DATA4,:32↵

COPY STARTING

DATA4 SYSTEM 2
COPIED TO
 SYSTEM 32

20 CHARS. TRANSMITTED

:
```

If a file is being telecopied from the current system, the system number can be omitted.

```
:(DOMINSKI)COMP,:38↵

COPY STARTING

COMP SYSTEM 2
COPIED TO
 SYSTEM 38

6 CHARS. TRANSMITTED

:
```

Several files can be copied at the same time by the following procedure.

```
:(DOMINSKI)ALPH4,ALPH3,ALPH2,:38,BET4,BET3,BET2↵

COPY STARTING

ALPH4 SYSTEM 2
COPIED TO
BET4 SYSTEM 38

6 CHARS. TRANSMITTED

COPY STARTING

ALPH3 SYSTEM 2
COPIED TO
BET3 SYSTEM 38

32 CHARS. TRANSMITTED

COPY STARTING

ALPH2 SYSTEM 2
COPIED TO
BET2 SYSTEM 38

111 CHARS. TRANSMITTED

:
```

The command string can be continued onto additional lines with a line feed and can be up to 300 characters long. The entry is always terminated with a carriage return.

If the TELECOPY is between two TYMCOM-IX's and the file name will be the same on the destination system, the following procedure can be used:

```
:(DOMINSKI:2)RANDOM,HHH1,(DOMINSKI:1)↵

COPY STARTING

RANDOM SYSTEM 2
COPIED TO
 SYSTEM 1

33 CHARS. TRANSMITTED

COPY STARTING

HHH1 SYSTEM 2
COPIED TO
 SYSTEM 1

15 CHARS. TRANSMITTED

:
```

If an attempt is made to copy to an existing file, the TELECOPY program asks for confirmation. For example, the file CONTROL is being copied to system 1 under the file name RANDOM. The system requests confirmation that it should write over an old file. If the user types a Y or YES, the program continues.

```
:(DOMINSKI:2)CONTROL,:1,RANDOM↵

OKAY TO WRITE ON OLD FILE? Y↵

UPDATING STARTED

CONTROL SYSTEM 2
UPDATED TO
RANDOM SYSTEM 1

213 CHARS. TRANSMITTED

:
```

If the user responds with N or NO, the run is aborted.

```
:(DOMINSKI:2)CONTROL,:1,RANDOM↵

OKAY TO WRITE ON OLD FILE? N↵

:
```

The project code that was used to log in to the system will be used by TELECOPY when logging in to the second system unless the user executes the PROJECT option to enter a new project code. This can be done as follows:

```
:PROJECT↵

ENTER PROJECT CODE: 9930↵

:
```

TELECOPY COMMAND FILES

The user can create a command file to execute the TELECOPY program.[1] For example, the user has the following command file called TRANSFER in his directory:

```
TELECOPY
(DOMINSKI:2)TAB1,TAB2,(DOMINSKI:38)TABLE1,TABLE2
QUIT
COMMAND T
```

[1] – See page 45 for a discussion of how to create command files.

The command file runs as follows:

```
-COMMAND TRANSFER↩

COPY STARTING

TAB1 SYSTEM 2
COPIED TO
TABLE1 SYSTEM 38

213 CHARS. TRANSMITTED

COPY STARTING

TAB2 SYSTEM 2
COPIED TO
TABLE2 SYSTEM 38

32 CHARS. TRANSMITTED

-
```

If an attempt is made to copy to an existing file on the receiving system during command file execution, the system prints an error message and aborts the command:

```
-COMMAND TRANSFER↩

OKAY TO WRITE ON OLD FILE?
DATA ERROR IN COMMANDS FILE- TRANSFER ABORTED

-
```

BLANKS, SPACES, AND LINE NUMBERS

Compressed blanks on TYMCOM-IX files are expanded when transferred to a TYMCOM-X or TYMCOM-370. TAB settings, indicating spaces, in TYMCOM-X files are converted to spaces when transferred to the TYMCOM-IX.

TYMCOM-X line-numbered files have the line numbers stripped when transferred to the TYMCOM-IX system.

Blank lines in the TYMCOM-IX file will cause a record of one blank on the TYMCOM-370 file. If the file is returned to the TYMCOM-IX by a subsequent TELECOPY, the SUPPRESS TRAILING BLANKS switch should be set. For example:

```
-TELECOPY↩

:SUPPRESS TRAILING BLANKS↩

:(MJL:42)LATER,:2,LATER9↩
```

METHOD OF FILE OVERWRITING

When the program is copying to an existing file, during a transfer between a TYMCOM-IX and either a TYMCOM-X or a TYMCOM-370 system, the recipient file is opened in a sequential mode. Therefore, the old contents of the existing file are totally replaced by the data in the dispatching file.

When copying to an existing file during transfers between two TYMCOM-IX systems, the old file is opened in an update mode and its contents are merely updated.

Section 13
USING PAPER AND CASSETTE TAPES

The Tymshare system has outstanding features for handling paper tape and cassette tapes. The system has programs to read and record both binary and symbolic tapes and to convert character codes for either input or output.

Symbolic tape is read and recorded with the TAPE program. TAPE has the recording option of including both a title and a terminating control character at the end of the tape. TAPE has the reading option of treating control characters as literals or accepting their editing function.

To store data on either cassette or paper tape, the user can call and execute the TAPE program. Then he gives the file name as the input and T (for terminal) as the output. For example:

```
-TAPE↵                          The user calls the tape program by typing TAPE in the EXECUTIVE.
                                He executes the program.
:RUN↵

INPUT FROM: XXX↵                He specifies the data file to be recorded.

OUTPUT TO: T↵                   T indicates the tape record mechanism.

TITLE: JM↵                      The title shows at the beginning of the tape as identification. If a
                                control shift N (control D on some terminals) concludes the data,
CONTROL-SHIFT-N AT END? Y↵      subsequent reads will terminate at the end of the tape and return
                                control to command level.

TYPE A CARRIAGE RETURN.
THEN TURN ON PUNCH.
↵
```

Recording begins as soon as the recording machine is turned on.

The procedure for reading a tape is very similar except that the input is from T and the output is to a file. In the following example, a tape is read and its contents stored in a file named YYY. Before beginning the program to read a tape, the user should position the tape in the reader. For example:

```
-TAPE↩                      The user calls the tape program by typing TAPE in the EXECUTIVE.

:RUN↩                       He executes the program.

INPUT FROM: T↩              T indicates the tape reader.

OUTPUT TO: YYY↩             The data read from the tape is stored in the file named YYY.
 NEW FILE↩

EDITING? N↩                 Control characters on the tape are treated as literals; the contents of the
                            tape are not edited as the tape is read. The tape is read. Reading
TURN ON READER              terminates when the control shift N at the end of the tape is encountered.


READ COMPLETE
66 CHARACTERS WRITTEN

:
```

Other useful tape programs are BINTAPE, CONFILE, and CONTAPE. BINTAPE punches and reads binary tapes; CONFILE reads a file, converts it to another code, and punches the new code; and CONTAPE reads a tape, converts the code, and writes it on a file. Both CONFILE and CONTAPE require a code conversion table which can be created by the TABLEMAKER program.

All of the tape programs are documented in the *Tymshare TYMCOM-IX Paper Tape Package Reference Manual.*

Appendix A
CONTROL CHARACTERS

The following control characters may be used at the EXECUTIVE level when entering text at the terminal.

Control Character	Symbol Printed	Function
A^c	← or _	Deletes preceding character.
W^c	\	Deletes preceding word.
Q^c	↑	Deletes entire line.
D^c	None	Terminates operation (end of line or file).
V^c	None	Protects a following carriage return, line feed, or control character; i.e., the function of the following character is inhibited, and the character is treated as text.

Multiple controls A, W, or Q can be used to delete any material in the current line, but not in preceding lines. See page 9 for a more complete discussion of editing with control characters.

Appendix B
COMMAND SUMMARY

The following alphabetical list summarizes the TYMCOM-IX EXECUTIVE commands and programs documented in this manual. Page number refers to the page on which complete discussion begins.

Command	Description	Page
CHECKSUM	Calculates and prints checksums for user-specified files.	79
CIPHER	Encodes and decodes files.	35
COMMAND file name	Executes commands contained in the command file.	45
“comments $\left\{\begin{matrix}” \\ D^c\end{matrix}\right\}$	Allows user to include comments in a command file.	48
COMPARE	Compares two files word for word and lists the discrepant lines and their location.	86
CONTINUE	Resumes execution of the program the user was running before transferring directly to the EXECUTIVE.	20
COPY $\left\{\begin{matrix}\text{file name} \\ \text{TERMINAL}\end{matrix}\right\} \left\{\begin{matrix}\text{TO} \\ ,\end{matrix}\right\} \left\{\begin{matrix}\text{TERMINAL} \\ \text{file name}\end{matrix}\right\}$	Copies source to destination.	25
DATE	Gives data and time of day.	51
DECLARE file name(s)	Sets access controls on specified files.	33
DEFER	Places job in deferred processing queue.	105
DEINIT	Deactivates the current initialized file.	49
DELETE file name(s)	Deletes specified files.	29
DIRECTORY [file name(s)]	Lists all characteristics of all or specified files.	39
DIRIT	Lists file names, deletes files, or copies files to another directory.	87
DSC	Prints current storage used and the maximum storage used since the last storage measurement.	52

Command	Description	Page
DUMP file name	Stores machine contents in a file. Contents are recovered with the RUN command.	18
EXIT	Terminates session at the terminal. Inhibits system response of CPU and terminal time.	7
FAILSAFE	If SETFAILSAFE was given, restores memory to state at time of accidental disconnect.	11
FDC	Sets file directory controls.	37
FDM	Crunches and uncrunches symbolic files.	55
FDX	Sets terminal to full duplex mode.	10
FILES [file name(s)]	Lists files and selected information for all files or for specified files.	42
GFD	Gets another user's file directory if it has been declared sharable and if it is in the same account.	38
GO file name	Executes program in GO file.	20
HDX	Sets terminal to half duplex mode.	10
INIT file name	Initializes a GO or a command file to execute automatically after the user logs in.	48
LAST	Prints number of files in directory.	44
LIST	Lists all file names beginning with the one most recently created.	39
LOGOUT	Terminates session at the terminal.	7
OFFESC	Inhibits all interrupts, including alt mode/escape and emergency exit key.	12
OFFLC	Deactivates lowercase.	12
ONESC	Deactivates OFFESC.	12
ONLC	Activates lowercase.	12
PASSWORD	Changes the user's password.	13
PERP	Places job in periodic processing queue.	123
PFDC	Prints file directory controls in effect.	38
PROJECT new project code	Changes project code.	8
QUIT	Returns to EXECUTIVE from a language. Commonly used with REENTER command.	16
RECOVER file name	Restores a DUMP file.	20
REENTER	Returns to the language and work the user was doing before transferring to the EXECUTIVE.	17

Command	Description	Page
REMOVE file number	Deletes file by number.	30
RENAME old file name $\left\{\begin{matrix}\text{AS}\\ ,\end{matrix}\right\}$ new file name	Renames a file.	24
RUN file name	Executes DUMP file; same as RECOVER and then REENTER.	19
SCOMPARE	Compares two symbolic files.	81
SETFAILSAFE	Creates fail-safe file.	11
SUMMARY	Lists total user storage in characters and number of files.	44
SYSNO	Prints system information.	51
TAPE	Calls the paper tape program to read or punch binary and symbolic tapes.	153
TELECOPY	Copies files from one system to another.	145
TIME	Prints time elapsed since log in.	51
TOUT file name	Directs terminal output to a file.	47
TYPE file name	Prints on the terminal the contents of the specified file.	27
VERIF	Calculates and prints checksums for a range of files.	79
WHY	Prints explanation of the preceding error message.	8

INDEX

NOTE: Page numbers that appear in boldface type refer to those pages where the listed items receive the most detailed discussion.