Time Sharing System (TSS) TEX

User's Guide

LARGE SYSTEMS

Time Sharing
TIME SHARING SYSTEM
GCOS 8

REFERENCE
67 A2 DF72 REV03

Time Sharing
TIME SHARING SYSTEM
User's Guide
Time Sharing System (TSS) TEX

GCOS 8

Subject: The TEX Subsystem of the Time Sharing System.

Special instructions: This revision supersedes 67 A2 DF72, REV02 dated January 1994.

Software supported: Large System GCOS 8 Software Release 5.0 and later.

Date: June 1999

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

67 A2 DF72 REV03

Suggestions and criticisms concerning the form, the content, and the presentation of this manual are invited. A form is provided at the end of some Bull manuals for this purpose. Use either this form or the PASS or OSCAR systems to submit comments or changes to this document.

67 A2 DF72 REV03 **File No: 1V13**

Preface

TEX is a multifaceted system that operates in the GCOS time sharing environment. It combines the functions of editing, computing and user resource management info a single language. Language statements may be executed from a terminal or may be stored in a file for execution.

Bull CD-ROM Product

Most of the GCOS 8 documents are available on the Bull CD-ROM product. Contact your marketing representative for more information.

On CD-ROM, some Software Release Bulletins (SRBs) are not identical to the printed versions. The difference exists when a paper SRB includes appendices that contain "change pages" to be inserted into manuals at the customer site. (Change pages are issued when only minimal changes are made to a manual and a re-issuance of the complete manual is not warranted.) The CD-ROM version of that same SRB does not include the change pages. Instead, on the CD-ROM those changes are incorporated directly into an addendum version (e.g., "A", "B") of the affected manual.

Example: Change pages against *XYZ User's Guide* (Order No. AZ18-01), contained in *ABC Software Release Bulletin* (Order No. ZZ99-00) will be included in Order No. AZ18-01A (or later) on CD-ROM.

Problem Analysis Solution System (PASS)

Documentation discrepancies are reported using Software Technical Action Requests (STARs) via either the PASS or OSCAR systems (depending on local account procedures). All documentation corrections are entered on the PASS database and then transferred to the OSCAR database. PASS users should query PASS regularly to determine if corrections have been added to the database. Installations that use OSCAR should check regularly with their local customer support organization to determine if corrections have been added.

Bull Hardware Platform

This document may have generic references to a DPS 9000 hardware platform. If so, such references are applicable to all models of the following Bull large-system computers.

- DPS 9000F — also identified as DPS 9000, Series 900

Model 9000/971	Model 9000/972T
Model 9000/991	Model 9000/992T
Model 9000/993T	Model 9000/994T

- DPS 9000G — also identified as DPS 9000, Series 700

Model 9000/721	Model 9000/722
Model 9000/731	Model 9000/732
Model 9000/741	Model 9000/742
Model 9000/751	Model 9000/752
Model 9000/753	Model 9000/754
Model 9000/755	Model 9000/756
Model 9000/757	Model 9000/758

NOTE: Throughout this document, "DPS 9000G System" refers to the DPS 9000/700 and DPS 9000/700-2 Systems; "DPS 9000G2 System" refers to only the DPS 9000/700-2 Systems.

If information applies only to models of a specific DPS 9000 Series (for example, DPS 9000/900), the documentation so indicates.

Contact your marketing representative for more information about DPS 9000 hardware models.

Table of Contents

Figures

1. Introduction

TEX, a major feature of the GCOS Time Sharing System (TSS) is a programming Language that can be used for many applications not economically programmed in other Languages. TEX borrows heavily from other programming Languages but it differs from them in several respects.

- ALL variables are ASCII strings. Data types are checked when variables are used with monadic or binary operators. Operators that attempt to use strings other than those for which they are intended fail.

- Programs using TEX are not compiled, they are executed one command at a time. It is possible to execute a program either from a file or by typing it one command (or line) at a time at the terminal. (A few commands, e.g., goto, ergoto, cannot be executed from the terminal.)

- The command structure borrows heavily from BASIC, PL/1 and the TEXT EDITOR. The combined ability to do conventional computing as well as extensive string manipulation, all within one language, makes TEX a powerful problem-solving tool.

- Most TSS commands can be invoked from TEX making it a "shell language" for the TSS operating system. TEX makes it possible to change the user environment. This change can restrict the user to a subset of commands or can expand the user's environment.

- TEX can open up to four files and work on them in turn. This makes it possible to extract, convert or merge data from several sources.

- TEX is easily debugged. The program execution halts upon encountering the first error and gives clear diagnostics of the cause of the error. Test runs and program modifications can be alternated rapidly.

New users of TEX will find the expanded functionality a challenge. TEX is best understood if one first learns how to compute. Here, TEX is similar to most conventional languages. The string manipulation commands are novel and need to be read carefully. Next, consider the file manipulation commands (e.g. , old, new, save, resave) are the equivalent of opening and closing of files. Also, consider the text editor commands to be equivalent to calling subroutines that manipulate the text in an opened file. Further, consider system and command file commands as subroutines called by name from your program or terminal. Finally, learn how to create variables from data in a file and how to insert variables in the file. Armed with this understanding, one should be ready to use TEX.

TEX can be used for solving many problems:

- Simple computations such as accumulating and distributing cost
- Maintenance of lists (e.g. , phone books, vendor or customer list, parts lists)
- Extraction information from lists (e.g., list only employees at mail station A53)
- Writing of non-procedural programs (e.g., question and answer sessions between users and the machine that involve little, if any knowledge of the computer)
- Job control programs (e.g., programs that permit non-programmers to run batch or TSS programs)
- Program conversion (e.g., prepare programs written for other systems) for running on a Bull large system.

The following is a summary of typical applications:

General

- Time Sharing
  - Enhancements
  - Customizing
- Prototype Programs
- Non-procedural for non-programmers

Database

- Relational
- Information retrieval
- Update
- Restructuring and co-files
- Data cleanup
- Data compression

Office automation

- Documentation
- Charts
- Automatic Letter writing
- Electronic mail
- Filing

Computer-Aided Learning

- Instruction programs
- Piece parts
- Graded human interface
- Program catalog

Application aids

- Configurators
- Bootload
- JCL generation for batch
- Source code format and maintenance
- Automatic testing of programs

A TEX user should be familiar with GCOS TSS information contained in the *Time Sharing System Reference Manual.*

Notes

2. TEX Subsystem

2.1 Tex Invocation And Done

TEX may be called from the command level or from another subsystem. The TEX command DONE returns from TEX to the calling system. Most subsystems, including TEX, may be called from TEX.

2.2 Character Set

TEX commands and statements contain the alphabetic characters A-Z, a-z, the numeric characters 0-9, and the following special characters:

!	exclamation point	/	slant
#	number sign	:	colon
%	percent	;	semicolon
&	ampersand	<	less than
'	apostrophe	=	equal sign
(	left parenthesis	>	greater than
)	right parenthesis	[	left bracket
*	asterisk	]	right bracket
+	plus sign	^	circumflex
,	comma	_	underline
-	hyphen	|	vertical

Some terminals may not have the complete TEX character set. TEX recognizes the following alternate forms in labels and operators:

Alternate Form	Recognized as
%(	[
%)	]
%+	^
%;	!
%:	|

Example:

```
A=B '] C
is equivalent to
A=B ' %)C
```

Some terminals display a vertical arrow in place of the circumflex. Only the circumflex will be used in this manual.

The text processed by TEX commands and statements is not confined to the ASCII character set. All of the 512 character codes that can be represented by a 9-bit byte are permissible. Text that contains other than ASCII characters can be inspected using the T command described in Section 5. Appendix A shows the ASCII characters and their octal equivalents.

Case

TEX is not sensitive to the case of names, labels, commands, or statements.

Example

```
SCAN:A:B
is equivalent to
scan:A:B
```

TEX commands and statements are sensitive to the case of the text they operate upon unless CASE MODE is in effect (see CASE MODE, Section 6).

2.3 Tex Subsystem Function

The TEX subsystem consists of edition commands, computation commands, and execution commands:

Editing commands

- Build a new text file
- Append to an existing text file
- Edit an existing text file by additions, deletions, or corrections

Computation commands

- Create variables
- Perform arithmetic
- Separate and/or assemble strings
- Perform comparison

Executing commands

- Call a file into execution
- Perform branching during file execution
- Terminate execution of a file

2.4 Command Format

There are three forms of commands:

- Commands followed by a contiguous operand fields
- Commands without an operand field
- Commands followed by space-separated operand fields

All available commands are listed in the paragraphs below. Later sections describe their operation. All commands exceeding four characters may be abbreviated to four characters, except STRIP and STRING.

2.4.1 Commands Followed By a Contiguous Operand Field

Some TEX commands have an optional, contiguous operand field (note the permissible abbreviations).

AFTLIN	DELETE or D	INTC	PRINT or P
AFLIN or A	FIND or F	OUT	REPLACE or R
BACKUP or B	IN	OUTC	SCAN
BEFLIN	INC	OUTT	SPLIT
COPY	INSERT or I	OUTTC	T
CUT	INT	PASTE	

2.4.2 Commands Followed by Space Separated Operand Fields

Some command have an operand field that is not contiguous with the command. Instead they are separated from the operand by one or more spaces. They are:

BREAK	FOUT	INTC	OUTT
CALL	GOTO	LIMIT	OUTTC
CF	IF	MARK	PREFIX
CLEAR	IGNORE	MASK	SHIFT
COLS	IN	OCTL	SUBS
ERCALL	INC	OUT	TRUL
ERGOTO	INT	OUTC	TRUR

2.4.3 Commands Without Operand Fields

The following commands have no operand field.

#NO	NOBREAK	NOOCTL	STOP
#YES	NOCASE	NOSHIFT	STRING
BUILD	NOCTL	NOSUBS	STRIP
CASE	NOFOUT	NOTRACE	TRACE
DONE	NOIGNORE	NOVERIFY	VARIABLES
LINE	NOLIMIT	RETURN	VERIFY
MODES	NOMASK	SEF	WHERE

2.4.4 Command Mode and Build Mode

TEX interacts with the terminal in two modes - command mode and build mode. TEX issues a hyphen to request command mode from the terminal. TEX or other TSS commands may be entered following the hyphen.

Build mode is used to enter text at the terminal. TEX can enter the build mode in response to the BUILD, INSERT, NEW, REPLACE, AFTLIN, and BEFLIN commands. TEX issues the message "enter" followed by an asterisk to request build mode input from the terminal. TEX issues and additional asterisk after each build mode input line is entered. TEX returns to the command mode when a single carriage return is entered in response to the asterisk. On VIP7700 keyboard/display type terminals and similar synchronous terminals, the build mode can be terminated by entering two number signs and a blank character.

Example

```
-new          (command mode)
enter
*aaa          (enter build mode)
*bbb
*             (carriage return)
-             (return to command mode)
```

Service functions recognizable in build mode are #AUTO, #TAPE, #LUCID, #RECOVER and #ROLLBACK. Refer to the *TSS Reference Manual*.

2.4.5 Line Numbered Files

Line numbers are not required by TEX, but line numbered files are required by some of the other time sharing subsystems. To enter line numbered files the user can supply one to eight numeric characters as the line number. These are the first characters of each line.

Line numbers can be supplied automatically by the TSS by the use of the #AUTO command in the build mode. #AUTO can be used as follows:

```
#AUTOMATIC
```

Causes the automatic creation of line numbers by the system. The automatic mode is entered, with line numbers initially starting at 010 and incrementing by 10. When build mode is ended, #AUTO ends too. If build mode is entered again, automatic line numbering does not begin until #AUTO is typed again. It resumes where the previous automatic numbering left off. These line numbers appear in the terminal copy, and are written in the file, just as though the user had typed them.

Example:

```
- build
enter
*#auto
*010 a
*020 s
*030 d
*040 f
*050
*
- build
enter
*#auto           (automatic reentered here)
*050 g
*060 h
*070
*
-b p;*
010 a
020 s
030 d
040 f
050 g
060 h
end of file - request executed 6 times
-
```

```
#AUTOMATIC n,m
```

Causes the automatic creation of line numbers, as above, but starting with line number n and incrementing by m.

```
#AUTOMATIC ,m
#AUTOMATIC n,
```

Causes automatic creation of line numbers beginning at 10 and incrementing by m, or beginning at n and incrementing by 10. (On reentry, the line numbering resumes where it left off.)

Automatic line numbers are normally prefixed to the user's text entry with an intervening blank. To suppress this blank, affix any nonblank, nonnumeric to the #AUTO command. For example,

```
#AUTONB or #AUTOMATICX
```

Entering a carriage return causes exit from the automatic mode but still leaves the user in the build mode.

No commands are recognizable in automatic mode. Use of the character delete (a) or the line delete (CTRL X) affects only the text entered by the user; the full line number is always prefixed to the user-entered text line in the current working file.

An important distinction exists in the treatment of duplicate line numbers under TEX and all other TSS subsystems except Text Editor (which behaves like TEX in this regard). Under TEX, duplicate line numbers are retained where they were entered into the current file. Under other TSS subsystems, the latest line numbered line automatically replaces a line already in the current file with that same line number.

Example:

```
- build
enter
*#auto
*010 first
*020 second
*030         (carriage return stops automatic line
             numbers)
*third       (entered without line numbers, build still in
             effect)
*#auto       (resume line numbering)
*030  fourth
*040
*#auto 10,10 (reset automatic line numbers)
*010 fifth
*020 sixth
*030
*autox       (try autox, no space between line number and
             text)
*0030seventh
*0040
*
```

```
- list

010  first
020  second
third
030          fourth (in TEX build mode, lines that have
             the same
010 fifth    line number do not overwrite.)
020 sixth
0030 seventh
```

2.4.6 Sequencing Line Numbered Files

The BSEQ and SEQUENCE commands can be used from TEX. They are described in the *TSS Reference Manual*.

Example:

```
- new
enter
*#auto
*010  a
*020  b
*030  c
*040  d
*050
*
- i : /020/
enter
*#auto
*050  e
*060  f
*070
*
- b p;*
010  a
020  b
050  e
060  f
030  c
040  d
end of file - request executed 6 times
- sequ
- p;*
10  a
20  b
30  e
40  f
50  c
60  d
end of file - request executed 6 times
```

2.4.7 Entering Text From Paper Tape or Tape Cassette

A text file can be created on tape to be entered into the computer at a later time. To do this, put the terminal in LOCAL, feed enough tape through the tape drive or otherwise ensure that there are no unwanted characters, and type the text. A carriage return, line feed, and two DELETE characters must follow every line of text. An X-OFF (ASCII "DC4") must indicate completion of the text, followed by two delete characters which provides timing.

To enter previously prepared text from tape, type "NEW" or "BUILD" then "#TAPE" following the initial asterisk. When the READY response appears, put the prepared tape in the terminal's tape reader and turn on the tape drive.

Input from the tape is accepted until the terminal operator stops the reader, the tape runs out, or and X-OFF character is read from the tape. On most terminals the contents of the tape is printed as it is read. When tape input is complete, the system looks for an X-OFF prior to transmitting a carriage return and printing an asterisk. At this time, additional text may be entered at the keyboard or a carriage return can be given to obtain the "-" response and allow edition or printing.

```
*tex new
enter
*#tape
Many different codes were used during early years of
computing.
ASCII is the only surviving code of any significance.
(X-OFF)
*(carriage return)
-
```

The #TAPE command can also be used to add text from paper tape to a text file built in a current session at the terminal or previously saved (refer to the SAVE and OLD command descriptions in the *TSS Reference Manual*).

```
-BUILD
enter
*#TAPE
(Read from paper tape as described above.)
(X-OFF)
*(carriage return)
-
```

A printout of the file will show the text from the tape appended to the original text.

The #LUCID request is substituted for #TAPE for non-ASCII tape input. TEX may be used to translate such text if the input records are correctly formatted.

The INSERT command, in conjunction with the #TAPE command, allows the user to insert text from tape in the file at any point in the file. At the selected point (as determined by the operand of the INSERT command), the operator activates the tape reader to read in the tape after the appearance of the ENTER response. Upon termination of tape read, the user gives a carriage return in response to the asterisk and the "-" response appears.

```
- INSERT (appropriate operand)
enter
*#TAPE

(User activates tape reader and
 text is read in from tape.)
(XOFF)
*(carriage return)
```

Text may be inserted alternately from the keyboard and from tape.

```
- INSERT (appropriate operand)
enter
*text entered by user
*more text
*last line of text
*#TAPE
(user activates tape reader and test
 is read in from tape.)
(XOFF)
*Text entered by user
*more text
*last line of text
*(carriage return)
-
```

2.5 Pointer Conventions

TEX always associates a pointer with the current file. This pointer points to the first line of the file until the first editing command is given. The pointer points to an entire line, never a character position within the line. This allows several edit operations to be performed on the same line, as long as the operation does not move the pointer. The line to which the pointer currently points is called the "current line."

The rules governing the movement of the pointer are as follows:

- The PRINT, INSERT, REPLACE, DELETE, FIND, CUT, COPY, and PASTE commands cause the pointer to move forward toward the end of the file, unless the command affects only the current line.

- Following the execution of any of the commands listed above, the pointer points to the last line affected by the command.

- The BUILD command appends text at the end of the current file. Exiting from the build mode repositions the pointer to the beginning of the file.

- The BACKUP command moves the pointer backward to the beginning of the file or a specified number of lines from its former location. Similarly, a form of the FIND command moves line pointer forward a specified number of lines.

- Many commands search for a specified string of characters. These commands start with the current line and search for that string until it is found (or the end of file is reached). The pointer rests at the line in which the string is found (or at the end of file). If the string is found in the current line, the pointer does not move. If the string is not found, the pointer must usually be backed up before and other commands can operate on the current file. The INSERT and PASTE commands may be used without a search argument even though the pointer is at the end-of-file.

If a given line has already been passed by the pointer, the BACKUP command may be used to backup the pointer to the desired location.

The content of the current line can always be determined by using a PRINT command with no operand field. The number of lines from the beginning of the file to inclusively, the current line may be displayed using the "=" command. Unless lines have been deleted or inserted ahead of this line by some intervening commands, the user can reposition to this line using the BACKUP and FIND commands.

The position of the pointer is unpredictable following interruption of a TEX command by the break (interrupt) key.

2.6 Time Sharing System Commands

Time Sharing System (TSS) commands perform service functions (e.g., saving or releasing files, calling in other subsystems) for TEX. At the terminal, commands can be entered immediate after the appearance of the "-" response. TSS commands are described in the *TSS Reference Manual.*

2.7 Multiple Commands On A Command Line

More than one command can be placed on a single line. With only a single command, a "-" response is issued upon command execution and control is returned to the user. With multiple commands on a line, the series of commands are executed before the "-" response is issued and control returned to the user. Commands (and their associated operands, if any) in a multiple command line must be separated from one another by one or more blanks. For example,

```
-F    P;5    B;5    P;3    D;5
```

An unsuccessful command execution in a multiple command line inhibits the execution of any remaining commands. An end-of-file condition does not inhibit the execution of the next command.

Time Sharing System commands described in the *TSS Reference Manual* cannot be followed by another system or TEX command. A TSS command can be the last in a multiple command line.

2.8 Break Handling

The break (interrupt) key on the terminal has the following effect:

- Output to the terminal is interrupted. Any waiting output is discarded.
- If TEX is processing a command and build mode is not in effect, it will be interrupted as soon as possible. If the current command is moving the current line pointer, its position after the break is unpredictable.
- If command mode is in effect, TEX will issue a new hyphen.
- If build mode is in effect, the line being typed is discarded and a new asterisk is issued.
- If TEX is processing an executive file and break mode is not in effect, file execution is stopped and TEX will request a new command line from the terminal.

Break processing under break mode is described in Section 6.

Notes

3. String Variables and Expressions

3.1 Strings

Calculations are performed by TEX using strings. Strings may be one of the types described below.

3.1.1 General String

A general string consists of up to 2048 ASCII characters. No other restrictions are imposed.

3.1.2 Legal Integer

A string is a legal integer if it consists of at least one and not more than 62 digits (0-9). It may have a leading plus or minus sign. It must not contain a decimal point or any other character.

Example:

The following are legal integers:

```
1500
+64
-0
0024
```

3.1.3 Unsigned Integer

An unsigned integer is a legal integer without a leading sign.

Example:

The following are legal unsigned integers:

```
  37
 070
0074
```

3.1.4 Legal Boolean Value

The following are legal boolean values:

```
t
true
f
false
```

These may be either uppercase or lowercase.

Many TEX operators and commands operate on general strings. If an operand must be a legal number, an unsigned number, or a legal boolean value, this will be noted in the description of that operator or command.

When the user constructs TEX commands or assignment statements, strings may be entered directly as delimited strings, or may be referenced by string variable specifiers or star functions.

3.2 Delimited String

A delimited string is an arbitrary line of ASCII characters enclosed by two identical delimiter characters. Clearly, the delimiter character must not appear in the string it delimits. The general form of a delimited string is:

```
IxxxxxxxI
```

where "I" represents a delimiter character and "x" is any ASCII character other than "I".

Delimited strings may be bounded by any ASCII graphic character except A-Z, a-z, 0-9, "*", "_", "+", "-", "^", "(", blank. The use of the excluded characters will produce unpredictable results.

Examples of simple delimited strings:

```
•    /THE COMMITTEE/
•    .WHEN THE VALUE OF (F/X)*N IS O,.
•    %MR.  PERRY, MR.  SMITH, AND MS.  JONES%
```

The delimiter in the first example is the slash, in the second example the period, and in the third example the character %. In succeeding examples, the slash is used as the delimiter unless it conflicts with a slash in the string, as in the second example. The string to be acted on must be contained within one line.

3.3 String Variables

Strings may be saved as the values of string variables. String variables do not require declaration as they might in other languages. They are created by the assignment statement. They are deleted by the "clear" command or by leaving the TEX subsystem via DONE, SYSTEM, BYE, or some other command.

3.3.1 The Assignment Statement

The assignment statement has the following format:

```
NAME = EXPN

Where

"NAME"  is a string variable specifier (defined below)
"EXPN"  is an expression (defined below)
```

The assignment creates a string variable having the name given by the string variable specifier and having the value given by the expression. If the named string variable already exists, it is merely given a new value.

Example:

```
x=1234
```

When this assignment statement is executed, the string variable named "x" is created and given the value "1234".

Example:

```
y=/abcd/
```

When this assignment statement is executed, the string variable named "y" is created and given the value "abcd".

3.3.2 String Variable Specifiers

A string variable specifier is either:

- a string variable name
- an indirect string variable name

3.3.3 String Variable Names

A string variable name must begin with an alphabetic character. The remaining characters may be alphabetic, the digits (0-9), or the underline. The case of the alphabetic characters in the name is ignored when the value of the string variable is fetched. A string variable name cannot be more than 40 characters in length.

Example:

The following can be string variable names:

```
A
XXX
Y95
Z_6__X
```

3.3.4 Indirect String Variable Names

An indirect string variable name is an underline (_) followed by a string variable specifier. The underline causes TEX to perform indirection when creating, modifying, or fetching a string variable. Indirection is performed by evaluating the string variable specifier following the underline. If its value is a legal string variable specifier, it is used in place of the indirect string variable name.

Example:

```
name 5=/data five/
pointer=/name 5/
Y=_pointer
```

"Y" contains "data five" after these statements are executed.

If two or more underlines precede a string variable name, multiple indirections are performed. Additional indirections are performed if an intermediate value fetched as part of an indirection contains leading underlines. The total number of indirections performed is equal to the total number of underlines encountered. The number of indirections is limited to 64 for protection against indirection looping.

Continuing the previous example:

```
A=/_pointer/
X=_A
```

X will contain "data five" after these statements are executed.

3.3.5 VARIABLES Command

"variables" causes TEX to print the names and values of all the current string variables as well as the star functions *in, *l, *m, and *r. Each variable is printed on a new line in the following format:

```
name=/value/
```

If this exceeds 80 characters, continuation lines are printed as needed. The slashes are printed as delimiters even if there are more slashes in the value of the variable.

3.3.6 CLEAR Command

```
CLEAR name
CLEAR *
```

"clear name" causes the named string variable to be cleared. "clear *" causes all string variables to be cleared. When a string variable is cleared it is removed entirely and cannot be used again until it has been created again by an assignment statement.

An attempt to clear a nonexistent string variable is always considered successful. No error message is issued. If the name is not a legal variable name, an error message is issued.

3.4 Star Functions

Star functions may be thought of as implicit, rater than explicit, variables. Because the user cannot assign arbitrary values to star functions, they have the "read-only" property.

The following star function names may be entered in uppercase or lowercase.

*account	contains a subaccount number of the "userid." This number is typed when logging on, is 1 to 12 characters long, and is separated from the "userid" by a semicolon (;).
*at	contains a single "commercial at" symbol (@). *at is sometimes more convenient than octal mode when the @ symbol must be entered from the terminal.
*break	initially contains "f". *BREAK is set to "t" if BREAK mode is in effect and a break is received from the remote terminal. *BREAK is returned to the value "f" when it is referenced or when a BREAK command is executed.
*cl	contains the current line of the current file. If there is no current file, or if TEX is at the end of the file, *cl is null. If *cl is used in a command that moves the pointer, the value of *cl is determined before any change is made in the contents of the current line and its pointer, and it is held fixed until the execution of the command is complete.
*clno	contains a number indicating the position of the current line in the file. TEX counts the lines starting with "1" at the beginning of the file. If the file is line numbered, *clno should not be confused with the "line number" which is at the beginning of each line.
*clvl	contains a number equal to the current "call level." This number is incremented every time a CALL command is executed, and it is decremented when a RETURN command is executed or simulated. It is zero when commands are entered at the terminal. At most, 15 levels of calls are permitted.
*corfil	contains a 40 character string derived from the TSS core file. The string is derived by assigning each successive 9-bit portion of the core file to one character in the *CORFIL string.

*cpu	contains the number of milliseconds of processor time used during the current TSS session.
*date	contains the current date in the form "yy-mm-dd" where yy is the units and tens digits of the year, mm is the month (01 through 12), and dd is the day (01 through 28-31).
*did	contains the deferred id of the current session if deferred processing is in effect. Otherwise *did is null. *did does *not* contain the id of the last deferred session spawned.
*dio	contains the number of disk I/Os issued during the current TSS session.
*eof	contains "f" if there is a current file and TEX is not at end of file, it contains "t" if there is no current file, or if TEX is at end of file.
*fdate	returns an ASCII string in the format yyyy-mm-dd (where yyyy = four-digit year, mm = two-digit month, and dd = two-digit day).
*in	contains the last user response to an IN, INT, INC or INTC command. If a null response was obtained from a file, or if a single carriage return was obtained from a terminal, *in will be null. *in is initially null.
*kin	contains the text provided by the DRL KIN less the trailing carriage return. When *kin is referenced, TEX issues a DRL KIN (see the *TSS Reference Manual*).
*kio	contains the approximate number of characters of terminal input and output processed during the current TSS session.
*l	is the abbreviation of *left. See SCAN and SPLIT commands.
*lcl	contains a number equal to the character length of *cl
*left	See SCAN and SPLIT commands.
*lin	contains a number equal to the character length of *in.
ll	is the abbreviation of *lleft.
*lleft	contains a number equal to the character length of *left.

*lm	is the abbreviation of *lmiddle.
*lmiddle	contains a number equal to the character length of *middle.
*lr	is the abbreviation of *lright.
*lright	contains a number equal to the character length of *right.
*m	is the abbreviation of *middle. See SCAN and SPLIT commands.
*match	See SCAN command.
*middle	See SCAN and SPLIT commands.
*null	contains the null string.
*r	is the abbreviation for *right. See SCAN and SPLIT commands.
*random	contains a digit selected randomly whenever *random is referenced.
*right	See SCAN and SPLIT commands.
*rmdr	contains the remainder of the last division operation.
*snumb	contains the snumb of the last batch job spawned during the current timesharing session. If no job has been spawned, *snumb is null.
*svmd	contains TEX commands that, if executed, would restore the TEX mode as it was at the time the last interfile CALL or GOTO was executed. *svmd contains the following mode commands:

```
BREA     or     NOBR
CASE     or     NOCA
COLS     or     NOCO
IGNO     or     NOIG
LIMI     or     NOLI
MASK     or     NOMA
OCTL     or     NOOC
PREF     or     NOPR
SHIF     or     NOSH
STRING   or     LINE
SUBS     or     NOSU
TRAC     or     NOTR
VERI     or     NOVE
```

*svmd is an executable command if subs mode is on.

Example:

```
SUBS %          (SET SUBS MODE USING "%")

%*svmd%         (set the modes to what they were when the
                 last interfile CALL or GOTO was issued)
```

*sw00 through *sw35	contain codes for switch word bits 0 through 35. If a switch word bit is set, the corresponding *sw value is "t". If a switch word bit is not set, the corresponding *sw value is "f".
*texid	contains a code identifying the TEX version in use. This code is changed each time a TEX update is issued.
*time	contains local 24-hour time in the form "hh:mm:ss".
*times	is updated each time an editing command with a repeat field reaches end-of-file. *times is set equal to the number of times that the editing command was executed before reaching end of file. *times is initially zero.
*userid	contains the userid under which TEX is being used.
*vip	contains "t" if the remote terminal is VIP7700 or similar synchronous device. *VIP contains "f" if the remote terminal is a TTY compatible device.

Star functions also exist for the 32 control characters in the first two columns of ASCII (see Appendix A), the delete character (DEL), and the commercial at (@) character.

*NUL	Null	*DLE	Data Link Escape
*SOH	Start of Heading	*DC1	Device Control 1
*STX	Start of Text	*DC2	Device Control 2
*ETX	End of Text	*DC3	Device Control 3
*EOT	End of Transmission	*DC4	Device Control 4 (Stop)
*ENQ	Enquiry	*NAK	Negative Acknowledge
*ACK	Acknowledge	*SYN	Synchronous Idle
*BEL	Bell	*ETB	End of Transmission Block
*BS	Backspace	*CAN	Cancel
*HT	Horizontal Tab	*EM	End of Medium
*LF	Line Feed	*SUB	Substitute
*VT	Vertical Tab	*ESC	Escape
*FF	Form Feed	*FS	Field Separator
*CR	Carriage Return	*GS	Group Separator
*SO	Shift Out	*RS	Record Separator
*SI	Shift In	*US	Unit Separator
*AT	Commercial AT (@)	*DEL	Delete

3.5 Class Specifier

Class specifiers are not variables. They are used only in SCAN commands and with scan operators. The following are class specifiers:

```
*a    alphabetic     - the 52 ASCII upper case and
                       lowercase alphabetic characters.
*n    numeric        - the digits 0-9.
*an   alphanumeric   - the 52 alphabetics and the 10
                       digits.
*lc   lowercase      - the 26 ASCII lowercase alphabetic
                       characters.
*uc   uppercase      - the 26 ASCII uppercase alphabetic
                       characters.
*c0   control-0      - the 32 ASCII control characters
*c1   control-1      - the 32 extended ASCII control
                       characters with octal values
                       200-237
*g0   graphic-0      - the 94 ASCII graphic characters
*g1   graphic-1      - the 94 extended ASCII characters
                       with octal values 241-376
*t    transparent    - the 32 ASCII control characters
                       and all characters having octal
                       values greater than 176.
```

3.6 Expressions And Expression Elements

Many TEX commands contain expressions or expression elements.

An expression is one of the following:

- An expression element
- Two expressions separated by a binary operator
- An expression element preceded by a unary operator
- An expression followed by a scan operator followed by a class specifier

An e*xpression element* is one of the following:

- A delimited string
- An unsigned number
- A string variable specifier
- A star function
- An expression within parentheses

Example:

```
The following are expressions:

    x+55  (two expressions separated by binary operator)

    ^*eof (expression element preceded by unary operator)

    x'>*n (expression followed by scan operator followed
           by class specifier)
```

Example:

```
The following are expression elements, hence expressions,
too:

    "abc"         (delimited string)
    55            (unsigned number)
    X             (string variable specifier)
    _pointer      (string variable specifier)
    *eof          (star function)
    (x+55)        (expression within parentheses)
```

3.6.1 Order of Precedence

An expression is evaluated left to right wherever operators are of equal precedence. Otherwise evaluation takes place in the following order:

- Any expression within parentheses
- Unary operators
- Scan/ split operators
- Multiplication/ division
- Addition/ subtraction
- Concatenation
- Comparison
- Logical AND
- Logical OR

Intermediate and final results are limited to 2048 characters in length. The maximum number of real and implied nested pairs of parentheses is 16.

Example:

```
3*(1+2)
```

The addition is performed first. The value of this expression is "9". Without the parentheses it would be "5".

Example:

```
14/(4/2)
```

The division "4/2" is performed first. The value of this expression is "7". Without the parentheses the division "14/4" would have been performed first yielding "3". Further division by two would have given the result "1".

3.6.2 A and B Operands

The A and B operands are the strings resulting from the evaluation of the fields to the left and the right, respectively, of each binary operator. The A and B operands can be identified for each binary operator once the order of evaluation of an expression has been established. For brevity, the A and B operands are sometimes referred to simply as "A" and "B".

Example:

```
x+y
```

here x is the A operand, y is the B operand and + is the operator.

Example:

```
(x+y) * z
```

The A operand of the addition is "x". The A operand of the multiplication is the value that results from the evaluation of the expression (x+y).

3.6.3 Normalization

The arithmetic operators normalize their results. A number is said to be normalized when it is in the following form:

- Positive numbers are unsigned with no leading zeros.
- Negative numbers have a leading menus sign and no leading zeros.
- Zero is represented as the single character "O".

Example:

The following are values of the string variable "x" before and after execution of the statement "x=x+0".

```
"x" before          "x" after

   15                   15
 0067                   67
  +24                   24
 +075                   75
  -12                  -12
 -007                   -7
   +0                    0
   -0                    0
 0000                    0
+0000                    0
```

3.7 Operators

3.7.1 Unary Operators

(+) Plus — The value of the expression element following the plus sign (+) must be a legal number. The plus operator causes this value to be normalized.

Example:

```
x = 007
y = +x

The value of "y" will be "7".
```

(-) Minus — The value of the expression element following the minus sign (-) must be a legal number. If it is positive, it is made negative and it is normalized. If it is negative, it is made positive and it is normalized.

Example:

```
x = 007
y = -a

The value of "y" will be "-7".
```

(^) Not — The value of the expression element following the circumflex (^) must be a legal boolean value. If it is true, it is changed to "f". If it is false it is changed to "t".

Example:

```
x = ^*eof
```

When TEX is positioned at end-of-file "x" is set "f". When TEX is not at end-of-file "x" is set "t".

3.7.2 Binary Operators

3.7.2.1 Arithmetic Operators

The A and B operlands of arithmetic operators must be legal numbers.

(+) Addition — The result of A+B is the normalized sum of the A and B operands.

Example:

```
x = 56
y = x+2

The value of y is 58.
```

(-) Subtraction — The result of A-B is the normalized difference when the B operand is subtracted from the A operand.

(*) Multiplication — The result of A*B is the normalized product of the A and B operands.

(/) Division — The result of A/B is the normalized integer portion of the quotient when the A operand is divided by the B operand. The remainder is assigned to the star function *rmdr.

Example:

```
x = 14
y = x/3

The value of "y" is "4".  The value of *rmdr is "2".
```

3.7.2.2 Concatenation Operator

(,) Concatenation — The result of A,B is the concatenation of the A and B operand values.

Example:

```
x = 55
y = "abc"
z = x,y

The value of "z" is "55abc".
```

3.7.2.3 Logical Operators

(|) Logical OR — the result of A/B is the logical inclusive OR as shown in the truth table:

A	B	A/B
t	t	t
t	f	t
f	t	t
f	f	f

Example:

```
if (count : 100| *eof goto !exit
```

The IF statement succeeds if the variable "count" is greater than 100 or if TEX is at end-of-file.

(&) Logical AND — The result of A&B is the logical AND as shown in the truth table:

A	B	A&B
t	t	t
t	f	f
f	t	f
f	f	f

Example:

```
if *match&(count:EQ:3)
```

The IF statement succeeds if *match is true and the variable "count" is equal to 3.

3.7.2.4 Numeric Comparison Operators

Each numeric comparison operator has two forms; either may be used with identical results. The operator may be entered in uppercase or lowercase. The A and B operands of a numeric comparison operator must be legal numbers. An algebraic comparison is performed. Leading zeros are not significant in the comparison. The result is one of the boolean values "t" or "f".

```
:eq:  Equals (numeric)
:eqn:
```

The result of A:eq:B is "t" if the A and B operands are equal. The result is "f" if the A and B operands are not equal.

```
:ge:  Greater than or Equal to (numeric)
:gen:
```

The result of A:ge:B is "t" if the A operand is greater than or equal to B operand. The result is "f" if the A operand is less than the B operand.

```
:gt:  Greater Than (numeric)
:gtn:
```

The result of A:gt:B is "t" if the A operand is greater than the B operand. The result is "f" if the A operand is less than or equal to the B operand.

```
:le:  Less than or Equal to (numeric)
:len:
```

The result of A:le:B is "t" if the A operand is less than or equal to the B operand. The result is "f" if the A operand is greater than the B operand.

```
:lt:  Less Than (numeric)
:ltn:
```

The result of A:lt:B is "t" if the A operand is less than the B operand. The result is "f" if the A operand is greater than or equal to the B operand.

```
:ne:  Not Equal to (numeric)
:nen:
```

The result of A:ne:B is "t" is the A operand is not equal to the B operand. The result is "f" if the A operand is equal to the B operand.

Example:

```
10:ge:5
```

The value of this expression is "t" because "10" is numerically greater than "5".

3.7.2.5 String Comparison Operators

The A and B operands are compared left to right until unequal characters are found. "Greater than" and "less than" conditions are based on the ASCII collating sequence. If two strings are of unequal length, and the shorter string is equal to the leftmost portion of the longer string, the strings are not considered equal; the longer string is considered greater in value. If TEX is in case mode, the uppercase alphabetic characters in each operand treated as if they were lowercase at the time the comparison is made. The result of the string comparison is one of the boolean values "t" of "f".

```
"eqs"  Equals (string)
```

The result of A:eqs:B is "t" if the A and B operands are equal. The result is "f" if the A and B operands are not equal.

```
:ges:  Greater than or Equal to (string)
```

The result of A:ges:B is "t" if the A operand is greater than or equal to the B operand. The result is "f" if the A operand is less than the B operand.

```
:gts:  Greater Than (string)
```

The result of A:gts:B is "t" if the A operand is greater than the B operand. The result is "f" if the A operand is less than or equal to the B operand.

```
:les:  Less than or Equal to (string)
```

The result of A:les:B is "t" if the A operand is less than or equal to the B operand. The result is "f" if the A operand is greater than the B operand.

```
:lts:  Less Than (string)
```

The result of A:lts:B is "t" if the A operand is less than the B operand. The result is "f" if the A operand is greater than or equal to the B operand.

```
:nes:  Not Equal to (string)
```

The result of A:nes:B is "t" if the A operand is not equal to the B operand. The result is "f" if the A operand is equal to the B operand.

Example:

```
"abc":gts:"def"
```

The value of this expression is "f" because "a" is not higher than "d" in the ASCII collating sequence.

Example:

```
10:gts:5
```

The value of this expression is "f" because "1" is not higher than "5" in the ASCII collating sequence.

Example:

```
10:gt:5
```

The value of this expression is "t" because "10" is numerically greater than "5" (see example above).

3.7.2.6 Split Operators

The split operators split the A operand string into two parts. The result is equal to one of these parts. No star functions are created or modified when these operators are evaluated. The B operand must be a legal number. It is used to count A operand characters to find the split position.

The split operators are constructed of these characters:

]	count from the left (think of it as an arrow - - - >).
[	count from the right (think of it as an arrow <- - -).
'	to the left of the "[" or "]" - the result is the leftmost portion of A.
'	to the right of the "[" or "]" - the result is the rightmost portion of A.

These characters are combined to form the following operators:

']	Split from left, save left

The result of A']B is as follows. If B is less than the length of A, the result is the leftmost B characters of A. If B is greater than the length of A, the result is the entire A operand. If B is negative or zero, the result is the null string.

Example:

```
-x="abcde"']3
-out:x
abc
-
```

]'	Split from left, save right

The result of A]'B is as follows. If B is less than the length of A, the result is the leftmost B characters of A are discarded; the result is the remaining portion of A. If B is greater than the length of A, the result is the null string. If B is negative or zero, the result is equal to the A operand.

Example:

```
-x="abcde"']3
-out:x
de
-
```

'[Split from right, save left

The result of A'[B is as follows. If B is less than the length of A, the rightmost B characters of A are discarded; the result is the remaining portion of A. If B is greater than the length of A, the result is the null string. If B is negative or zero, the result is equal to the A operand.

Example:

```
-x="abcde"'[3
-out:x
ab
-
```

[' Split from right, save right

The result of A['B is as follows. If B is less than the length of A, the result is the rightmost B characters of A. If B is greater than the length of A, the result is the entire A operand. If B is negative or zero, the result is the null string.

Example:

```
-x="abcde"['3
-out:x
cde
-
```

3.7.2.7 Scan Operators With A and B Operand

The scan operators scan the A operand string for characters equal to (or not equal to) the B operand. If no apostrophe is used, the result equals the scan count. If an apostrophe is used, the result is a portion of the A operand. No star functions are created or modified when these operators are evaluated. The scan operators are constructed of the following characters:

> scan left to right (think of it as an arrow - - ->).

< scan right to left (think of it as an arrow <- - -).

' to the left of the "<" or ">" - the result is the leftmost portion of A.

' to the right of the "<" or ">" - the result is the rightmost portion of A.

^ scan for not equal.

These characters are combined to form the following operators:

> Scan from left for substring

The result of A>B is as follows. The A operand is scanned left to right for the first occurrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the number of A operand characters to the left of the substring. If the substring is not found, the result is equal to the character length of that A operand. If the B operand is null, the result is equal to the character length of the A operand.

Example:

```
-x="abcdef">"c"
-out:x
2
-
```

>^ Scan from left for not equal

The result of A>^B is as follows. The B operator must be one character. The A operand is scanned left to right for the first character that is not equal to the B operand character. If such a character is found, the result is equal to the number of A operand characters to its left; if it is not, the result is equal to the length of the A operand.

Example:

```
-x="aabc">^"a"
-out:x
2
-
```

< Scan from right for substring

The result of A<B is as follows. The A operand is scanned right to left for the first occurrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the number of A operand characters to the right of the substring. If the substring is not found, the result is equal to the character length of the A operand. If the B operand is null, the result is equal to the character length of the A operand.

Example:

```
-x="abcdef"<"c"
-out:x
3
-
```

<^ Scan from right for not equal

The result of A<^B is as follows. The B operator must be one character. The A operand is scanned right to left for the first character that is not equal to the B operand character. If such a character is found, the result is equal to the number of A operand characters to its right; if it is not, the result is equal to the length of the A operand.

Example:

```
-x="abcdeff"<^"f"
-out:x
2
-
```

'> Scan from left for substring, save left

The result of A'>B is as follows. The A operand is scanned left to right for the first occurrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the portion of the A operand to the left of the substring. If the substring is not found, the result is equal to the entire A operand. If the B operand is null, the result is equal to the A operand.

Example:

```
-x="abcdef"'>"c"
-out:x
ab
-
```

>' Scan from left for substring, save right

The result of A>'B is as follows. The A operand is scanned left to right for the first occurrrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the portion of the A operand to the right to the substring. If the substring is not found, the result is null. If the B operand is null, the result is null.

Example:

```
-x="abcdef">'"c"
-out:x
def
-
```

'>^ Scan from left for not equal, save left

The result of A'>^B is as follows. The B operand must be one character. The A operand is scanned left to right for the first character that is not equal to the B operand character. If such a character is found, the result is equal to the substring of the A operand to its left. If it is not, the result is equal to the A operand.

Example:

```
-x="aabcd"'>^"a"
-out:x
aa
-
```

>'^ Scan from left for not equal, save right

The result of A>'^B is as follows. The B operand must be one character. The A operand is scanned left to right for the first character that is not equal to the B operand character. If such a character is found, the substring to the left is discarded and the result is equal to the remainder. If such a character is not found, the result is null.

Example:

```
-x="abcdef">'^"a"
-out:x
bcde
-
```

'< Scan from right for substring, save left

The result of A'<B is as follows. The A operand is scanned right to left for the first occurrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the portion of the A orerand to the left of the substring. If the substring is not found, the result is null. It the B operand is null, the result is null.

Example:

```
-x="abcdef"'<"a"
-out:x
abc
-
```

<' Scan from right for substring, save right

The result of A<'B is as follows. The A operand is scanned right to left for the first occurrence of a character substring equal to the B operand. If such a substring is found, the result is equal to the portion of the A operand to the right of the substring. If the substring is not found, the result is equal to the entire A operand. If the B operand is null, the result is equal to the A operand.

Example:

```
-x="abcdef"<'"d"
-out:x
ef
-
```

'<^ Scan from right for not equal, save left

The result of A'<^B is as follows. The B operand must be one character. The A operand is scanned right to left for the first character that is not equal to the B operand character. If such a character is found, the substring to the right is discarded and the result is equal to the remainder. If such a character is not found, the result is null.

Example:

```
-x="abcdef"'<^"f"
-out:x
abcde
-
```

<'^ Scan from right for not equal, save right

The result of A<'^B is as follows. The B operand must be one character. The A operand is scanned right to left for the first character that is not equal to the B operand character. If such a character is found, the result is equal to the substring of the A operand to its right. If it is not, the result is equal to the A operand.

Example:

```
-x="abcdef"<'^"f"
-out:x
f
-
```

3.7.2.8 Scan Operators With A Operand and Class Specifier

These operators do not have a B operand. Instead, there is a class specifier immediately to the right of the operator. Characters belonging to this class are referred to here as belonging to "class-B." No star functions are created or modified when these operators are evaluated.

`>` Scan from left for class

The result of A>class-B is as follows. The A operand is scanned left to right for the first character belonging to class-B. The result is equal to the number of characters to the left of this character. If no A operand character belongs to class-B, the result is equal to the number of characters in the A operand.

Example:

```
-x="123abc">*a
-out:x
3
-
```

`>^` Scan from left for not in class

The result of A>^ class-B is as follows. The A operand is scanned left to right for the first character not belonging to class-B. The result is equal to the number of characters to the left of is character. If all A operand characters belong to class-B, the result is equal to the number of characters in the A operand.

Example:

```
-x="abc123">^*a
-out:x
3
-
```

`<` Scan from right for class

The result of A<class-B is as follows. The A operand is scanned right to left for the first character belonging to class-B. The result is equal to the number of characters to the right of this character. If no A operand character belongs to class-B, the result is equal to the number of characters in the A operand.

Example:

```
-x="abc12"<*a
-out:x
2
-
```

<^ Scan from right for not in class

The result of A<^ class-B is as follows. The A operand is scanned right to left for the first character not belonging to class-B. The result is equal to the number of characters to the right of this character. If all A operand characters belong to class-B, the result is equal to the number of characters in the A operand.

Example:

```
-x="123abc"<^*a
-out:x
3
-
```

'> Scan from left for class, save left

The result of A'> class-B is as follows. The A operand is scanned left to right for the first character belonging to class-B. The result is equal to the portion of the A operand to the left of this character. If no A operand character belongs to class-B, the result is equal to the A operand.

Example:

```
-x="123abc"'>*a
-out:x
123
-
```

>' Scan from left for class, save right

The result of A>' class-B is as follows. The A operand is scanned left to right for the first character belonging to class-B. The result is equal to this character and the remaining A operand characters to the right. If no A operand character belongs to class-B, the result is null.

Example:

```
-x="123abc">'*a
-out:x
abc
-
```

'>^ Scan from left for not in class, save left

The result of A'>^ class-B is as follows. The A operand is scanned left to right for the first character not belonging to class-B. The result is equal to the portion of the A operand to the left of this character. If all A operand characters belong to class-B, the result is equal to the A operand.

Example:

```
-x="abc123"'>^*a
-out:x
abc
-
```

>'^ Scan from left for not in class, save right

The result of A>'^ class-B is as follows. The A operand is scanned left to right for the first character belonging to class-B. The result is equal to this character and the remaining portion of the A operand to the right of this character. If all A operand characters belong to class-B, the result is null.

Example:

```
-x="abc123">'^*a
-out:x
123
-
```

'< Scan from right for class, save left

The result of A'<class-B is as follows. The Aoperand is scanned right to left for the first character belonging to class-B. The result is equal to portion of the A operand to the left. If no operand character belongs to class-B, the result is null.

Example:

```
-x="abc123"'<*a
-out:x
abc
-
```

<' Scan from right for class, save right

The result of A<'class-B is as follows. The A operand is scanned right to left for the first character belonging to class-B. The result is equal to the portion of the A operand to the left of this character. If no A operand character belongs to class B, the result is equal to the A operand.

Example:

```
-x="abc123"<'*a
-out:x
123
```

'<^ Scan from right for not in class, save left

The result of A'<^class-B is as follows. The A operand is scanned right to left for the first character not belonging to class-B. The result is equal to this character and the remaining portion of the A operand to the left. If all A operand characters belong to class-B, the result is null.

Example:

```
-x="123abc"'<^*a
-out:x
123
```

<'^ Scan from right for not in class, save right

The result of A<'^class-B is as follows. The A operand is scanned right to left for the first character not belonging to class-B. The result is equal to the portion of the A operand to the right of this character. If all A operand characters belong to class-B the result is equal to the A operand.

Example:

```
-x="123abc"<'^*a
-out:x
abc
-
```

3.8 String Parsing Commands

3.8.1 SCAN Command

The SCAN command has these forms

```
SCAN:A:B
SCANR:A:B
SCANN:A:B
SCANNR:A:B
```

"A" is an expression element. Its value is referred to here as the A operand. "B" may be an expression element or a class specifier. If it is an expression element, its value is referred to here as the B operand.

The SCAN verb assigns values to seven star functions *left, *middle, *right, *lleft, *lmiddle, *lright, and * match. The first six may be abbreviated *l, *m, *r, *ll, *lm, and *lr. The three star functions *left, *middle, and *right are derived from the A operand in a manner described separately for each variation of the SCAN verb. The concatenation of these three star functions following execution of a SCAN verb is equal to the A operand of the SCAN verb. The three star functions *lleft, *lmiddle, and *lright contain numbers equal to the character lengths of *left, *middle, and *right, respectively. *match contains the value "t" or "f" depending on the equality of strings when the scan terminates.

scan (without class specifier)

The A operand is scanned left to right for the first occurrence of the B operand. *middle contains the portion of the A operand found equal to the B operand. *Left and *right contain the portions of the A operand to the left and right of *middle. *match contains "t".

If the B operand is not found, or if the B operand is null, *left contains the A operand, *middle and *right are null, and *match will contain "f".

Example:

```
-scan:"abcde":"c"
-vari
*r=/de/
*m=/c/
*l=/ab/
```

scan (with class specifier)

The A operand is scanned left to right for the first character belonging to the class specified by "B". *right contains this character and the other A operand characters to the right. *left contains the A operand characters to the left of *right. *middle is null. *match contains "t".

If no character in the A operand belongs to the class specified by "B" *left contains the A operand, *middle and *right will be null, and *match contains "f".

Example:

```
-scan:"123abc":*a
-vari
*r=/abc/
*m=//
*l=/123/
```

scanr (without class specifier)

The A operand is scanned right to left for the first occurrence of the B operand string. *middle contains the portion of the A operand found equal to the B operand. *left and *right contain the portions of the A operand to the left and right of *middle. *match contains "t".

If the B operand is not found, or if the B operand is null, *right contains the A operand, *middle and *left are null, and *match contains "f".

Example:

```
-scanr:"abcde":"b"
-vari
*r=/cde/
*m=/b/
*l=/a/
-
```

scanr (with class specifier)

The A operand is scanned right to left for the first character belonging to the class specified by "B". *left contains this character and the other A operand characters to the left. *right contains the A operand characters to the right of *left. *middle is null. *match contains "t".

If no character in the A operand belongs to the class specified by "B" *right contains the A operand, *left and *middle are null, and *match contains "f".

Example:

```
-scanr:"abc123":*a
-vari
*r=/123/
*m=//
*l=/abc/
```

scann (without class specifier)

The B operand must be one character. The A operand is scanned left to right for the first character that is not equal to the B operand character. If such a character is found, *left contains the substring to the left of it, *right contains the remainder, *middle is null and *match contains "f". If such a character is not found, *left contains the A operand, *middle and *right are null and, *match contains "t".

Example:

```
-scann:"aabcde":"a"
-vari
*r=/bcde/
*m=//
*l=/aa/
```

scann (with class specifier)

The A operand is scanned left to right for the first character that does not belong to the class specified by "B". *right contains this character and the other A operand characters to the right. *left contains the A operand characters to the left of *right. *middle is null. *match contains "f".

If all characters in the A operand belong to the class specified by "B", *left contains the A operand, *middle and *right are null, and *match contains "t".

Example:

```
-scann:"abc123":*a
-vari
*r=/123/
*m=//
*l=/abc/
```

scannr (without class specifier)

The B operand must be one character. The A operand is scanned right to left for the first character that is not equal to the B operand character. If such a character is found, *right contains the substring to the right of it, *left contains the remainder, *middle is null and *match contains "f". If such a character is not found, *right contains the A operand, *middle and *left are null and, *match contains "t".

Example:

```
-scannr:"abcdee":"e"
-vari
*r=/ee/
*m=//
*l=/abcd/
```

scannr (with class specifier)

The A operand is scanned right to left for the first character that does not belong to the class specified by "B". *left contains this character and the other A operand characters to the left. *right contains the A operand characters to the right of *left. *middle is null. *match contains "f".

If all characters in the A operand belong to the class specified by "B", *right contains the A operand, *left and *middle are null, and *match contains "t".

Example:

```
-scannr:"123abc":*a
-vari
*r=/abc/
*m=//
*l=/123/
```

3.8.2 SPLIT Command

The SPLIT command has these forms:

```
SPLIT:A:B
SPLITR:A:B
```

"A" and "B" are expression elements. Their values are referred to here as the A operand and B operand. The SPLIT command assigns values to six star functions - *left, *middle, *right, *lleft, *lmiddle, and *lright. These may be abbreviated *l, *m, *r, *ll, *lm, and *lr. The star functions *left and *right are derived from the A operand in a manner described separately for the SPLIT and SPLITR commands. The B operand must be a legal number. *lleft and *lright contain numbers equal to the character lengths of *left and *right, respectively. *middle is always assigned a null value. *lmiddle contains the number "0".

SPLIT

*left contains the leftmost characters of the A operand specified by the B operand number. *right contains the remaining A operand characters. If the B operand number is greater than the character length of the A operand, *left contains the entire A operand, and *right is null. If the B operand number is zero or negative, *left is null, and *right contains the A operand.

Example:

```
-split:"abcde":3
-vari
*r=/de/
*m=//
*l=/abc/
```

SPLITR

*right contains the rightmost characters of the A operand specified by the B operand number. *left contains the remaining A operand characters. If the B operand number is greater than the character length of the A operand, *right contains the entire A operand, and *left is null. If the B operand number is zero or negative, *right is null, and *left contains the A operand.

Example:

```
-splitr:"abcde":3
-vari
*r=/cde/
*m=//
*l=/ab/
```

3.9 Conditional Command

IF Command

IF expn

"expn" is an expression. It must have a legal boolean value. If its value is true, any additional commands on the same line are executed. If the value of the expression is false, the remaining input on this line is ignored. The next input line is fetched normally. Multiple IF commands may appear on a single input line.

Example:

```
if x:gt:100  y=y+1  x=0
```

If the string variable "x" has a value greater than 100, "y" is incremented and "x" is set to zero.

Example:

```
if *eof goto !exit
goto !loop
```

If TEX is at end-of-file, it will transfer control to the label !exit. Otherwise, it will transfer control to the label !loop.

Example:

```
if ^*eof goto !loop
goto !exit
```

This is equivalent to the example above.

3.10 Input-Output Commands

3.10.1 IN Command

```
IN    expn or IN:expn
INC   expn or INC:expn
INT   expn or INT:expn
INTC  expn or INTC:expn
```

"expn" is and expression. The IN and INT commands cause a carriage return, line feed, and the value of the expression to be issued as output. The INC and INTC commands issue the value of the expression above. Normally this output is sent to the terminal and the terminal is put into readiness for data input. The user response is assigned to the star function *in. If command file processing or deferred processing is in effect, these commands perform the following functions:

IN/INC

The output is sent to the output file only (if any). The input is obtained as normal command file input.

INT/INTC

Under deferred processing these commands are illegal and cause aborts. Under command file processing the output is sent to the output file (if any) and to the terminal. The input is obtained from the terminal.

Example:

```
-in /???/
???xxxx
-vari
*in=/xxxx/
```

3.10.2 OUT Command

```
OUT    expn or OUT:expn
OUTC   expn or OUTC:expn
OUTT   expn or OUTT:expn
OUTTC  expn or OUTTC:expn
```

"expn" is an expression. The OUT and OUTT commands cause a carriage return, line feed, and the value of the expression to be issued as output. The OUTC and OUTTC commands issue the value of the expression alone. Normally this output is sent to the terminal. If command file processing or deferred processing is in effect, these commands perform the following functions:

OUT/OUTC

The output is sent to the output file only (if any).

OUTT/OUTTC

The output is sent to the output file (if any). Under command file processing the output is also sent to the terminal.

Example:

```
-out /abc/
abc
-
```

Example:

```
-out /abc/    outc:/def/
abcdef
```

Example:

```
-x=15
-out "request executed",x,"times."
requested executed 15 times.
```

3.10.3 CORFIL Command

CORFIL expn

"expn" is an expression. The CORFIL command causes the value of the expression to be written to the TSS core file in the form of 40 ASCII characters, four per word of the core file. If the value of the expression is more than 40 characters, the leftmost 40 are used. If the value of the expression is less than 40 characters it is left justified and padded to the right with blanks.

3.11 Analytic Commands

3.11.1 "=" Command

An "=" command causes a printout of:

```
Line # nnn
```

where "nnn" is a number indicating the position of the current line in the file. TEX counts the lines starting with "1" at the beginning of the file. If the file is line numbered, "nnn" should not be confused with the "line number" which is at the beginning of each line.

3.11.2 WHERE Command

The WHERE command causes the octal block number and record control word (RCW) offset of the current line to print at the terminal. These numbers are useful when the file is to be examined or corrected through the FDUMP subsystem.

Example:

```
-where
octal block no. 1
octal rcw offset 26
-
```

Notes

4. Executive Files and Commands

4.1 Executive Files

TEX commands and other time sharing commands may be saved in an executive file. In response to the CALL command, TEX reads an executive file and treats each line as if it had been entered at the terminal in direct conversation with TEX. The lines are read and executed from the executive file in sequence. This "command sequence" may be modified by the CALL, GOTO, and RETURN commands.

The executive file is a TSS standard format file. Therefore, it may be created and modified in the same way as text files. It cannot be line numbered. Because the TSS standard file does not permit a line of zero length, the code "*null" has been chosen to represent a null response (carriage return) from the terminal. This code is used to return to command mode after build mode input has been entered.

TEX discards command responses when the commands are from an executive file. The following exceptions cause output to the terminal during file execution:

- The PRINT command
- Editing command execution in verify mode
- File execution in trace mode
- TEX input/output commands
- Execution of CRUN command files containing $*$MARK records or other direct output to the terminal
- The SEF command allowing another command or subsystem to issue output to the terminal

Example:

The file "prog 1" contains the following:

```
backup
find:/xxxx/
delete
return
```

These commands are executed when "call prog1" is entered at the terminal. The line in the current file which begins with "xxxx" will be deleted. The editing commands (backup, find, and delete) are described in Section 5.

The string variable "fields" contains "field1, field2, field3, field4". The file "separate" contains:

```
scan:fields:/,/
f1=*left
scan:*right:/,/
f2=*left
scan:*right:/,/
f3=*left
f4=*right
return
```

When the command "call separate" is entered at the terminal, the following string variables are created:

```
f1 contains "field1"
f2 contains "field2"
f3 contains "field3"
f4 contains "field4"
```

4.2 Branching Commands

4.2.1 Labels

!Label

When an exclamation point (!) appears as the first character on a line, it establishes a label. The label is terminated by a blank character or the end of the line. Labels are used by the CALL and GOTO commands. A label may be followed by commands. If a label is encountered during sequential execution of the file it is ignored.

Labels are made up of the same characters and by the same rules as variable names. Indirection in labels is not permitted.

TEX does not check for duplicate labels. If two or more labels have the same name on the same file, CALL and GOTO commands will generally have unpredictable results.

4.2.2 CALL Command

The CALL command has these forms:

```
CALL   cat/filestring
CALL   !label
CALL   cat/filestring!label
```

The CALL command causes the specified command sequence to be executed. Following its execution, additional commands on the same input line as the call command are executed if they are present. Otherwise the following input line is fetched and executed normally. Execution of the specified command sequence is terminated normally in the following ways:

- A RETURN command is encountered during execution of the specified command sequence.
- A STOP command is encountered during execution of the specified command sequence.
- The file containing the command sequence is exhausted.

Execution of the specified command sequence may be terminated abnormally in the following ways:

- A detectable error is encountered during execution of the command sequence and error recovery mode is not in effect.
- A break signal (interrupt) is received from the terminal.

CALL cat/filestring (interfile call)

The file specified by the cat/filestring must be an executive file. It will be accessed with read permission unless additional permissions are specified in the cat/filestring. The specified command sequence begins with the first line of the file.

Example:

```
The file "cge1" contains the following:

a=1
b=2
c=3
return
```

These commands are executed when "call cge1" is entered at the terminal. The string variables "a", "b", and "c" are created.

CALL !label (intrafile call)

This format may be used only in an executive file. "Label" must match a label in the same file. The specified command sequence begins with the label.

Example:

```
The file "cge2" contains the following:

a=1
call !label1
c=3
return
!label1
b=2
return
```

These commands are executed when "call cge2" is entered at the terminal. The string variable "a" is created first. The CALL command causes control to go to "! label1". The string variable "b" is created next. The RETURN command at the end of the file then causes control to return to the "call ! label1" line. The string variable "c" is created next. The RETURN command on the fourth line causes control to return to the terminal.

CALL cat/filestring ! label (interfile call)

The file specified by the cat/filestring must be an executive file. It is accessed with read permission unless additional permissions are specified in the cat/filestring. "Label" must match a label in the called file. The specified command sequence begins with the label.

Example:

```
The file "cge3" contains the following:

!entry1
x=/xxx/
!entry2
y=/yyy/
return
```

These commands are executed when "call cge3!entry1" is entered at the terminal. The string variables "x" and "y" are created. If, instead, "call cge3!entry2" is entered at the terminal, execution begins at "!entry2" and only the string variable "y" is created.

4.2.3 ERCALL Command

The ERCALL command has these forms:

```
ERCALL  cat/filestring
ERCALL  ! label
ERCALL  cat/filestring!label
```

The ERCALL command cannot be executed from the terminal. It may be used in an executive file. When the ERCALL command is encountered, it sets the error recovery mode. TEX then continues with the execution of the next command. The error recovery mode remains in effect during the execution of subsequent commands until one of the following:

- The command line is exhausted.
- A detectable error occurs.
- A CALL or GOTO command is successfully executed.
- Another ERGOTO or ERCALL is encountered. In this case, the previous ERCALL command is ignored and the new ERCALL or ERGOTO becomes effective.

If a detectable error occurs while the recovery mode is effective, file execution is backed up to the ERCALL command and a simulated CALL command is executed using the cat/filestring and/or ! label following the ERCALL command.

Example:

```
The file "erc1" contains the following:

ercall  !no_x  x=x
return
!no_x
x=*null
return
```

These commands are executed when "call erc1" is entered at the terminal. If the string variable "x" has not been created previously, the statement "x=x" fails. Instead of issuing an error message, TEX simulates a CALL command to the label "! no_x". The variable "x" is created by the statement "x=*null" and control is returned to the ERCALL command. The statement, "x=x" is attempted a second time with success.

4.2.4 GOTO Command

The GOTO command has these forms:

```
GOTO  cat/filestring
GOTO  !label
GOTO  cat/filestring!label
```

The GOTO command may be used only on an executive file. The GOTO command interrupts the current command sequence and continues it at the specified place. The command GO TO will not be recognized.

GOTO cat/filestring (interfile goto)

The file specified by the cat/filestring must be an executive file. It is accessed with read permission unless additional permissions are specified in the cat/filestring. The current command sequence is continued with the first line of that file.

Example:

```
The file "cge4" contains the following:

a=1
goto cge5
b=2

The file "cge5" contains the following:

c=3
d=4
return
```

These commands are executed when "call cge4" is entered at the terminal. The string variable "a" is created first. The GOTO command causes control to go to the file "cge5". The string variables "c" and "d" are created next. Control then returns to the terminal. The statement "b=2" is not executed.

GOTO ! label (interafile goto)

"Label" must match a label elsewhere on the same file. The current command sequence continues at the label.

Example:

```
The file "cge6" contains the following:

a=1
goto !lbl
b=2
!lbl
c=3
return
```

These commands are executed when "call cge6" is entered at the terminal. The string variable "a" is created first. The GOTO command causes control to go to the label "! lbl". The string variable "c" is created next. Control then returns to the terminal. The statements "b=2" is not executed.

GOTO cat/filestring ! label (interafile goto)

The file specified by the cat/filestring must be an executive file. It is accessed with read permission unless additional permissions are specified in the cat/filestring. "Label" must match a label in that file. The current command sequence continues at the label.

Example:

```
The file "cge7" contains the following:

a=1
goto cge8!label1
b=2

The file "cge8" contains the following:

c=3
!label1
d=4
return
```

These commands are executed when "call cge7" is entered at the terminal. The string variable "a" is created first. The GOTO command causes control to go to the label "! label1". The string variable "d" is created next. Control then returns to the terminal. The statements "b=2" and "c=3" are not executed.

4.2.5 ERGOTO Command

The ERGOTO command has these forms:

```
ERGOTO  cat/filestring
ERGOTO  !label
ERGOTO  cat/filestring!label
```

The ERGOTO command cannot be executed from the terminal. It can be used in an executive file. When the ERGOTO command is encountered it sets the error recovery mode. TEX then continues with the execution of the next command. The error recovery mode remains in effect during the execution of subsequent commands until one of the following:

- The command line is exhausted.
- A detectable error occurs.
- A CALL or GOTO command is successfully executed.
- Another ERGOTO or ERCALL is encountered. In this case, the previous ERGOTO command is ignored and the new ERCALL or ERGOGO becomes effective.

If a detectable error occurs while the recovery mode is effective, file execution is backed up to the ERGOTO command and a simulated GOTO command is executed using the cat/filestring and/or !label following the ERGOTO command.

Example:

```
The file "erc2" contains the following:

!get_input
in:"enter number -"
n=*n
ergoto {non_numeric n=n+0
return
{non_numeric
out:*in,"is not numeric"
goto {get_input
```

These commands are executed when "call erc2" is entered at the terminal. If the response to the IN command is not a number, the statement "n=n+0" will fail. Instead of issuing an error message, TEX simulates a GOTO command to the label "! non_numeric". The OUT statement issues a message at the terminal and the request for input is repeated.

4.2.6 RETURN Command

RETURN

The command may be used only in an executive file. The RETURN command terminates the current command sequence. The next commands executed are the commands following the last CALL command executed.

Example:

```
The file "ret1" contains the following:

a=1
call !lbl1 f=6
g=7
return
!lbl1
b=2
call !lbl2 d=4
e=5
return
!lbl2
c=3
return
```

These commands are executed when "call ret1" is entered at the terminal. The string variables "a", "b", "c", "d", "e", "f", and "g" are created in order. Control is then returned to the terminal.

4.3 Service Commands

4.3.1 *NULL Command

*NULL

The NULL command serves as the file equivalent of a no-input carriage return from a terminal.

Example:

```
The file "nul1" contains the following:

f;*i
abc
def
*null
return
```

These commands are executed when "call nul1" is entered at the terminal. The two lines "abc" and "def" are added to the end of the current file.

4.3.2 STOP Command

STOP

File execution is terminated if the STOP command is encountered. If command file processing is in effect, it is terminated. If deferred processing is in effect, it continues with the next available response line on the deferred processing input file.

Example:

```
The file "stop1" contains the following:

 a=1
 call !lbl1
 c=3
 return
 !lbl1
 b=2
 stop
```

These commands are executed when "call stop1" is entered at the terminal. The string variables "a" and "b" are created. The STOP command then terminates file execution and control is returned to the terminal. The statement, "c=3" is not executed.

4.3.3 REMARKS Command

The REMARKS command is made up of one or more underline (_) characters followed by a blank character. When this command is encountered all further processing of the line is stopped and the next line of the executive file is fetched.

Example:

```
The file "rem1" contains the following:

_ remarks example
a=1_create variable "a"
b=2_create variable "b"
return_return to the caller
```

These commands are executed when "call rem1" is entered at the terminal. The string variables "a" and "b" are created.

4.3.4 TRACE and NOTRACE Commands

The TRACE commands causes TEX to enter the trace mode. The NOTRACE command causes TEX to leave the trace mode.

The trace mode has no effect on commands received from a terminal. If commands are received from an executive file following a CALL command, each command and its response are printed at the terminal. Each editing command is executed in verify mode. The result of each assignment statement is also printed.

Output to the terminal under trace mode is buffered. As a result, the program trace printed at the terminal may lag behind actual execution of the file commands. If a break signal (interrupt) is received, the file execution may be stopped at some point beyond the point traced at the terminal.

Example:

```
The file "trc1" contains the following:

x=0
!loop xx=x+1 if x:lt:100 goto !loop
return
```

The loop is intended to be executed 100 times. The variable name "x" has been accidentally typed as "xx", creating an infinite loop. When "call trc1" is entered at the terminal, the program goes into the loop and does not return to the terminal. The error is easily identified by the following dialog at the terminal:

```
-trace
-call trc1
x=0
0
!loop xx=x+1 if x:lt:100 goto !loop
1
!loop xx=x+1 if x:lt:100 goto !loop
1
!loop xx=x+1 if x:lt:100 goto !loop
1
!loop xx=x+1 if x:lt:100 (break entered here)
-notrace
```

4.4 Interaction With Tss And Tss Subsystems

4.4.1 Calls To Other Subsystems

Commands such as OLD, NEW, RUNOFF, and SEQUENCE are processed by other subsystems. Additional site-specific or user-specific commands may be processed by the command loader. When interacting directly with the terminal, TEX passes system commands to the appropriate subsystems and other non-TEX commands to the command loader. Commands invoking other subsystems or the command loader are referred to here as invoking "foreign subsystems."

If a command invoking a foreign subsystem is encountered on an executive file a simulated command file processing (CRUN) environment is automatically entered. Further interaction with the other subsystem occurs under the simulated CRUN environment. When control is returned, TEX file execution is resumed. Usually the use of CRUN to perform this function need not be visible to the TEX user. The following restrictions should be observed:

Input on an executive file that is to be presented to a foreign subsystem must be input recognizable by that subsystem. TEX commands will not be correctly processed. COUT and CPOS commands may yield unexpected results and should be avoided. When a foreign subsystem is called, control must be returned to TEX before the file in execution is exhausted. Subs mode modification of input lines is effective only for the first line passed to the foreign system.

A command may not be followed immediately by an equal sign(=). Commands such as FRN and BRN may be followed by an equal sign if it is separated from the command by a space.

4.4.2 Operation Under Command File Or Deferred Processing

Terminal input and output is, of course, modified if command file (CRUN) or deferred (DRUN) processing is in effect. The TEX commands INT, INTC, OUTT, and OUTC may be used to override normal CRUN operation.

5. Editing Commands

TEX editing commands are used to examine the current file and make changes to it. The basic commands move the line pointer, print portions of the file, and delete, replace, or insert text.

An editing command consists of:

- A verb

and optionally one or all of:

- A verb modifier
- A search field
- An insert field
- A repeat field
- An occurrence field

The verb modifier consists of one or more characters immediately following the verb. These cause the command to operate in a specified mode.

The *search field* and *insert field* specify character strings that are to be searched for or inserted in the current file. The *repeat field* and *occurrence field* specify counts used when operations are performed repetitively.

Some verbs permit only certain modifiers or fields. The search field precedes the insert field, if both fields are present. The occurrence field, if present, precedes the repeat field. Otherwise, these fields may be entered in any order.

Except for the contents of delimited strings (defined below), a command may not contain embedded blank characters.

5.1 Operand Strings

The search, insert, repeat, and occurrence fields contain operand strings that may be:

- A number
- A delimited string
- A variable name
- An expression element
- An asterisk (repeat or occurrence only)

5.2 Operand Field Of The Command

5.2.1 Verb Modifier

Verb modifier characters may be attached to the end of the verb. As many as three may be used in any order. They may be the following:

- "L", "S", "O", "P", or "T"
- The letter "V"
- The letter "B"

"L" (line) is used when TEX is in the string mode. It causes the command to be executed in the line mode. If TEX is already in the line mode, the "L" modifier does not change the operation of the command.

"S" (string) is used when TEX is in the line mode. It causes the command to be executed in the string mode. If TEX is already in the string mode, the "S" modifier does not change the operation of the command.

"O" (occurrence)causes the command to be executed in the occurrence mode.

"V" (verify) causes the command to be executed in the verify mode. A "V" modifier used with a command on an executive file causes the verification to be printed at the terminal.

"B" (before) is used only with the INSERT or PASTE commands. It makes the insert or paste effective before, instead of after, the current line or specified string.

"P" (prefix) is used in PREFIX mode. TEX executes all editing commands without affecting the prefix character and all characters following on that line.

"T" (text) is used in PREFIX mode. TEX executes editing commands without affecting the prefix character and any character preceding it on the same line.

5.2.2 Search Field

The search field consists of a colon (:) followed by one of the following:

- An operand string
- An ellipsis
- An AND function
- An OR function

A command that uses a single operand string as a search field causes TEX to search the current file for a string which matches it. The search begins at the current line. If no matching string is found, the pointer ends up at the end-of-file.

The following conventions apply:

- Consecutive blanks in the file must be matched by the blanks in the search string.
- The carriage return key may not be used inside search strings.

An *ellipsis* is two operand strings separated by a comma.

In *string mode* an ellipsis permits the omission of characters that lie between two bounding strings. It is used when the string to be searched for is long or spans lines.

For example, to find the string

```
][The committee][
```

an ellipsis would permit the following for the same string

```
][The ][, ][ee][
```

instead. The strings identifying the ellipsis must uniquely identify the starting and stopping points in the text being scanned. Operand strings using the ellipsis may not contain a carriage return. An ellipsis can be used to span more than one line.

Example

```
-print;3
Consecutive blanks in the line
must be matched exactly
by the blanks in the search string.
-backup;2
-replaces:/be/,/by/:/match/
-print
must match the blanks in the search string.
```

In *line mode* the ellipsis always defines a span of lines starting at the nearest forward line that begins with the first string and includes the line that begins with the second string.

Example

```
-print;3
Consecutive blanks in the line
must be matched exactly
by the blanks in the search string.
-backup;2
-replaces:/must/,/by/   (replace the second two lines)
enter
*must match the blanks  (insert these two)
*in the search string.  (lines in their place)
*(Carriage return)
-BACKUP;2
-PRINT;3
Consecutive blanks in the line
must match the blanks
in the search string.
```

The commands PRINT, FIND, DELETE, INSERT, REPLACE, CUT, and COPY may be used to search for the presence of more than one string in any one line. The *AND function* searches for a line containing all strings specified in the search string. The *OR function* searches for a line containing one or more of the strings specified in the search field. AND (+) and OR (-) operations must not be intermixed in a search field. Although commands with AND/OR functions search in the string mode the remainder of these commands always operate in the line mode.

The AND and OR functions may consist of up to ten separate operand strings connected by plus signs for the AND form or minus signs for the OR form. The operand strings may be in any order; i.e., the fifth string in order of appearance in the line may be listed first in the search field.

For the AND form, the user lists operand strings separated by plus signs. All of the strings listed must be present to be true. Let "][" represent a delimiter. The AND function is represented as follows:

```
VERB:][STRING1][+...+][STRING6][
```

For the OR form, the user lists operand strings separated by minus signs. Only one of the listed strings need be present to be true. Let][represent a delimiter. The OR function is represented as follows:

```
VERB:][STRING1][-...-][STRING6][
```

The AND and OR forms of the operand are equivalent in line or string mode, but the string modifier cannot be used with the verb.

Example

```
-list

Why do people use so many long words in writing?  I've
collected a list of the most common and I've found that
there are many reasons.  Some pompous words are used
because the writer wants to cloak himself in a mantle
of false dignity.  Some are used for hedging.  Some are
used as a status symbol.

-b
-p:/collect/+/list/          (Print the line
                             that contains both
                             "collect" and "list".)

writing? I've collected      (Pointer is now at
a list of the most           line.)

-findv:/word/-/cloak/        (Find and verify a
-/long/                      line containing one of
                             the three words.)

reasons.  Some pompous       (This is the first
words are used because       line found.)
```

5.2.3 Insert Field

The insert field consists of a colon (:) followed by an operand string.

When used with the INSERT or REPLACE verbs, the insert field is always preceded by a search field. When used with the BEFLIN or AFTLIN verbs, it is the only field beginning with a colon (:). When the insert field is not present in the command, TEX will respond

```
enter
*
```

and enter the build mode.

5.2.4 Repeat Field

The repeat field consists of a semicolon (;) followed by an operand string.

The repeat field specifies the number of times an operation is to be repeated. Its value must be a number or it must be an asterisk. A number states the exact number of repetitions wanted; the asterisk (*) causes the operation to be repeated from the current line to the end of file. When the repeat field is not included, the operation is performed only once. The BACKUP verb is an exception: If a repeat field is not present, it sets the pointer to the first line. If the repeat field contains an asterisk it also sets the pointer to the first line.

When the repeat field is used without a search field, the operation is always performed in the line mode.

The effect of the repeat field is explained in the detailed descriptions of each command (see "Editing Commands" below). A few brief descriptions are given below.

```
PRINT;5 (Prints five lines, beginning at the location
of the pointer.  The pointer is left at the last line
printed.)

PRINT:/YOU/;3 (Prints the first three lines that begin
with the characters YOU.  This would include YOUR,
YOURS, YOU'RE, etc.  The pointer is left at the last
line printed.)
```

```
PRINT: /YOU/;3 (Prints the lines containing the first
three occurrences of the string YOU.  This results in
three lines or less of print, possibly only one if all
three occurrences are in the same line.  The pointer is
left at the last line printed.  If the search field is
found more than once in the same line, that line is
only printed once, but the count of occurrences is kept
intact.  If there are less than three YOU strings, the
pointer will be at the end-of-file and the count will
show how may times it was found.)

PRINT;* (Prints the complete file beginning with the
current line.  The pointer is left at the end-of-file.)
```

5.2.5 Occurrence Field

The occurrence field consists of a semicolon (;) followed by an operand string.

The occurrence field may only be used with the "O" verb modifier. Its value may only be a number. It must always precede the repeat field, if any. If no repeat field is present, the operation is performed once. Commands with the "O" modifier must always have a search field and they always search in string mode. These commands search for the "nth occurrence" of the string before operating on it.

Example

```
-p
AAA-BBB-CCC-DDD
-RVO:/-/:/+/;3
AAA-BBB-CCC+DDD
-
```

5.3 Editing Commands

TEX commands are described below in the following order (note the permissible abbreviations):

```
BUILD
BACKUP or B
PRINT or P
FIND or F
DELETE or D
INSERT or I
REPLACE or R
COPY
CUT
PASTE
AFTLIN or A
BEFLIN
T
MARK
TRUL
TRUR
```

5.3.1 BUILD Command

The BUILD command is used to append additional text to the current file. When receiving the BUILD command, the TEX subsystem responds with a line that prints "enter", followed by a line containing a prompting asterisk.

The text to be entered is typed immediately following the asterisk. After each carriage return, an asterisk is issued and TEX is ready for another line of text input. If a line consisting of only a carriage return is typed, text entry is ended, the "-" response is given, and the pointer is returned to the beginning of the file.

```
-print;*

Time-sharing permits a dialogue between the computer
and user, permitting the dialogue to begin immediately,
without waiting for the computer to complete previous
programs.  Data is fed from the terminal directly to
the computer and answers are received quickly at the
same terminal.

end of file - request executed 7 times
BUILD
enter
*The program can be corrected or changed by
*the user as if he were conversing by phone,
*except in this case, the conversation is
*typed or displayed, depending upon the type
*of terminal in use.
*(blank, carriage return)
*If the program contains a mistake, the
*computer informs the user.
*(carriage return)
-print;*

Time-sharing permits a dialogue between the computer
and user, permitting the dialogue to begin immediately,
without waiting for the computer to complete previous
programs.  Data is fed from the terminal directly to
the computer and answers are received quickly at the
same terminal.  The program can be corrected or changed
by the user as if he were conversing by phone, except
in this case, the conversation is typed or displayed,
depending upon the type of terminal in use.

If the program contains a mistake, the computer informs
the user.

end of file - request executed 15 times
```

5.3.2 BACKUP Command

The BACKUP (B) command moves the pointer backward the number of lines specified in the repeat field. If the repeat field is not present, the pointer is backed to the beginning of the file. The formats and execution are as follows:

Command	**Execution**
BACKUP	Backup pointer to beginning of file.
BACKUP;n	Backup pointer n consecutive lines.

5.3.3 PRINT Command

The PRINT Command

The PRINT (P) command is used when either a selected portion of a file or the entire file is to be printed. The user can vary the PRINT command to print any one of the following:

- A single line
- The entire file
- Any number of consecutive lines
- Any number of lines containing a given character string or strings
- From a line containing one given character string through a line containing another given character string (ellipsis).

The formats and execution are as follows:

Command	**Execution**
PRINT or P	Print the current line.
PRINT;n or P;n	Print n consecutive lines starting with the current line.
PRINT;* or P;*	Print entire file starting with the current line.
PRINT: /xxx/ or P:/xxx/	Print the next line which begins with xxx.
PRINT:/xxx/; n or P: /xxx/; n	Print the next *n* lines which begin with xxx. (An asterisk can be used instead of *n* to print all such lines.)
PRINT:/xxx/, /yyy/ or P: /xxx/, /yyy/	Print the block of lines starting with the line which begins with xxx through the line which begins with yyy. (A repeat field can be used to print n or all such blocks of lines.)
PRINTS: /yyy/ or PS: /yyy/	Print line containing specified string.

PRINTS: /yyy/ ; n or PS: /yyy/ ;n	Find the string yyy n times and print the lines containing these strings. (The asterisk can be used to print all lines containing the specified string.) The line is printed only once if the string occurs more than once on a line. If the end of file is reached, the execution count will reflect the number of strings found, not the number of lines printed.
PRINTS:/yyy/, /zzz/ or PS: /yyy/,/zzz/	Print from line containing string yyy to line containing string zzz, inclusive. (A repeat field can be used with this form also.
PRINT:/xxx/+/yyy/+... or P: /xxx/+/yyy/+...	Print line containing all of the specified (a maximum of ten) strings. (A repeat field can be used to print n or all such lines.)
PRINT: /xxx/-/yyy/-... or P: /xxx/-/yyy/-...	Print line containing one or more of the specified (a maximum or ten) strings. (A repeat field can be used to print n or all such lines.)
PRINTO: /yyy/; m;n or PO: /yyy/; m;n	Print the line containing every mth occurrence of the string yyy until this occurrence has been found n times.

To print the complete file, use the PRINT command in line mode with the asterisk in the repeat field. Printing begins at the location of the pointer and continues to the end of the file.

```
-PRINT;*
PROGRAMMING LANGUAGES

Human languages are impractical for preparing computer
programs because these languages contain many
ambiguities and redundancies.  The computer interprets
language absolutely literally.  By the same token,
machine languages are also impractical because they are
difficult for people to use.  Most programming
languages are compromises between human and machine
languages.
end of file - request executed 11 times
```

To print a single line, use the PRINT command with or without a string field. If no search field is specified, the current line (where the pointer is located) is printed.

```
-backup
-print
PROGRAMMING LANGUAGES
-
```

When a search field is specified, the first line that begins with the specified string is printed. The search starts at the current line. If the search string is matched in the current line, it is printed. The pointer is moved to the line that is printed.

```
-BACKUP
PRINT:/Human/
Human Languages are impractical for preparing
-
```

To print any number of consecutive lines, use PRINT in the line mode with a repeat field. Printing begins at the location of the pointer.

```
-BACKUP
-PRINT;3
PROGRAMMING LANGUAGES
                      (Line space inserted during build)
Human languages are impractical for preparing
-
```

To print a line containing a specified string, use PRINTS with a search field, with or without a repeat field. The underline in the following example is used to help the reader identify matching strings; they do not represent underlined strings.

```
-prints:/shar/
time-sharing system
-prints:/shar/;4
time-sharing system
the time-sharing system uses a technique by
thus, time-sharing permits a user to work
many others at the same time share this
-
```

To print the lines specified by an ellipsis, use PRINTS and two search fields separated by a comma.

```
-prints:/time/,/use./
Time-sharing permits a dialogue between the computer
and user, permitting the dialogue to begin immediately,
without waiting for the computer to complete previous
programs.  Data is fed from the terminal directly to
the computer and answers are received quickly at the
same terminal.
```

```
The program can be corrected or changed by the user as
if he were conversing by phone, except in this case,
the conversation is typed or displayed, depending upon
the type of terminal in use.
-
```

The first search field must contain data unique to the first line to be printed (not necessarily in the beginning of the line) and the second search field must be unique to the last line to be printed. In the last example, if the second search field did not contain the period after USE, only the two lines of text, through the line containing the word USER, would have been printed.

5.3.4 FIND Command

The FIND command moves the pointer through the file. FIND may be used with or without an operand field.

If in doubt as to where the pointer is currently pointed, enter the PRINT command with no operand field and the current line will be printed out. When editing a file in which the specified string may appear more than once, it is advisable to print the current line before changing the file. This ensures that the change will be made in the right place.

The repeat field can be used with a search field in the FIND command. The search and comparison continues until the comparison is made as many times as indicated. When execution is completed, the "-" response appears. If the repeat field is used without a search field, the pointer moves forward n lines as indicated by the repeat field. The formats and execution are as follows:

Command	**Execution**
FIND or F	Advance pointer one line.
FIND; n or F; n	Advance pointer n lines. (An * instead of n will move the pointer to the end of the file.)
FIND: /xxx/ or F: /xxx/	Find line which begins with xxx.
FIND: /xxx/ ;n or F: /xxx/; n	Find nth line which begins with xxx.
FINDS: /yyy/ or FS: /yyy/	Find line containing specified string.
FINDS: /yyy/ ; n or FS: /yyy/ ; n	Find the line containing the nth occurrence of the specified string.
FIND: /xxx/+/yyy/+... or F: /xxx/+/yyy/+...	Find line containing all of the specified strings. (A repeat field can be used to find n or all such lines.)
FIND: /xxx/-/yyy/-... or F: /xxx/-/yyy/-...	Find line containing all of the specified strings. (A repeat field can be used to find n or all such lines.)
FINDO: /xxx/; m or FO: /xxx/;m	The FINDO command acts the same as the FINDS command.

To find a specified string not at the beginning of the line, use FIND in the string mode. (The underline in the following examples is used to help the reader identify the matching strings. They do not represent underlined words within the text.)

```
-FINDS:/MUST/
-PRINT
     The program must be submitted to the
-BACKUP;4
-PRINT;*
A program must meet two primary requirements (the
pointer was here when before it can be run (have all
instructions the find command was given.) executed) on
a computer.

   The program must be submitted to the computer in a
   language that the computer recognizes.
end of file - request executed 7 times
```

To find a string past the nearest location point where it next occurs, use FIND in the string mode with a repeat field.

```
-PRINT;6
COMPUTER PROGRAMS

A computer program is a set of instructions that tells
a computer how to accomplish a specific task.  Each
instruction is performed in the sequence specified by
the program.  In this way
-B
-FINDS:/is/;3
-PRINT
task.  Each instructions is performed in the
```

To move the pointer ahead by a specified number of lines, use FIND in line mode with a repeat field. The pointer will move forward the number of lines specified in the repeat field. If FIND is used without a search field, the pointer moves forward one line, same as FIND;1.

```
-PRINT;4
The time-sharing systems uses a technique by which
programs are handled in parallel.  A supervisory
program acts as a controller of these programs,
controlling "STOP" and "GO"
-FIND;1
-PRINT
signals to inputs from terminals and
```

5.3.5 DELETE Command

The DELETE command allows the deletion of any number of characters, words, or lines from the current file. The search field of the command specifies the text to be deleted. If no search field is given, the line where the pointer is located is deleted. After a line or lines are deleted, the pointer is located at the first line that was not deleted. If the last deletion is only a portion of a line, the pointer is located at that line.

The formats and execution are as follows:

Command	**Execution**
DELETE or DELETE;n or D or D;n	Delete line or lines at which pointer is currently located.
DELETE: /xxx/	Delete the first line beginning with the string xxx.
DELETE: /xxx/; n or D: /xxx/; n	Delete the next n lines beginning with the string xxx. (*can be used instead of n to delete all such lines.)
DELETE:/xxx/,/yyy/ or D:/xxx/, /yyy/	Delete the block of lines starting with the line beginning with the string xxx through the line beginning with the string yyy. (A repeat field can be used to delete n or all such blocks of lines.)
DELETES: /yyy/ or DS: /yyy/	Delete specified string.
DELETES: /yyy/; n or DS: /yyy/; n	Delete n occurrences of specified string. (*can be used instead of n to delete all such occurrences.)
DELETES: /yyy/, /zzz/ or DS: /yyy/, /zzz/	Delete all text between the first occurrences of strings yyy and zzz, inclusive. (A repeat field can be used with this form also.)
DELETE: /st1/ + . . . /stn/ or D: /st1/+ . . . /stn/	Delete the first line containing all of the specified (a maximum of ten) strings. (A repeat field can be used to delete n or all such lines.)
DELETE: /st1/ - . . . /stn/ or D: /st1/ - . . . /stn/	Delete line containing one or more of the specified (a maximum of ten) strings. (A repeat field can be used to delete n or all such lines.)
DELETEO: /xxx/;m;n or DO:/xxx/;m;n	Delete the mth occurrence of string "xxx" n times.

To delete a string of characters, use DELETE in the string mode, with or without a repeat field.

```
-PRINT
(have all instructions executed 0) on a computer.
-DELETES:/0/
-PRINT
(have all instructions executed) on a computer.
-
```

To delete a string specified only by its beginning and ending characters use delete in the string mode with an ellipsis using two search strings, with or without a repeat field. All text between and including the two strings indicated is deleted. Short verb forms are used in this and the following examples.

```
-PRINT;4
computer programs because these languages contain many
ambiguities and redundancies; the computer interprets
language absolutely literally.  By the same token,
machine
-B
-DS:/the c/,/. /
-B
-P;4
computer programs because these languages contain many
ambiguities and redundancies; By the same token,
machine languages are also impractical because they are
-
```

To delete one or more lines, use DELETE in line mode, with or without a search field and/or repeat field. If both a search field and repeat field are used, the indicated number of lines beginning with the specified string are deleted. If no search field is used with the repeat field, the indicated number of lines is deleted, beginning at the location of the pointer.

```
-PRINT;4
Human Languages are impractical for preparing computer
programs because these languages contain many
ambiguities and redundancies; the computer interprets
language absolutely
-B;3
-D;3
-PRINT;2
the computer interprets language absolutely literally.
By the same token, machine
-
```

To delete all lines having a common beginning, use DELETE in line mode with a search field and a repeat field. In the following example, the line containing the string "all language instruction must be" is not deleted because the letter A of the text is preceded by blanks.

```
-PRINT;*
COMPUTER PROGRAMS

A computer program is a set of instructions that tells
a computer how to accomplish a specific task.  Each
instruction is performed in the sequence specified by
the program.  In this way the computer processes and
produces information as directed by the program.

A program must meet two primary requirements before it
can be run (have all instructions executed) on a
computer.

       The program must be submitted to the computer in
       a language that the computer recognizes.

       All language instruction must be complete and be
       precisely stated.
end of file - request executed 19 times
-B
-DELETE:/A/;*
end of file - request executed 2 times
-B
-P;*
COMPUTER PROGRAMS

tells a computer how to accomplish a specific task.
Each instruction is performed in the sequence specified
by the program.  In this way, the computer processes
and produces information as directed by the program.

Before it can be run (have all instructions executed)
on a computer.

       The program must be submitted to the computer in
       a language that the computer recognizes.

       All language instruction must be complete and be
       precisely stated.
end of file - request executed 17 times
```

5.3.6 INSERT Command

The INSERT command allows the insertion of any number of characters, words or lines into the current file. The search field of the INSERT command specifies the point after which the insertion is to be made. The search field of INSERTB (or IB) command specifies the point before which the insertion is to be made.

- The string to be inserted can be entered two different ways:
- to insert any string that contains "carriage returns".
- when the string to be inserted contains no "carriage return" and is short enough to fit on a line where it is appended to the INSERT command as an insert field.

The following list illustrates the format to be used when the insert field is not appended to the command on one line. The system responds to the INSERT command with the word "enter" followed by a line that contains an asterisk for a prompting character. The text to be inserted is then typed on a line or several lines, each of which will be prompted by the asterisk. When text entry is complete, a carriage retrurn following the asterisk generates the "-" response. The formats and execution are as follows:

Command	**Execution**
INSERT or I	Insert after the line at which the pointer is currently located.
INSERT: /xxx/ or I: /xxx/	Insert after the line starting with xxx.
INSERT: /xxx/ ; n or I: /xxx/ ; n	Insert after each of the next n lines that start with xxx. (* can be used instead of n to insert after all such lines.)
INSERT: /xxx/ , /yyy/ or I: /xxx/ , /yyy/	Insert a line after the ellipsis starting with a line beginning with the string xxx and ending with the line beginning with the string yyy.
INSERTS:/yyy/ or IS: /yyy/	Insert after the string yyy.
INSERTS : /yyy/; n or IS: /yyy/ ; n	Insert after each of n successive strings yyy. (* can be used instead of n to insert after all such occurrences.)
INSERTS: /yyy/, /zzz/ or IS: /yyy/, /zzz/	Insert after ellipsis starting with yyy and ending with zzz. (A repeat field can be used with this form.)

INSERT: /st1/+ . . . /stn/ or I: /st1/+ . . . /stn/	Insert after line containing all of the specified (a maximum of ten) fields. (A repeat field can be used to insert after n or all such lines.)
INSERT: /st1/- . . . /stn/ or I: /st1/- . . . /stn/	Insert after line containing one or more of the specified (a maximum of ten) strings. (A repeat field can be used to insert after n or all such lines.)

NOTE: The suffix B modifies the INSERT command to INSERT BEFORE.

The next list illustrates the use of INSERT with short strings that do not span lines. When inserting short fields of text, and insert field is added to the command. Here, the command and the entire operand field must be on the same line. This format does not accept a carriage return before the final delimiter. No "enter: response occurs because the command form is complete without further text entry.

Command	**Execution**
INSERT: /xxx/ : /bbb/ or I : /xxx/ : /bbb/	Put a line consisting of bbb after the line which begins with the string xxx.
INSERT: /xxx/ : /bbb/ ; n or I : /xxx/ : /bbb/ : n	Put a line consisting of bbb after each of the next n lines which begin with the string xxx. (* can be used instead of n to insert after all such lines.)
INSERTS : /yyy/ : /bbb/ or IS : /yyy/ : /bbb/	Put string bbb after string yyy.
INSERTS : /yyy/ : /bbb/; n or IS : /yyy/ : /bbb/; n	Put string bbb after each of n successive occurrences of string yyy. (* can be used instead of n to insert after all such occurrences.)
INSERTS: /yyy/, /zzz/ : /bbb/ or IS : /yyy/, /zzz/ : /bbb/	Put insert string bbb after the first occurrence of the ellipsis defined by strings yyy and zzz. (A repeat field can be used with this form.)
INSERTO : /xxx/ ; m : /yyy/ or IO : /xxx/ ; m : /yyy/	Insert string yyy after the mth occurrence of xxx (repeat field permitted, search fields may have the same form as INSERTS).

NOTE: The suffix B modified the INSERT command to INSERT BEFORE.

To insert one or more lines, use INSERT in the line mode with or without a search field and/or repeat field. If no search field is used, the insertion is made after the line where the pointer is located. More than one line may be inserted following the ENTER response.

```
-PRINT;5
quickly at the same terminal.  The program can be
corrected or changed by the user as if he were
conversing by phone, except in this case, the
conversation is typed or displayed.
-B;4
-INSERT
enter
*If the program contains a mistake, the
*computer informs the user.
*(carriage return)
-B;3
-PRINT;*
quickly at the same terminal.  If the program contains
a mistake, the computer informs the user.  The program
can be corrected or changed by the user as if he were
conversing by phone, except in this case, the
conversation is typed or displayed.
end of file - request executed 7 times
```

Example

To insert a string of characters, use INSERT in the string mode with a search field, with or without an insert field. The search field must identify the point after which the insertion is to be made.

```
-PRINT;4
The time-sharing system uses a technique by which
programs are handled in parallel.  Thus, time-sharing
permits a user to work directly with the computer,
whether it is
-B
-INSERTS:/lel./
enter
* A
*Supervisory program acts as a controller of
*these programs, controlling "STOP" and "GO"
*signals to inputs from terminals and
*preventing demands of one terminal from
*interfering with demands of other terminals.
*(carriage return)
-B
-PRINT;9
```

```
The time-sharing system uses a technique by which
programs are handled in parallel.  A supervisory
program acts as a controller of these programs,
controlling "STOP" and "GO" signals to inputs from
terminals and preventing demands of one terminals and
preventing demands of one terminal from interfering
with demands of other terminal.  Thus, time-sharing
permits a user to work directly with the computer,
whether it is
-P:/the program/
The program be corrected or changed by
-INSERTS:/ram/:/can/
-P
The program can be corrected or changed by
```

The INSERT command, as indicated in the descriptions of the command above, provides for insertion of text following the specified line or string. An optional verb modifier, the letter B, can be used with the INSERT command to achieve insertion before the specified line or string.

```
-STRING
-P
The program can be corrected or changed by
-INSERTB:/The/:/therefore,/
-P
therefore, The program can be corrected or changed by
-
```

To insert at the beginning of the file, back up the pointer to the first line and use INSERTB in the line mode with no search field.

```
-PRINT;3

A program must meet two primary requirements before it
can be run (have all instructions executed) on a
computer.
-LINE
-B
-INSERTB
enter
*COMPUTER PROGRAMS
*(blank, carriage return)
*A computer program is a set of instructions that
*tells a computer how to accomplish a specific
*task.  Each instruction is performed in the
*sequence specified by the program.  In this way,
*the computer processes and produces information
*as directed by the program.
*(carriage return)
-B
-PRINT;11
COMPUTER PROGRAMS
```

A computer program is a set of instructions that tells a computer how to accomplish a specific task. Each instruction is performed in the sequence specified by the program. In this way, the computer processes and produces information as directed by the program. A program must meet two primary requirements before it can be run (have all instruction executed) on a computer.

5.3.7 REPLACE Command

The REPLACE command allows the user to replace any number of characters, words, or lines of text with new text of any length. REPLACE may or may not have a search field. If no search field is given, the line where the pointer is located is replaced.

The operand field can take one of two forms, depending on the length of the replacement text. The first list illustrates the format to be used when the insert field cannot be contained in one line. The second list illustrates the use of REPLACE with short strings.

Following the REPLACE commands below, the system responds with "enter" followed by line containing a prompting asterisk. The replacement text is then typed in. Replacement text is typed line by line following each asterisk prompt. When text entry is complete, a carriage return in response to the asterisk generates the "-" response. In string mode, the RETURN key entered on the last line of text is ignored; that is, it does not cause a line break at that point in the text.

Command	**Execution**
REPLACE or R	Replace the current line. (A repeat field can be used with this form. It replaces n lines starting with the current line.)
REPLACE : /xxx/ or R : /xxx/	Replace the next line which begins with the string xxx.
REPLACE : /xxx/; n or R : /xxx/ ; n	Replace the next n lines which begins with the string xxx. (* can be used instead of n to replace all such lines.)
REPLACE : /xxx/ , /yyy/ or R : /xxx/ , /yyy/	Replace the block of lines starting with the line that begins with the string xxx through the line that begins with the string yyy. (A repeat field can be used with this form also.)
REPLACES : /yyy/ or RS : /yyy/	Replace specified string.
REPLACES : /yyy/ ; n or RS : /yyy/ ; n	Replace n successive occurrences of the specified string. (* can be used instead of n to replace all such occurrences.)
REPLACES : /yyy/, /zzz/ or RS : /yyy/, /zzz/	Replace text starting with the string yyy and ending with the string zzz. (A repeat field can be used with this form also.)

REPLACES : /st1/+. . . /stn/ or R : /st1/+ . . . /stn/	Replace line containing all of the specified (a maximum or ten) strings. (A repeat field can be used to replace n or all such lines.)
REPLACE ES : /st1/-. . . /stn/ or R : /st1/- . . . /stn/	Replace line containing all of the specified (a maximum or ten) strings. (A repeat field can be used to replace n or all such lines.)
REPLACEO : /yyy/ ; m or RO : /yyy/ ; m	Replace the mth occurrence of "yyy". (Repeat field permitted; search field may have the same form as REPLACES.)

When replacing strings of text that do not span lines, the build mode may be bypassed. The command and the entire operand field must be on the same line. This format does not accept a carriage return before the final delimiter. This added string is called the insert field. A few variants of the commands are shown below.

Command	**Execution**
REPLACE : /xxx/ : /bbb/ R : /xxx/ : /bbb/	Replace line which begins with the string xxx with the line bbb.
REPLACE : /xxx/ ; n : /bbb/ R : /xxx/ ; n : /bbb/	Replace the next n lines that begin with the string xxx with the line bbb. (* can be used instead of n to replace all such lines.)
REPLACES : /yyy/ ; n : /bbb/ RS : /yyy/ ; n: /bbb/	Replace n successive occurrences of the string yyy with string bbb. (* can be used instead of n to replace all such occurrences.)
REPLACES : /yyy/, /zzz/ : /bbb/ RS : /yyy/, /zzz/ : /bbb/	Replace text between points yyy and zzz, inclusive, with string bbb. (A repeat field can be used with this form.)
REPLACEO : /yyy/ : /bbb/; m ; n RO : /yyy/: /bbb/ ; m ; n	Replace the mth occurrence of "yyy" by "bbb" and starting from there repeat n times. (All search fields permitted for REPLACES are permitted.)

To replace a string of characters, use REPLACE in the string mode with a search field, with or without a repeat field. Replacement begins at the first character position in the string where the search string was found and ends at the last position of that string. If a repeat field is specified, n identical replacements are performed (unless end of file is encountered first).

```
-PRINT
A program must meet two primary requirements
-REPLACES:/ery/:/ary/
-PRINT
A program must meet two primary requirements
-
```

To replace a complete line, use REPLACE in the line mode, with or without a search field and/or repeat field. The search field, when used, must contain the characters unique to the beginning of the line. When no search field or repeat field is given, the line where the pointer is located is replaced.

Example:

```
-print
time-sharing languages
-REPLACE:/til/:/time-sharing system/
-PRINT
time-sharing system
-
```

Example:

```
-print
time-sharing languages
-replace
enter
*time-sharing system
*(carriage return)
-print
time-sharing system
-
```

When the repeat field is used, the lines beginning with the specified search string are replace number of times indicated. If no search field given, the indicated number of lines is replaced.

```
-PRINT;4
The time-sharing system uses a technique by which
programs are handled in parallel.  A supervisory
program acts as a controller of these programs,
controlling "STOP" and "GO"
-backup;3
-replace;2
enter
*A time-sharing system
*(carriage return)
-print;3
A time-sharing system
supervisory program acts as a controller of
these programs, controlling "STOP" and "GO"
```

To replace a string specified only by its beginning and ending characters, particularly if the points in the ellipsis are on separate lines, use REPLACE in the string mode with an ellipsis using two search strings. A repeat field may be used. Text affected by the commands in the next example is underlined.

```
-print;*
Data is fed from the terminal directly to the computer
and answers are received quickly at the same terminal.

The program can be corrected or changed by the user as
if he were conversing by phone, end of file - request
executed 6 times
-b;6
-replaces:/same/,/the/
enter
*same terminal.
*(blank, carriage return)
*If the program contains a mistake, the
*computer informs the user.
*(blank, carriage return)
*The
*(carriage return)
-b;7
-print;*
Data is fed from the terminal directly to the computer
and answers are received quickly at the same terminal.

If the program contains a mistake, the computer informs
the user.

The program can be corrected or changed by the user as
if he were conversing by phone, end of file - request
executed 9 times
```

5.3.8 COPY Command

The COPY command copies a specified portion of text from the current file and holds it in reserve for a PASTE command. The copied text is not removed from the file. Several sequential COPY commands can be given. The collected text can be inserted with a PASTE command. Examples of the use of COPY are included with the PASTE examples. The format and execution are as follows:

Command	**Execution**
COPY	Copy the current line. (A repeat field can be used with this form. It will copy n lines starting with the current line.)
COPY : /xxx/	Copy line that begins with the string xxx.
COPY : /xxx/ ; n	Copy the next n lines that begin with xxx. (* can be used to copy all such lines.)
COPY : /xxx/ , /yyy/	Copy the block of lines starting with the line beginning with xxx through the line beginning with yyy. (A repeat field can be used to copy n or all such blocks of lines.)
COPYS : /yyy/	Copy the line containing specified string.
COPYS : /yyy/ ; n	Copy n occurrences of the specified string. (* can be used to copy all such strings.)
COPYS: /yyy/ , /zzz/	Copy text between points yyy and zzz, inclusive. (A repeat field can be used with this form also.)
COPY : /st1/+ . . . /stn/	Copy line containing all of the specified (a maximum of ten) strings. (A repeat string can be used to copy n or all such lines.)
COPY : /st1/- . . . /stn/	Copy line containing all of the specified (a maximum of ten) strings. (A repeat string can be used to copy n or all such lines.)
COPYO : /yyy/; m ; n	Copy the line containing the mth occurrence of yyy and repeat n times.

5.3.9 CUT Command

The CUT command performs the same functions as COPY, except that the copied text is deleted from its present location in the current file. Examples of this are included with the PASTE examples. The formats and execution are as follows:

Command	Execution
CUT	Copy and remove the current line. (A repeat field can be used with this form. It will copy n lines starting with the current line.)
CUT : /xxx/	Copy and remove the line that begins with the string xxx.
CUT : /xxx/;n	Copy and remove the next n lines that begin with xxx. (* can be used to copy and remove all such lines.)
CUT : /xxx/ , /yyy/	Copy and remove the block of lines starting with the line beginning with xxx through the line beginning with yyy. (A repeat field can be used to copy and remove n or all such blocks of lines.)
CUTS : /yyy/	Copy and remove the line containing the specified string.
CUTS : /yyy/ ; n	Copy and remove the lines containing n occurrences of the specified string. (* can be used to copy and remove all such strings.)
CUTS : /yyy/ , /zzz/	Copy and remove text between strings yyy and zzz, inclusive. (A repeat field can be used to copy and remove n or all such occurrences of text.)
CUT : /st1/+ . . . /stn/	Copy and remove line containing all of the specified (a maximum of ten) strings. (A repeat field can be used to copy and remove n or all such lines.)
CUT : /st1/- . . . /stn/	Copy and remove line containing all of the specified (a maximum of ten) strings. (A repeat field can be used to copy and remove n or all such lines.)
CUTO : /yyy/ ; m	Copy and remove the line containing the mth occurrence of the specified string. (A repeat field can be used to CUT n or all such strings.)

5.3.10 PASTE Command

The PASTE command inserts the collected CUT or COPY text into the current file after the specified location. In order to PASTE the copied text after more than one location, successive or repeated PASTE commands must be used. The PASTEB command inserts the collected CUT or COPY text before the specified location. Once a PASTE command has been executed, the next COPY or CUT command wipes out the previously accumulated COPY or CUT text.

In the following list the PASTEB command is not illustrated. In each instance, the words "insert text after" would be "insert text before" when using PASTEB.

Command	**Execution**
PASTE	Insert text after the line at which the pointer is currently located.
PASTE : /xxx/	Insert text after the line that begins with the string xxx.
PASTE : /xxx/ ; n	Insert text after each of n lines that begin with xxx. (* can be used to insert after all such lines.)
PASTES : /yyy/	Insert text after string yyy.
PASTES : /yyy/ ; n	Insert text after each of n successive occurrences of string yyy. (* can be used to insert after all such occurrences.)
PASTE : /st1/+ . . . /stn/	Insert text after a line containing all specified (a maximum of ten) strings. (A repeat field can be used to insert text after line containing n or all such lines.)
PASTE : /st1/ - . . . /stn/	Insert text after line containing one or more of the specified (a maximum of ten) strings. (A repeat field can be used to insert text after n or all such lines.)
PASTEO : /yyy/ ; m	Insert text after the mth occurrence of yyy. (A repeat field can be used to PASTE after the n or all such strings.)

The area that contains the collected CUT and COPY material is known as a temporary file *ed2 (See *TSS Reference Manual* for explanation of temporary files). The area can be initialized to no content by the system commands "erase *ed2" or "remo *ed2".

To cut and paste one or more lines, use CUT in the line mode, with or without a search field and/or repeat field. If both a search field and repeat field are used, the indicated number number of lines beginning with the specified string is copied, removed, and is then ready to be inserted by PASTE. If no search field is used with the repeat field, the indicated number of lines is copied and removed, beginning at the location of the pointer.

```
-print;*
Data is fed from the terminal directly to the computer
and answers are received quickly at the same terminal.

If the program contains a mistake, the computer informs
the user.  The program can be corrected or changed by
the user as if he were conversing by phone, except in
this case, the conversation is typed or displayed,
depending upon the type of terminal in use.
end of file-request executed 11 times
-b
-find:/quickly/
-find;1
-cut;3
-paste:/of/
-b
-print;*
Data is fed from the terminal directly to the computer
and answers are received quickly at the same terminal.
The program can be corrected or changed by the user as
if he were conversing by phone, except in this case,
the conversation is typed or displayed, depending upon
the type of terminal in use.

If the program contains a mistake, the computer informs
the user.
end of file-request executed 11 times
```

To paste the same text in several locations, use CUT or COPY, then successive PASTE commands, one for each insertion needed. The example illustrates a form letter and mailing list contained in the same file. In this case, a repeated PASTE command is used, since each insertion is made following a line beginning with the same word.

```
-PRINT;*

We take great pleasure in announcing
             .                  (This part of the
             .                   file will be copied
             .                   into collector file)
Yours very truly,

Widget Corporation
935 Elm Street
Wickenberg, AZ
```

```
Mr. A. A. Adams
P.O. Box 17
Nashville, TN

Dear Mr. Adams:
                                        (Collector file
Mr. X. Y. ZOLTAN                         pasted here)
195 Van Buren Ave.
La Jolia, CA

Dear Mr. Zoltan:
end of file - request executed nn times
-B
-COPY:/ /,/Wick/
(The space character between the first set of
delimiters causes the blank line at the beginning of
the file to be included with the copied text.)
-PASTE:/Dear/;*
end of file - request executed 2 times
-B
-FIND:/Mr./
-PRINT;*
Mr. A. A. Adams
935 Elm Street
Wickenberg, AZ

Dear Mr. Adams:

We take great pleasure in announcing
              .
              .
              .
              .
Yours very truly,

Widget Corporation
934 Elm Street
Wickenberg, AZ

Mr. X. Y. Zoltan
195 Van Buren Ave.
La Jolia, CA.
```

```
Dear Mr. Zoltan:

We take great pleasure in announcing
               .
               .
               .
               .
Yours very truly,

Widget Corporation
934 Elm Street
Wickenberg, AZ
end of file - request executed nn times
```

5.3.11 AFTLIN Command and BEFLIN Command

"AFTLIN" and "BEFLIN" are mnemonics for "AFTer LINe" and BEFore LINe". The commands insert data at the end or at the beginning of a line.

The "AFTLIN" (short form A) and "BEFLIN" (no short form) commands insert data before or after a line or a number of lines specified in the repeat field. Input data can be entered in two ways.

The first list illustrates the use of the command when the insert field cannot be contained in one line.

Command	**Execution**
A A ; 1 BEFL BEFL ; 1	Following the "enter" prompt, one line may be typed in build mode. If more than one line is entered, only the first is used and the rest are discarded.
A : n BEFL ; n	Following the "enter" prompt, n lines may be typed in build mode. If more than n lines are entered, only the first n are used and the rest are discarded. If less than n lines are entered, only those entered are used, but the pointer moves forward n times and TEX leaves lines for which there is no matching entry intact. (* may be used in the repeat string.)

The second list illustrates the use of the command when the insert field can be contained in one line.

Command	**Execution**
A : /yyy/ A : /yyy/ ;1 BEFL : /yyy/ BEFL : /yyy/ ;1	The string "yyy" is added after or before the current line.
A : /yyy/ ; n BEFL : /yyy/ ; n	The string "yyy" is added after or before n lines starting with the current line. (* may be used in the repeat string.)

Examples:

The following examples all add the string "abcdef" to the beginning of a line.

```
1.   -BEFL:/abcdef/

2.   -BEFL:/abcdef/;1

3.   -BEFL
     enter
     *abcdef
     *(carriage return)
     -

4.   -BEFL
     enter
     *abcdef
     *(carriage return)
     -
```

The next example illustrates expansion of a list:

```
     -p:*             (Print the current file)
     Frank
     Betty
     George
     Nancy
     Joyce
     end of file-request executed 5 times
     -b befl:/The color of/;* (add string before every
                               line)
     -b a:/'s hair is /;*     (add string after every
                               line)
     -b a;*
     enter
     *brown
     *blonde
     *black
     *auburn
     *brunette
     *(carriage return)
     -b p;*              (print the result)
     The color of Frank's hair is brown
     The color of Betty's hair is blonde
     The color of George's hair is black
     The color of Nancy's hair is auburn
     The color of Joyce's hair is brunette
     end of file-request executed 5 times-
```

5.3.12 T Command (Print Transparent)

The T (Print Transparent) command is similar to the PRINT command. A repeat field may be specified with it. The T used alone prints the current line only if it contains a transparent character. When T is used with a repeat field, it examines every line beginning with the current line until it has examined the number of lines specified in the repeat string. The last line so examined becomes the current line. Only those lines in the span of lines examined that contain transparent characters are printed showing the character(s) bracketed by asterisks and translated to ASCII mnemonic characters. For example, if the current line contains a backspace, it would be printed as follows:

```
-T

This line has a backspace *BS* embedded.
```

Refer to "Star Functions" in Section 3 for a list of mnemonic characters used. When a repeat string is used (e.g., T;n), the command is executed for the number of lines requested or to the end of file, whichever comes first.

5.3.13 TF Command (Find and Print Transparent)

The TF command finds the first line (including the current line) that contains a transparent character and prints it. The pointer is moved to the line that was found.

5.3.14 MARK Command

The MARK command searches the file for a line starting with a ".MARK" or ".MARK filename". If a line is located, and a file name is specified, that file is accessed and the text on the specified file replaces the ".MARK" line. If the line does not contain a file name, the user is queried at the terminal as the file to be accessed. If a ".MARK" line is not found, the user is so informed. After a MARK command the pointer is at the last line inserted.

Files accessed utilizing the ".MARK filename" sequence may themselves contain embedded ".MARK" lines. If the MARK command is followed by a repeat string ;*, each time a normal end of file condition is reached following a successful access of a specified "MARK" file, the current file is searched again to ensure that the accessed file did not contain a ".MARK" line. The MARK command observes the following rules:

- The action of the MARK command cannot be verified.
- Catalog/file strings as well as multiple files are permitted, e.g., file1;file2.

Example 1:

```
-new
enter
*is the time for
*all good men
*
-save time
DATA SAVED-TIME
-new
enter
*Now
*.mark time
*to come
*
-b p;*
Now
.mark time
to come
end of file-request executed 3 times
-b mark p;*
all good men       (Note: the pointer is at the last
                          inserted line)
to come
end of file-request executed 2 times
-b p;*
Now
is the time for
all good men
to come
end of file-request executed 4 times
-
```

Example 2:

```
-old time
-build
enter
*.mark come
*
-resa time
DATA SAVED-TIME
-new
enter
*to come to
*
-save come
DATA SAVED-COME
-new
enter
*Now
*.mark time
*the aid
*
-b p;*
Now
.mark time
the aid
end of file-request executed 3 times
-b mark;*
end of file-request executed 2 times
-b p;*
Now
is the time for
all good men
to come to
the aid
end of file-request executed 5 times
-
```

5.3.15 TRUL and TRUR Commands

The formats of the commands are:

Command	Execution
TRUL n	Delete all characters to the left of the specified character position n.
TRUR n	Delete all characters to the right of the specified character position n.

The mnemonics TRUL and TRUR stand for "TRUncate Left" and "TRUucate Right" respectively. They must be followed by a blank and by a value that is a positive decimal column number. Columns are numbered left to right starting with column "1". The value may be the result of an evaluation of an expression or a decimal number. Negative or zero column numbers cause an error message to be issued. The entire current file (without regard to pointer position) is truncated, left or right, leaving the column specified in the command intact. The pointer will end up at the end of file.

Example:

```
-b p;*
12345aaa9999
12345bbb9999
12345ccc9999
end of file-request executed 3 times
-trur 9 trul 5 b p;*
5aaa9
5bbb9
5ccc9
end of file-request executed 3 times
```

5.4 Multiple Current Files

Only one current file is passed from one TSS subsystem to the next. This file is always named *SRC in the AFT. TEX, however, can maintain auxiliary current files. Any one of these can quickly become the current file through the use of the "CF" commands. A maximum of four files can be managed this way. Each auxiliary current file has its own current line pointer. These pointers are not disturbed when file switching is performed. Each file is assigned a current file identifier. Upon entry to TEX, the current file is assigned the identifier "a". The other files are assigned identifiers by the user. The file identifiers are subject to the same rules as variable names-they may contain alphabetic or numeric characters, or the underline. They must begin with alphabetic characters. They may not be more than 40 characters in length. Current files may be switched in two ways:

1. Directly from one file to any other file.

2. Using a stack pushing each file down into the stack and popping them back up in the order "last in first out." Once a current file has been pushed down into the stack, it may only be retrieved by popping it up from the stack.

The commands CF and CFD may be used to create new auxiliary current files. A new current file is created as a file without any lines of data. The CF or CFD command may be followed by an OLD, NEW, INSERT, PASTE, or other command to place data on the file.

5.4.1 CF Ident (Switch Current Files)

The identified file becomes the new current file. The previous current file is saved for future reference. If the identified file does not exist, a new current file is created.

5.4.2 CFD Ident (Switch Current File and Push Down)

The identified file becomes the new current file. The previous current file is pushed into the current file stack. If the identified file does not exist, a new current file is created.

5.4.3 CFU (Pop up and Switch Current File)

The file on top of the current file stack becomes the new current file. The previous current file is saved for future reference.

5.4.4 CFC Ident (Clear Auxiliary Current File)

The identified file is released and its identifier is cleared. An auxiliary current file while it is the current cannot cleared when it is the current file.

5.4.5 CFC * (Clear all Auxiliary Current Files)

All auxiliary current files except the current one are released.

5.4.6 CFL

All auxiliary current file identifiers are printed at the terminal. The current file and the files in the current file stack are so identified.

Notes

6. Mode Commands

6.1 LINE And STRING Commands

The default mode for TEX editing commands is the line mode. This default can be changed to string mode by entering the STRING command. It can be restored to line mode by entering the LINE command.

The main difference between line and string mode operation is that in line mode any search is for strings only at the beginning of the line and in string mode any search is for strings contained anywhere in the line. For a detailed discussion of the use of the various commands, see Section 4.

The line and string modes affect the operation of the following editing commands:

```
COPY
CUT
DELETE
FIND
INSERT
PASTE
PRINT
REPLACE
```

These editing commands operate in the default mode in the absence of the "L" or "S" modifiers. The exceptions to this rule are:

- AND and OR search functions search only in string mode. The associated editing function operates only in the line mode.
- Verbs using the occurrence modifier operate only in string mode.
- A command without a search string operates in line mode.

NOTE: Because the first four characters of the TEX command STRING and of the system command STRIP are the same, TEX interprets the abbreviation STRI as a string command. The STRIP command must be spelled out.

Example:

```
-string
-print;6         (string mode ignored)
A computer program is a set of instructions that
tells a computer how to accomplish a specific
task.  Each instruction is performed in the
sequence specified by the program.  In this way,
the computer processes and produces information
as directed by the program.
-backup
-replace:/task/  (Acts in string mode)
enter
*job
*(carriage return)
-print:/job/     (Acts in string mode)
job.  Each instruction is performed in the
-
```

Occurrence Mode

The occurrence mode can be invoked by a verb modifier for only one command at a time.

The use of the "O" modifier allows the user to operate on a specific occurrence of a string. A operand of the form(; n) specifies which occurrence. A repeat string may follow the occurrence string. The use of the "O" modifier also puts the editing command in the string mode. The "O" and "S" modifiers cannot be used together.

Suppose a line contained the following repetitive data:

```
D.....D.....D.....D.....D.....D.....D.....D.....D.....D.....
```

Replacement of the ninth occurrence of the string "D" would be awkward without replacing the entire line.

Using "Occurrence" modifier, replacement of the character would be performed as follows:

```
-RVO:/D/:/X/;9    (Replace and verify the ninth occurrence of
                    the character "D" with "X")

D.....D.....D.....D.....D.....D.....D.....D.....X.....D.....
```

To replace every second occurrence of the character "D" with the character "Y" three times, proceed as follows:

```
-RVO:/D/:/Y/;2;3  (The occurrence string (;2) indicates which
                   occurrence, the repeat string (;3)
                   indicates the number of times)

D.....Y.....D.....Y.....D.....Y.....D.....D.....X.....D.....
```

6.2 VERIFY And NOVERIFY Commands

The VERIFY command turns on the verify mode. This mode causes printed verification of editing commands at the terminal.

For the FIND and BACKUP commands, the verify mode prints out the new current line after the command is executed. If the pointer is at the end of file, "end of file" is printed out.

For the REPLACE, INSERT, and PASTE commands in line mode, the verify mode prints the line preceding the change, the change itself, and the line following the change. Although the line following the change is printed, the pointer remains at the last line changed.

For the DELETE and CUT commands in line mode, the verify mode causes printing of the lines above and below the deletion.

For text altering commands in string mode, the verify mode causes printing of the one or more lines affected by the change.

The NOVERIFY command cancels the verify mode.

TEX OLD filename VERIFY	(TEX line positioning and text altering commands are verified by the TEX subsystem upon execution. The VERIFY command remains in effect until cancelled by a NOVERIFY command.)

Any editing command may be set to verify its action by using the letter "V" in the mode modifier of the command.

-FINDV : /xxx/ ; n	(Upon finding the nth occurrence of the specified line, the line is printed out. Only the last line is printed here. Use PRINT to print all lines matching the search string.)
-REPLACEVS : /xxx/ : /bbb/	(Upon replacement of string xxx by string bbb, the altered line is printed out.)

The "V" modifier only affects the command it modifies.

6.3 CASE And NOCASE Commands

The CASE command turns on the case mode. When TEX is in case mode, the search operations of editing commands are performed without regard for whether the text in the search string or the current file is in uppercase or lowercase. In case mode, the SCAN command and the scan and comparison operators also ignore case. The NOCASE command cancels the CASE command.

6.4 SHIFT And NOSHIFT Commands

There are two shift modes. One converts all alphabetic characters entered within a delimited string or in the build mode to lowercase unless they are delimited by the defined shift character. The other converts input alphabetics to uppercase unless they are similarly delimited. The NOSHIFT command cancels either shift mode command.

Command	**Execution**
SHIFT UPPER][	Text entered at the terminal between two case shift characters][is made uppercase. Other text entered at the terminal is made lowercase. When this case conversion is performed, the case delimiters are discarded. When text is printed at the terminal, case delimiters are inserted around uppercase text.
SHIFT LOWER][	Text entered at the terminal between two case shift characters][is made lowercase. Other text entered at the terminal is made uppercase. When this case conversion is performed, the case delimiters are discarded. When text is printed at the terminal, case delimiters are inserted around lowercase text.

The symbol][is used here to represent an ASCII character declared by the user.

The shift model is provided for use with single case terminals, e.g. , Teletype model 33.

The example illustrates a dual shift terminal in shift upper mode. It shows the difference between the text that was typed in, the text that was typed out, and the text stored in the file.

```
-shift upper~
-new
enter
*-The time is now 12:10 pm~          (convert to uppercase)
*
-p
~THE~ ~TIME~ ~IS~ ~NOW~ 12:10 ~PM~   (TEX display)
-noshift
-p
THE TIME IS NOW 12:10 PM             (characters displayed as
                                      stored)
-shift upper ~
-new
enter
*The time is now 12:10 pm            (convert to lowercase)
*
-p
the time is now 12:10 pm             (TEX display)
-noshift
-p
the time is now 12:10 pm             (characters displayed as
                                      stored)
-
```

The second example shows how an uppercase terminal would work.

```
-SHIFT LOWER '
-NEW
ENTER
*I'T IS NOW 12:30 PM'      (everything between apostrophes is
                            to be made lowercase)
*
-P
I'T' 'IS' 'NOW' 12:30 'PM' (TEX separates shifted letters from
                            non-shifted ones)
-
It is now 12:30 pm         (The computer contains the intended
                            text)
```

6.5 LIMIT And NOLIMIT Commands

The LIMIT command can be used only with line-numbered files. It sets a mode that limits all further operations on text to lines whose line numbers fall in the range specified by the LIMIT command. The NOLIMIT command cancels the limit mode.

Sample usage:

```
       LIMIT 203,506
```

This mode establishes a subset of a file in which the line numbered 203 is the first line and line number 506 is the last line. Leading zeros need not be specified in the LIMIT command. All subsequent commands are executed only within the specified range, i.e., lines which begin with numbers between 203 and 506.

If, when the "LIMIT" is set, the pointer is outside of the range specified, the pointer is positioned automatically to be within the range. When returning to the normal nolimit mode, the pointer remains at the last line accessed while in the limit mode.

If the line numbers of the line(s) inserted are inside the specified range, the limits mode remain in effect. If the line numbers of the line(s) inserted are outside the range specified by the LIMITS command, the insertion is completed, TEX will exit the LIMITS mode, and a warning message is printed at the terminal.

Automatic line numbering (#AUTO) or SEQUENCE cannot be used while the limit mode is set.

Example:

```
-p;*
10 a
20 b
30 c
40 d
50 e
end of file - request executed 5 times
-b
-limits 20,40
iv:20
enter
*31 q
*sss
*
20 b
31 q
sss
30 c
file is not properly line numbered-limit mode not in
effect
-
```

6.6 IGNORE And NOIGNORE Commands

The IGNORE command turns on the ignore mode. This mode causes TEX to disregard line numbers while making modifications to a file using string functions. The NOIGNORE command cancels the ignore mode.

When the first character of the text following a line number is numeric, some temporary nonnumeric character, (e.g., a space) should be inserted between it and the line number.

6.7 OCTL And NOOCTL Commands

The OCTL command turns on the octl mode and reserves a character for use with this mode.

```
OCTL ][
```

where "][" becomes the "OCTL character." This mode permits the entry of ASCII characters other than those available on the keyboard of the remote terminal. (A chart of ASCII characters is found in Appendix A.) When such a character must be entered in a delimited string or in build mode input, the user enters "][" followed by the OCTAL representation of the character. The octl mode is cancelled by the NOOCTL or NOCTL command. The rules for the use of "][" are:

- At most, three octal digits (0-7) may follow the character.
- If only one or two octal digits follow, the number is assumed to be padded left with zeros.
- Two successive characters "][" are interpreted as one "][".
- If no octal digit follows it, the character "][" itself is assumed.

For example, if the special character is declared to be "$":

```
typed in        interpreted as

$132Q           ZQ     (132 is the octal value of "z")
$045X           %X     (45 is the octal value of "%")
$45X            %X     (leading zero assumed)
$$$45X         $%x     ("$$" is interpreted as "$")
$$7             $7     ("$$" is interpreted as "$")
$9              $9     (9 is not an octal number)
```

Examples of use of the OCTL commands:

```
-OCTL $   (The dollar sign is declared to be the octal
          character.)
-P       (Print the current line.)
.......on at 9.084 - off at 9.140 on 06/24/75.............
- RVS:/at/;2:/$100/  (Replace the string "at" twice with the
                      octal character 100 (@) and verify.)
.......on @ 9.084 - off @ 9.140 on 06/24/75.............
```

The OCTL mode may be used while in the build mode of TEX provided it was set prior to entering "BUILD". For example:

```
-OCTL%            (The percent sign is the octal character.)
-BUILD            (User enters the build mode.)
enter             (TEX response.)
*.......on %100 9.084 - off %100 9.140 on 06/24/75.........
*(carriage return) (User exists the build mode.)
f;* bv;1          (Print the last line)
end of file
.......on @ 9.084 - off @ 9.140 on 06/24/75.........
```

6.8 #NO And #YES Commands

The #NO mode allows a user to print a line numbered file without printing the line numbers. This mode is cancelled by the #YES command.

6.9 MASK And NOMASK Commands

The MASK command turns the mask mode on, NOMASK turns it off. In this mode the user declares a special character:

```
MASK ][
```

where "][" means "any character." A search field containing a mask character matches a string in a line if it matches the pattern of characters in the search string one for one. The mask characters matches any character; all others must match precisely. The mask character can only be used in the search string. For example, if # is the mask character the pattern ##ABC##XYZ matches 123ABC34XYZ. It also matches QrABC9zXYZ.

```
-mask #
-p;2
   12ABC34XYZ
...  QrABC9zXYZ
-b
-ivs;2:"##ABC##XYZ":"...."
...12ABC34XYZ....
.....QrABC9zXYZ....
-
```

6.10 SUBS And NOSUBS Commands

The SUBS][command turns on the subs mode. The][represents any ASCII character the user wishes to declare as the subs character. The NOSUBS command turns off the SUBS mode. SUBS][may be used to change the current SUBS character to a different one. Only the last specified SUBS character is in effect.

As each command input line is fetched, it is examined for SUBS characters (][) in effect, if any. If two or more are present, they are paired, left to right. The input characters bracketed by each pair must represent a legal expression as defined in Section 3. Each expression and its two bracketing SUBS characters are replaced by the value of the expression. This is done after removal of any leading label and before the input line is examined for commands. If the input line only contains a single SUBS character, it is not changed.

Labels are not examined for SUBS characters (see Section 4 for definition of labels).

The expressions bounded by SUBS characters are evaluated using the string variable values in effect before any commands in the current input line are executed. A newly-defined SUBS character is not effective until the following input line is fetched.

If a series of input lines are to be passed to another TSS subsystem, only the first line is examined for the SUBS character.

6.11 SEF Command

The SEF (Suspend Executive File) command sets a special "SEF" mode for the execution of all commands following it on a line. The SEF command has no effect when it is entered at the terminal. It is useful only in executive files. In this mode, each command is executed as if it had been entered from a terminal. Responses from/to these commands are also directed to the terminal.

Example:

An executive file contains:

```
sef list
```

The current file is listed at the terminal. Without the sef command nothing would happen at the terminal.

Example:

An executive file contains:

```
out:"Now we compose" sef runoff
```

Execution is suspended when "sef" command is encountered, and the "READY" message from RUNOFF is issued to the terminal. TEX continues to suspend execution of the executive file while the user enters commands known to RUNOFF. Execution is resumed upon exit from RUNOFF, as the line ends there.

If command file or deferred processing is in effect, the SEF command causes the inputs to be fetched from the command file, and the response to be directed to the specified output file (if any).

6.12 FOUT And NOFOUT Commands

The FOUT command turns on the file out mode. The NOFOUT command turns off the file out mode. The status of the fout mode is not kept in *svmd.

The file out mode offers three options of directing output to a file without otherwise affecting the operation of TEX.

- a command and responded trace
- a full trace using the suffix "d"
- response trace using the suffix "x"

Where "d" stands for debug and "x" for exclude.

The output can be directed to either clear and write to a file, or using the suffix "a", to append to a file.

```
FOUT    cat/filestring   The command and response trace is
                         directed to a cleared file.
FOUTA   cat/filestring   The command and response trace is
                         appended to a file.
FOUTD   cat/filestring   The complete trace is directed to a
                         cleared file.
FOUTAD  cat/filestring   The complete trace is appended to a
                         file.
FOUTX   cat/filestring   Only the responses are directed to a
                         cleared file.
FOUTAX  cat/filestring   Only the responses are appended to a
                         file.
```

The file out can be used to debug a program or to capture output that is needed for later use.

6.13 TEX Default Modes

TEX always sets the following initial modes when it is invoked:

```
line    noignore   nomask   noshift   noverify   notrace
nocase  nolimit    nooctl   nosubs    #yes       nobreak
nofout
```

Trace and notrace modes are described in Section 4.

6.14 BREAK And NOBREAK Commands

The BREAK command has the following format:

```
BREAK n
```

Where "n" contains a number for limiting the range of the break mode.

The BREAK command turns on the break mode. The BREAK mode is used to postpone break processing in an executive file. Instead of terminating file execution, the break key on the terminal sets the star variable *break to the value "t". If a TEX command is being processed when the break is received, the command will be completed and file execution will continue with the next command. If another TSS subsystem is in execution when the break is received the subsystem is interrupted, control is returned to TEX, and executive file execution continues with the next available command line.

*break is returned to the value "f" when it is referenced or when the BREAK command is executed. The argument "n" specifies the maximum number of TEX commands over which breaks can be postponed. Command counting begin as soon as the BREAK command is executed. Counting begins over again each time *break is referenced.

The break mode limit "n" allows the user to protect himself from accidental uninterruptible loops. The limit "n" should be set slightly larger than the maximum number of TEX commands to be executed between tests of *break. Break mode can only be used during executive file processing.

The NOBREAK command turns off the break mode. The break mode is also turned off when control is returned the terminal following executive file processing.

Example:

```
break 10
!loop x=x+1 y=y+1
if ^*break goto !loop
out: "break received"
nobreak return
```

The loop will increment x and y the same number of times (assuming no overflow). When the break is received, the loop will be exited and the "break received" message will be printed at the terminal. Since less than 10 commands are executed between references to *break the command counter will not expire.

Example:

```
break 100
!loop x=x+1 goto !loop
if *break goto !brkhdlr
```

By mistake the IF statement was not placed inside this loop. The loop is uninterruptible until 100 commands have been executed. Any break held pending is then honored and file execution is stopped. If the break is received after 100 commands have been executed, it is processed as soon as possible.

NOTES:

1. The break mode should be used carefully because uninterruptible loops can be created if *break is not tested correctly or if the break limit is set to a very large value.

2. The break mode should be used carefully when TEX invokes other TSS subsystem from an executive file. To determine whether the other subsystem has been interrupted, test *break before and after the other subsystem is invoked. The other subsystem may use parameters passed on the same command line that invokes the subsystem, but should not request additional input lines because TEX might treat these as command lines if a break is received.

7. Responses from TEX

By convention, Time Sharing System (TSS) responses are always shown in uppercase letters. TEX responses are shown in lowercase letters to make it easier to tell which system responds. Responses from TSS, other subsystems or user programs are not modified by TEX.

7.1 Prompting Or Execution Response

- (hyphen)	The last command has been executed and TEX is ready to accept either another TEX command or a TSS command.
enter *	This response to a REPLACE, INSERT, NEW, BEFLIN, AFTLIN or BUILD command informs the user that the replacement, insertion, or additional text can now be entered. It is referred to as the "build mode" in this document. An asterisk appears on the next line after the "enter" response, indicating that the time sharing data collector now accepts text entry.
end of file - request executed xxxx times	This message occurs following execution of an editing command with a search field or repeat field that reaches the end of the file. "xxxx" represents the number of times the specified function was executed.
end of file	This message occurs when the pointer is at the end of the file following execution of an editing command.
paste not executed, no data	This message occurs when the user failed to cut or copy data prior to issuing a PASTE command.

7.2 Diagnostic Messages

When a mistake occurs in execution of a file, three lines print:

```
executing file "name" line #nnn
(the guilty line)
(the diagnostic message)
```

When executing from a terminal, or from an executive file in trace mode, the guilty line is, of course, already printed; the diagnostic message follows immediately.

'xxxx' is not a known operator

TEX attempted to evaluate an expression. It has encountered the undefined operator, "xxxx".

'xxxx' is not a known variable

TEX attempted to evaluate the nonexistent variable "xxxx".

'xxxx' is not a legal number

TEX attempted to use the string "xxxx" as a number, and cannot for one of these reasons:

- "xxxx" is null.
- "xxxx" contains a non-numeric character other than a leading plus or minus sign.
- The numeric portion of "xxxx" contains more than 62 characters.

'xxxx' is not legal as a variable name

TEX attempted to create or evaluate a variable named "xxxx". This name is illegal for one of the following reasons:

- It is null.
- It is greater than 40 characters in length.
- It contains characters other than the alphanumeric characters or the underline.
- It does not begin with an alphabetic character.

'xxxx' must be true or false

TEX attempted to use the string "xxxx" as a boolean value, and cannot because it does not have one of the following values:

```
"t"
"f"
"true"
"false"
```

'xxxx' no more room for variables

TEX attempted to create or modify the variable "xxxx" and failed because the available space for variable storage has run out/

'xxxx' is in the current file stack

A CF, CFC or CFD command has been attempted and the identified file is in the current file stack.

'xxxx' is not legal command

TEX detected an error in a command format. The quotes contain the portion of the command examined when the error was detected. The error, therefore, was probably detected when TEX attempted to process the rightmost character within the quotes.

'xxxx' is not a legal expression

TEX unsuccessfully attempted to evaluate the field 'xxxx' as an expression.

'xxxx' is not legal as a file ident

A CF, CFC, or CFD command has been attempted with 'xxxx' in the file identifier field. This identifier is not legal for one of the following reasons:

- It is null.
- It contain characters other than the alphanumeric characters or the underline.
- It does not begin with an alphabetic character.

'xxxx' is the current file

A CFC or CFD command has been attempted and the identified file is the current file.

and/or function not allowed

TEX encountered an and-function or an or-function in an editing command where it is not allowed.

and/or functions mixed

TEX attempted to execute a command containing both an and-function and an or-function. These functions cannot be mixed in the same command.

arithmetic overflow

During expression evaluation, an intermediate or final numeric value has exceeded 62 digits.

column number must be 1 or more

Column specified in COLUMN, TRUL, and TRUR commands are numbered left to right starting with 1.

command illegal under deferred processing

TEX attempted to execute an INT or INTC command.

COMMAND UNKNOWN

TSS does not recognize the command, either because it is illegal or because it is misspelled.

current file stack is empty

A CFU command has been attempted with an empty current file stack.

divide check

During expression evaluation, a division by zero or a division yielding an excessively large quotient was attempted.

ellipsis not allowed with "p" or "t" modifier

An editing command with a "p" or "t" modifier also has comma separated search strings.

ending delimiter][missing

A delimited string is missing its ending delimiter.

expression error - parentheses in the wrong place

During left to right expression evaluation, TEX encountered a right parenthesis without a matching left.

expression error - parentheses not paired

During expression evaluation, TEX encountered more left parentheses than right parentheses.

expression error - too many parentheses

The number of real and implied parentheses in an expression exceeds 16.

file ident too long

A CF, CFC, or CFD command has been attempted with a file identifier containing more than 40 characters.

file identifier missing

The required current file identifier is missing from a CF, CFD, or CFC command.

file is not properly line numbered - limit mode not in effect

TEX encountered one of the following in the line mode:

- A line with no line number
- A line with line number out of order

As a result the limit mode is no longer in effect.

incomplete command

TEX command is missing a required field or operand.

incomplete expression

TEX attempted to evaluate an expression and encountered one of the following:

- A blank or end of line immediately to the right of the equal sign in an assignment statement.
- A blank or end of line immediately to the right of an operator in an expression.

label 'xxxx' not found

TEX failed to find the label 'xxxx' during execution of a CALL or GOTO command.

label missing

An exclamation point in a CALL or GOTO command is followed by a blank or end of line.

label too long

TEX attempted to use a label of more than 40 characters.

line number must be 1 or more

TEX attempted a LIMIT command with a line number less than 1.

line too long

TEX attempted to insert or replace one or more lines in the current file greater than the TSS limit of 1272 characters. The operation has been completed using the leftmost portion of the new line(s).

may only be executed from a file

TEX obtained one of the following verbs from a remote terminal or from a command file via CRUN or DRUN:

```
ercall
ergoto
goto
call (intra-file)
return
```

name indirection too deep

During evaluation of a string variable specifier, TEX exceeded 64 indirections.

name too long

TEX attempted to use a variable name of more than 40 characters.

negative repeat or occurrence count

TEX evaluated a repeat field or occurrence field yielding a negative number.

operand 'xxxx' must be one character

A SCANN command or a "scan for not equal" operator has been used with a null B-operand string or a B-operand string of more than 1 character.

permanent current file can not be used with CF commands

A CF, CFC, CFD, CFL or CFU command has been attempted and the current file is permanent.

repeat field not allowed

TEX encountered a repeat field in an editing command where it is not allowed.

search field not allowed

TEX encountered a search field in an editing command where it is not allowed.

second column number must be greater than or equal to the first

TEX attempted an illegal COLUMN command. Columns are numbered left to right starting with "1".

star function is not known

During expressing evaluation, TEX encountered an undefined star function.

string too long

During expression evaluation, TEX attempted to create an intermediate or final result string in excess of 2048 characters.

substitution delimiters not paired

In subs mode, TEX attempted to perform substitutions in a line containing an odd number of subs characters greater than two.

too many calls

The depth of nested calls has exceeded 15.

too many current files

A CF or CFD command has attempted to create a fifth current file.

7.3 TSS Error Messages

A number of standard error messages are issued by the Time Sharing System. These are preceded by a message number usually enclosed by carets. These messages are described in the *TSS Reference Manual*. Further explanation of these messages is available at the terminal through the HELP subsystem. Type "HELP" and the system will respond by asking for the message number. The following are examples of numbered messages you might see:

```
<50>     WORK FILE, FILE TABLE FULL, STATUS 36
<50>     WORK FILE, SYSTEM LOADED, STATUS 40
<50>     WORK FILE, STATUS 10
<52>     CURRENT FILE NOT DEFINED
<50>     WORK FILE, NO FILE SPACE, STATUS 13
```

Notes

A. ASCII Chart

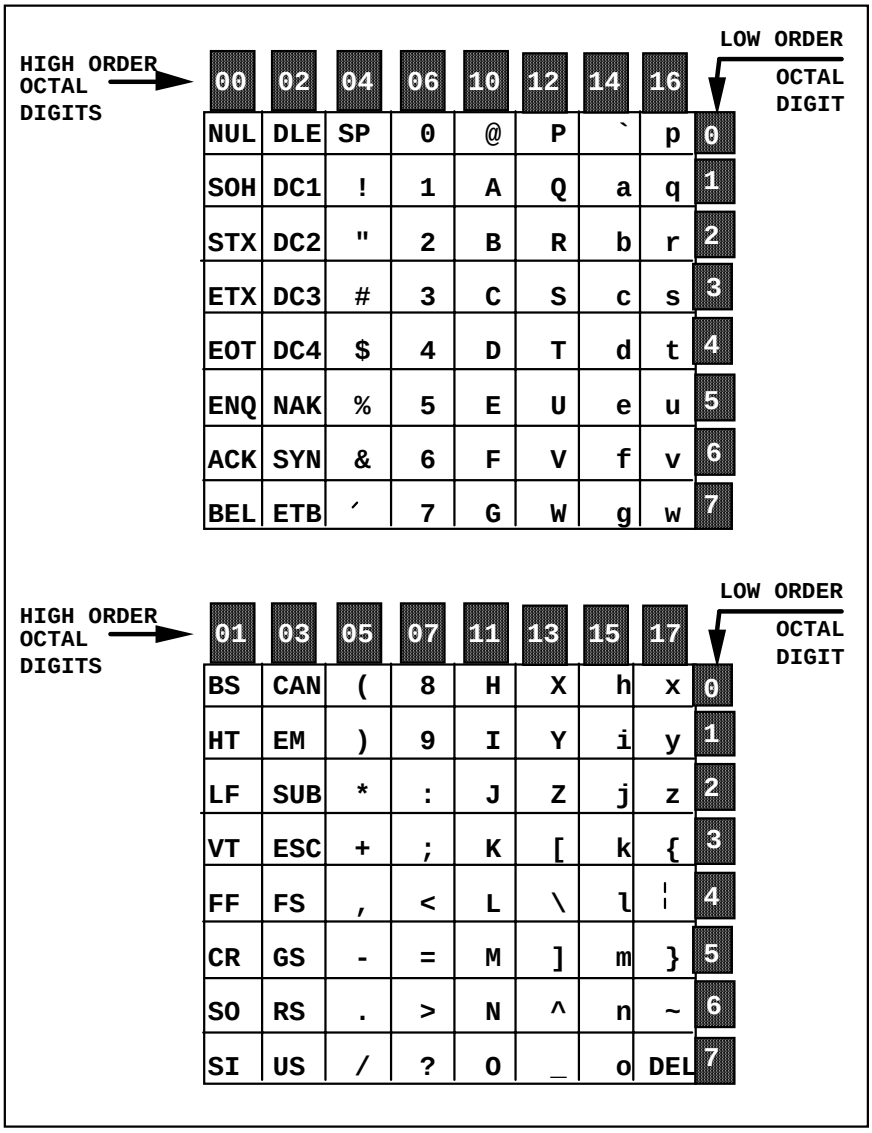

Figure A-1. Chart of ASCII Characters

Notes

B. Differences Between TEX and TEXT Editor Subsystems

1. TEX does not permit spaces between an editing verb and the colon or semicolon when a search field or repeat field is present.

2. Text Editor CASE UPPER and CASE LOWER modes are replaced by the TEX SHIFT UPPER and SHIFT LOWER modes. The CASE mode remains unchanged.

3. Text Editor STANDARD and NORMAL commands are replaced by the TEX NOCASE, NOSHIFT, and NOLIMIT commands.

4. In octl mode, TEX accepts 1 to 3 octal digits following the octl escape character as an octal number. Text Editor converts 3 characters to an octal value, yielding some unexpected results when they are not octal digits.

5. Surprise entry to BUILD mode does not occur in TEX. Example: Text Editor enters BUILD mode upon execution of a BACKUP command when there is no data on the current file.

6. Text Editor T command unconditionally searches the current file for transparent characters. The TEX T command allows the current line (T) or a specified number of lines (T;n) to be searched.

7. TEX does not permit single character abbreviations for LINE, STRING, MODES, and WHERE commands.

8. TEX does not support plus/minus line positioning within an editing command:

EDITOR	**TEX**

`P+5;3`
`  or`
`F;5 P;3`
	`F;5 P;3`

9. TEX does not permit alphanumeric or certain other characters to be used as delimiters in search fields and insert fields.

10. The format of the LIMIT command is different:

EDITOR	**TEX**
`LIMIT:/10/,/20/`	`LIMIT 10,20`

Notes

C. TEX Enhancements for Release WP2.0

1. TEX allows the user to direct a trace of a command execution to a file. FOUT page 6-9.

2. The TEX user can create multiple current files. CF page 5-36.

3. TEX has additional star functions. Pages 3-6 through 3-9.

4. TEX users can apply logical AND/OR operations. Page 3-15 and applicable command descriptions.

5. TEX allows break processing to be deferred in executive files. BREAK/ NOBREAK page 6-8.

Notes

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
France

67 A2 DF72 REV03 Printed in U.S.A.