Ultimate

THE ULTIMATE CORP.

PROC Reference Guide

Ultimate

THE ULTIMATE CORP.

Ultimate PROC Reference Guide

The Ultimate Corp.
East Hanover, NJ

Version 3

Ultimate® PROC Reference Guide
Version 3

Publication Information

Ultimate Operating System Release 10, Revision 210
© 1992 by The Ultimate Corp., East Hanover, NJ

Document No. 6936

CONTENTS

How to Use This Manual

This manual is intended as a reference for programmers and analysts using the Ultimate PROC software package. It covers all aspects of using PROC with the Ultimate system commands and the BASIC programming language. The material is presented in a structured format, with text and graphics integrated into single-topic units.

How the Manual is Organized

Chapter 1 is an introduction to PROC. This chapter describes:

- what PROC is and how to use it
- the PROC buffers
- components of a PROC
- PROC execution
- PROC execution example
- system commands associated with PROC
- BASIC statements that interact with PROC
- PROC commands by function

Chapter 2 is an alphabetical PROC commands reference. Syntax and usage are provided for each command. Examples are given for most commands.

Chapter 3 presents sample PROCs and explains them in detail.

Appendix A consists of a listing of ASCII codes.

Conventions

This manual presents general formats for each of the PROC commands. In presenting and explaining these general forms, the following conventions apply:

Convention	Description
UPPER CASE	Characters or words in syntax definitions shown in upper case must be entered exactly as shown. PROC commands cannot be entered in lower case.
lower case	Characters or words in syntax definitions shown in lower case are parameters to be supplied by the programmer, such as filename, itemlist, and options.
{ }	Braces surrounding a parameter indicate that the parameter is optional.
enter	The word enter means to type in the required text and press RETURN.
↵	The ↵ symbol means press the RETURN key on the keyboard. It is used in examples.
<KEY>	Angle brackets indicate a key other than letters, numbers, or punctuation, such as <ESC>.
`Courier`	Courier typeface is used for messages or prompts displayed by the system and, when boldfaced, for user input.
Helvetica	Helvetica typeface is used for title headings.
bold	In courier font, boldface type is used in syntax definition and to indicate user input.
↑	An arrow in a PROC example indicates the current location of the pointer in an active buffer.
_	An underscore in a PROC example represents a blank.
parameters	Elements in buffers.
current parameter	The data from the current position of the pointer to the end of the parameter.

arguments	Variable parts of the syntax.
statement	Used when referring to a command and its complete text.
filename	The word filename can indicate any of the following, depending on the section of the file to be accessed:

- **dataname**. Specifies the DATA section of a file with the same name as its DICT.
- **dictname,dataname**. Specifies a DATA section when the dictname has multiple DATA sections.
- **DICT dictname**. Specifies the DICT section of a file.
- DICT **dictname,dataname**. Specifies the DICT section of a file. Same as **DICT dictname** above.
- DATA **dataname**. Specifies the DATA section of a file with the same name as its DICT. Same as **dataname** above.
- DATA **dictname,dataname**. Specifies the DATA section when the dictname has multiple DATA sections. Same as **dictname,dataname** above.

Notes

1 Introduction to PROC

The word PROC is used in two ways:

- it is the name of a language just as BASIC, FORTRAN, or COBOL are names of languages
- it is the identification of a stored procedure written in the PROC language

Stored procedures are not called PROC programs; they are simply called PROCs. Therefore, it is appropriate to say that a PROC is a stored procedure made up of PROC commands and Terminal Control Language (TCL) statements. Any sequence of operations that can be keyed in directly by a user at the TCL level can also be prestored via PROC. The PROC then passes the sequence of operations to TCL for execution.

PROCs are often used for generating TCL statements and associated data that the TCL statements may require. PROCs are also commonly used to create menus and then, based on user input, call other PROCs.

A PROC provides the following features:

- a means of storing a sequence of operations that can then be invoked from the terminal by a one- or multi-word command
- conditional execution of statements
- the ability to interactively prompt the terminal user
- the ability to test and verify input data as it is entered from the terminal keyboard

A PROC can exist in a master dictionary, or it can be stored as an item in any dictionary or data file. If a PROC is stored in a master dictionary, it can be executed directly by keying its name in as a command at the TCL level. If a PROC is stored elsewhere, it can be executed by using the RUNPROC command or it can be called by another PROC in a master dictionary or elsewhere.

The first line (attribute 1) of a PROC is always the code PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent lines contain PROC statements.

PROC statements consist of an optional numeric label, a one- or two-character command, and optional command arguments. PROCs can be created and modified using either the line editor, ED, or the screen editor, SE. A PROC can also be created from any statement in the TCL stacker by using the BUILD-PROC command.

The PROC Buffers

Note: All commands used in the following discussions are described in detail in Chapter 2 of this manual.

PROC utilizes two input and two output buffers:

- the primary input buffer
- the secondary input buffer
- the primary output buffer
- the secondary output buffer (sometimes referred to as the stack)

Operations specified within a PROC involve the movement of data from either of two input buffers (data storage areas) to either of two output buffers, thus forming the desired TCL statements. At any given time, one of the input buffers is specified as the active input buffer, while one of the output buffers is specified as the active output buffer. Buffers are selected as active via certain PROC commands such as IN, IP, S, SP, SS, STOFF, and STON. When moving data between the buffers, the source of transfer is the active input buffer. The destination of the transfer is the active output buffer. The PROC output buffers can then be passed to TCL for execution.

Primary Input Buffer

The primary input buffer contains the PROC name and any arguments, exactly as they were entered when the PROC was invoked. The contents of this buffer remain the same throughout the execution of the PROC unless explicitly modified by an Input (IH, IP, IT), Reset (RI), +, or - command.

Secondary Input Buffer

The secondary input buffer can contain variable data from one of several sources at any given time. It receives data from certain system software (for example, the spooler when a hold file is created). It can hold data input by the user in response to an IN, IP, or IS command. System message identifiers are also reported in this buffer as a result of executing the last TCL statement. The data in this buffer is temporary and is erased by subsequent IN or IS commands.

Primary Output Buffer

The primary output buffer is used to build one TCL statement that is eventually executed by a P, PH, PP, or PW command. After the statement in the primary output buffer is executed, PROC regains control at the statement immediately following the P command. If the statement being executed is itself a PROC, at the conclusion of that statement, control is returned to TCL, not to the current PROC.

The A command is used to copy data from the active input buffer to the active output buffer. The H command is used to copy data from the H command line in the PROC to the active output buffer. The primary output buffer is the active output buffer unless a STON command has been executed prior to the H command.

A carriage return is automatically maintained at the end of the primary output buffer. This means that a carriage return never needs to be explicitly added at the end of the TCL statement in the primary output buffer.

Secondary Output Buffer

The secondary output buffer is used for data needed by the TCL statement created by the PROC. For example, a BASIC program executed from a PROC may contain an INPUT statement; data in the secondary output buffer would be used by the INPUT statement to complete the program execution.

The secondary output buffer is sometimes referred to as the stack; the secondary output buffer is a FIFO (first-in, first-out) stack, which must be selected as the active output buffer in order to receive data. The STON command is used to select the secondary output buffer to be active (see the STON command listed alphabetically in Chapter 2).

In contrast to the primary output buffer, which contains a single line of data (a statement) to be passed to TCL, the secondary output buffer is used to store zero or more lines of data to satisfy terminal input requests by the processor invoked by the TCL statement being executed. (The word *line* here refers to a string of characters terminated by a carriage return.)

The characters can be all on one line or they can be separated from the next line by a < delimiter corresponding to a carriage return at the terminal. The < delimiter is placed in the secondary output buffer via the H command.

Similar to the primary output buffer, a carriage return is automatically maintained at the end of the secondary output buffer. This means that a carriage return never needs to be explicitly added after the last (or only) line of data in the secondary output buffer. However, a carriage return must be explicitly placed after each data line prior to the last one in the secondary output buffer.

Each request for terminal input (for example, each INPUT statement in a BASIC program) is satisfied with a line of data from the secondary output buffer. If more data than exists in the secondary output buffer is requested, data is acquired from the terminal from that point forward.

Parameters

A parameter is a string in an input or output buffer. It is identified either by being separated by blanks or by being enclosed in single quote marks ('). The first parameter in an input buffer starts with the first non-blank character in the buffer. If parameters are separated by blanks, one blank between parameters is sufficient; however, any number of blanks are allowed. If parameters are enclosed in single quotes, one single quote should precede the parameter and one single quote should follow the parameter. Multiple consecutive single quotes are assumed to delimit multiple (null) parameters.

If two consecutive parameters are enclosed in single quotes, blanks are not required between the trailing quote mark of one parameter and the leading quote mark of the next parameter. However, any number of blanks are allowed.

If a parameter is delimited by blanks, single quotes in the parameter are treated as ordinary characters (that is, they are not used to terminate the parameter).

If a parameter is enclosed in single quotes, blanks in the parameter are treated as ordinary characters (that is, they are not used to terminate the parameter).

The current parameter of an active buffer is the data from the current position of the pointer to the end of the parameter.

Surround Characters

A surround character is a character placed before and after the data being copied via the A command.

If data is copied from the input buffer to the output buffer via the A command and a surround character is not specified, a blank is used as the surround character. If a backslash [\] is used as the surround character, data is copied without any surrounding characters. For more information, see the A command listed alphabetically in Chapter 2.

If data is copied to the output buffer via an H command, there are no surrounding characters placed in the output buffer.

Pointers

Pointers are set to a specified location. Each buffer has a pointer which points to the current position of that buffer. These pointers are depicted in figures in this manual as arrows (🠉) above or below the buffers.

PROC commands such as the B, F, S(m), and Sn commands are used to change the position of the input pointers. The BO command is used to change the position of the output pointers.

Figure 1-1 illustrates a parameter being copied from the input buffer to the output buffer. The active buffers are the primary input buffer and the primary output buffer. Two parameters, ABC XYZ, are in the primary input buffer, with the current position of the input pointer at the first parameter (ABC). Parameter ABC is copied to the primary output buffer with blanks as surround characters via an A command. An H command is used to copy DEF to the primary output buffer. The secondary input and output buffers are not affected.

```
001 PQ
002 IHABC XYZ
003 A
004 HDEF
                  Primary            Primary
Statement         input buffer       output buffer

IHABC XYZ         ABC_XYZ
                  ↑
A                 ABC_XYZ            _ABC_
                      ↑                   ↑
HDEF              ABC_XYZ            _ABC_DEF
                      ↑                      ↑
```

Figure 1-1. Buffer Activity

Select-Lists and the Secondary Output Buffer

After a SELECT, SSELECT, QSELECT, or GET-LIST statement is executed, and items are present, PROC goes to the secondary output buffer to execute a TCL statement. (PROC does not gain control between these commands.) If the secondary output buffer is empty, PROC displays the TCL prompt and waits for the user to enter a command. PROC attempts to execute the user's response then continues to execute the rest of the PROC.

If, however, a null input (that is, an H< command) is placed in the secondary output buffer, PROC regains control after the select command. PROC can then test for the existence of a select-list (see the IF S command listed alphabetically in Chapter 2) before building the TCL statement that uses the select-list; see Figure 1-2.

After a SELECT, SSELECT, QSELECT, or GET-LIST statement is executed, and no items are present, PROC does not go to the secondary output buffer to execute the TCL statement. PROC continues to execute the rest of the PROC.

```
    TEST
001 PQ
002 HSELECT MD WITH 1 = "9"
003 STON
004 HSAVE-LIST ITEMS
005 P
006 OEnd of PROC

:TEST↵

[401] No items present.
End of PROC

:
```

```
   EXIST.TST

001 PQ
002 HGET-LIST
003 OENTER LIST-NAME+
004 IP?
005 A
006 STON
007 H<
008 P
009 IF #S XNo select-list

:SELECT  INVENTORY↵

8 items selected.

:SAVE-LIST  INVENTORY↵

'INVENTORY' saved - 1 frames used.

:EXIST.TST↵
ENTER LIST-NAME?INVENTORY↵

8 items selected.

:
```

Figure 1-2. Existence of a Select-List

Components of a PROC

The first line (attribute) of a PROC must contain the code PQ. This identifies the item as a PROC. The remaining lines in the PROC are either comment lines or valid PROC statements. There is no limit to the number of lines in a PROC. However, each line must contain only one PROC statement, and each statement must begin in position one of the line.

PROC statements consist of an optional numeric label, a one- or two-character command, and optional command arguments. PROCs can be created using either the line editor, ED, or the screen editor, SE.

One or more arguments may follow a PROC command in a PROC statement. Whether or not a blank is required to separate arguments depends on the specific command. Anywhere a single blank is required or allowed, multiple blanks may occur.

A comment is identified by either an asterisk (*) or C as the first character. The text following the asterisk (*) or the C can contain strings of any number of characters. This text is ignored by the PROC processor during execution of the PROC.

Any PROC statement or comment may begin with a statement label. Such a label serves to uniquely identify its associated PROC command for purposes of branching or looping within the PROC. Labels must be numeric. Only the first occurrence of a label is used as the destination of any control transfer. Duplicate labels are ignored. When a label is used, at least one blank must separate the label from the PROC command. For example:

Command	Explanation
`1 A`	Label 1
`10 GO 5`	Label 10 branch to label 5
`20 IF A = ABC GO 3`	Label 20 conditionally branch to label 3
`30 C This is a comment`	Label 30 represents a comment line. Processing continues with the next PROC line

When a PROC is executed, processing begins at the second line and continues until one of the following occurs:

- an X command is encountered
- a (...) command transfers control to another PROC
- an attempt is made to execute past the last line of the PROC

The PROC commands are listed by function at the end of this chapter so that new programmers can get acquainted with them. Chapter 2 presents a complete description of each command in alphabetical order as a reference for programmers.

PROC Execution

A PROC can be executed in the following ways:

- by entering its name at the TCL level
- from a BASIC program via an EXECUTE statement
- by the RUNPROC command
- by another PROC calling it
- by another PROC transferring control to it

The PROC processor both interprets and executes the PROC, so no compilation phase is required.

Once a PROC is invoked, it remains in control until it terminates. When the PROC temporarily relinquishes control to other system software such as Ultimate RECALL®, a BASIC program, or a programmer-supplied subroutine, it regains control when the executed software terminates.

If a TCL statement resides in the primary output buffer, TCL can regain temporary control when all desired data has been moved to the output buffers. Temporary control is passed to TCL via a P, PH, PP, or PW command. The TCL statement that resides in the primary output buffer is executed and the data in the secondary output buffer (if any) is used to feed data to the command, such as a BASIC program or a COPY command.

When the TCL statement has completed, control returns to the PROC to the statement immediately following the P command. At that time, both output buffers will be empty, both primary buffers will be active, and all data previously placed in the secondary output buffer will be cleared whether it was used or not.

Note: *When control is passed to TCL via a PX command, the PROC terminates after the TCL statement has been executed (see the P command listed alphabetically in Chapter 2).*

TCL only regains control when a PROC has not been called from another PROC and the PROC is terminated explicitly, or when an attempt is made to execute past the last line of the PROC.

PROC Execution Example

Figure 1-3 is a PROC stored as item LISTDICT in the PROCLIB file. The PROC passes data from a TCL statement to Ultimate RECALL. The actual LISTDICT PROC can be displayed to the terminal by entering the following at TCL:

```
CT  PROCLIB  LISTDICT
```

```
     LISTDICT
001 PQ
002 C Sort all attribute or synonym definitions in any
003 C dictionary and list them on the terminal or the
004 C lineprinter if LPTR specified.
005 H SORT DICT D/CODE A/AMC S/NAME S/AMC V/CONV
    V/CORR V/TYP V/MAX WITH D/CODE "A""X""U" BY
    A/AMC BY D/CODE BY *A0
006 F
007 IF #A H M/DICT
008 10 A
009 IF A G 10
```

Figure 1-3. LISTDICT PROC

The PROC in Figure 1-3 is interpreted and processed as follows:

1. The TCL statement invokes the PROC processor and places the TCL statement into the primary input buffer. Each word is considered to be a parameter in the buffer. The primary input pointer points to the first parameter.
2. Lines 2, 3, and 4 are comment lines that explain the PROC 's purpose and are not processed.
3. Line 5 is an H statement, which causes the following text to be placed into the primary output buffer:

   ```
   SORT DICT D/CODE A/AMC S/NAME S/AMC V/CONV V/CORR
   V/TYP V/MAX WITH D/CODE "A""X""U" BY A/AMC BY
   D/CODE BY *A0
   ```

   This text is a SORT statement that sorts the DICT items that have a value A, X, or U in the D/CODE attribute.
4. Line 6 is an F command, which moves the primary input pointer forward one parameter.
5. Line 7 checks for a second parameter; if there is not one, it moves the characters _M/DICT to the primary output buffer, which would cause the SORT to list the account's master dictionary.
6. Line 8 moves the current parameter from the primary input buffer to the primary output buffer. If there had been no parameter, nothing would have been moved.
7. Line 9 checks for another parameter. If there is one, PROC loops back to line 8 to copy the parameter. (For example, there could be a (P to indicate the report is to be sent to the spooler.)
8. There are no more lines, so the primary output buffer is executed. At the TCL level, the SORT statement is interpreted as a TCL statement that calls the Ultimate RECALL software for execution of the sort. This Ultimate RECALL statement causes the selected items to be sorted and the data from the specified file to be listed. (Refer to *The Ultimate RECALL and Ultimate UPDATE®User Guide* for details on the SORT command.)
9. At the conclusion of the sort, control is returned to the PROC. Because there are no further PROC statements, the PROC is terminated and control returns to TCL.

The user can execute this PROC from the SYSPROG account by entering the following at TCL:

```
LISTDICT  STAT-FILE
```

Figure 1-4 shows the execution of LISTDICT.

```
STAT-FILE : %UT
CODE   A
A/AMC   0
S/NAME   %UT
CONVERSIONS   A;(7*"100")/(12*"FRMSIZE"(TSTAT-FILE;X;;1))
TP   R
MAX   3

STAT-FILE : 5
CODE   A
A/AMC   0
S/NAME   AV/ITM
CONVERSIONS   MR0,Z
CORRELATIVES   F7;6;/
TP   R
MAX   6

STAT-FILE : 9
CODE   A
A/AMC   0
S/NAME   %UT
.
.
```

Figure 1-4. Execution of LISTDICT PROC

System Commands Associated With PROC

The following system commands described in this section are commands associated with PROC. Refer to the *Ultimate System Commands Guide* for more details on the commands.

BUILD-PROC command	builds a PROC from a TCL statement.
LISTPROCS command	sorts and lists all PROCs in a file.
RUNPROC command	executes a PROC from a specified file.

BASIC Statements That Interact With PROC

The following BASIC statements are statements that interact with PROC. Refer to the *Ultimate BASIC Language Reference Guide* for more details on the statements.

ABORT statement	terminates program execution. If the program was run from a PROC, the PROC is terminated as well.
CHAIN statement	terminates program execution and passes control to a specified TCL statement. If the program was run from a PROC, control is not returned to the PROC prior to executing the TCL statement.
CLEARDATA statement	all data previously placed on the stack and not yet used is removed, even when that data stack has been established by a PROC.
DATA statement	is used to store data for stacked input. If a PROC passes data to the program as an argument via the PROC's secondary output buffer, the input is stacked at the beginning and should be used as soon as possible since the first DATA statement overwrites any unprocessed stacked input.
EXECUTE statement	allows a BASIC program to execute any valid TCL statement and to use the results of the statement in later processing. If a BASIC program calls a PROC using an EXECUTE statement, control returns to the BASIC program when all PROC processing has finished.

INPUT statement	is used to request data. If both INPUT and DATA statements are used in a program, any INPUT statements intended to process input from an external source (such as a PROC or user input) should appear in the program prior to any DATA statements. The first DATA statement overwrites any unprocessed stacked input.
PROCREAD statement	allows programs executed from PROC to read values in the primary input buffer and store them in a variable.
PROCWRITE statement	allows programs executed from PROC to write to the primary input buffer.
PUT statement	places a system message into the output of a program. The message is also placed in the system message buffer and may be passed back to a PROC. If the program is invoked by a PROC, the buffer can be later inspected by the PROC using the IF E command.
STOP statement	terminates program execution. If the program was run from a PROC, the PROC is not terminated.

PROC Commands By Function

The following pages list the PROC commands by function. By reviewing the commands as grouped here, you can get an overview of PROC's capabilities.

Note: *Many commands appear in more than one category because they affect more than one thing.*

PROCs perform the following functions:

External Results Testing

IF E	tests for system message after executing a TCL statement
IF S	tests for select-list

Execution Control

*	comments
(...)	transfers control to specified PROC
[...]	executes a PROC as a subroutine
C	comments
G{O}	branches to specified PROC statement label
IF	tests specified parameter and conditionally executes the PROC command which follows on the same line
P	executes the TCL statement in the primary output buffer
PH	executes the TCL statement in the primary output buffer, suppresses terminal output associated with the command
PP	displays both output buffers, then executes the TCL statement in the primary output buffer
PW	displays both output buffers, then waits for terminal input before conditionally executing the TCL statement in the primary output buffer
PX	executes the TCL statement in the primary output buffer; control is transferred, upon completion of the TCL statement, the same as if an X command were executed next in the PROC

X	exits current PROC and transfers control to TCL, the calling PROC, or the BASIC program.

Selecting Buffers

IN	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IS	selects secondary input buffer and prompts for input which is then stored in the secondary input buffer
S(m)	selects the primary input buffer and positions pointer at column position m
SP	selects the primary input buffer
SS	selects the secondary input buffer
STOFF	selects the primary output buffer
STON	selects the secondary output buffer

Terminal Use

D	displays parameter stored in the active input buffer
IN	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IP	prompts for input which is then stored in the active input buffer
IS	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
O	displays text
T	terminal control command; outputs text, bell, or cursor positioning control codes

Working with Input Buffers

+n	adds n to the current parameter in the active input buffer
-n	subtracts n from the current parameter in the active input buffer
A	copies data from the active input buffer to the active output buffer
B	moves the pointer back one parameter in the active input buffer
D	displays parameter stored in the active input buffer
F	moves pointer forward one parameter in active input buffer
IF	tests specified parameter and conditionally executes the PROC command which follows on the same line
IH	copies data to the active input buffer
IN	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IP	prompts for input which is then stored in the active input buffer
IS	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IT	reads current tape label from attached tape unit and stores it in the primary input buffer
RI	resets (clears) both input buffers and selects the primary input buffer
S(m)	selects the primary input buffer and positions pointer at column position m
Sn	positions the input pointer at parameter n in the active input buffer

Working with Output Buffers

A	copies data from the active input buffer to the active output buffer
BO	moves the pointer back one parameter in the active output buffer
H	copies data to the active output buffer
P	executes the TCL statement in the primary output buffer
PH	executes the TCL statement in the primary output buffer, suppresses terminal output associated with the command
PP	displays both output buffers, then executes the TCL statement in the primary output buffer
PW	displays both output buffers, then waits for terminal input before conditionally executing the TCL statement in the primary output buffer
PX	executes the TCL statement in the primary output buffer; control is transferred, upon completion of the TCL statement, the same as if an X command were executed next in the PROC
RO	resets (clears) both output buffers and selects the primary output buffer

Notes

2 PROC Commands

In this chapter, each command is described separately in detail. The commands are presented in alphabetical order. Each command begins on a new page, and consists of one or more pages, as necessary.

The following information is provided for each command:

- a brief description of the command's function
- the exact syntax of the command
- a detailed description of the command's usage
- PROC examples, if any, with an explanation and sample output

The commands identified by symbols, such as the + command and the [...] command, are listed before the commands with alphabetical names.

In a PROC example, an arrow [↑] indicates the current location of the pointer in an active buffer and underscores [_] represent blanks.

A Summary of the PROC Commands

The valid PROC commands are as follows:

General Command Format	Command Description
*{text}	comments
+n	adds n to the current parameter in the active input buffer
-n	subtracts n from the current parameter in the active input buffer
(filename {item-ID}) {n}	transfers control to specified PROC
[filename {item-ID}] {n}	executes a PROC as a subroutine
A{c}{p}{,m}	copies data from the active input buffer to the active output buffer by parameter
A{c}({n}{,m})	copies data from the active input buffer to the active output buffer by character
B	moves the pointer back one parameter in the active input buffer
BO	moves the pointer back one parameter in the active output buffer
C{text}	comments
D{p}{,n}{+}	displays parameter stored in the active input buffer
F	moves pointer forward one parameter in the active input buffer

G{O}n	branches to specified PROC statement label
G{O}A{m}	branches to a PROC statement label referred to by the specified parameter in the active input buffer
H{data}{<}	copies data to the active output buffer
IF {#}a-stmt proc-stmt	tests for existence of parameter
IF{N} a-stmt op string proc-stmt	tests for parameter relations
IF a-stmt op (pattern) proc-stmt	tests for pattern matching on parameter
IF{N} {#}E {op string} proc-stmt	tests for system message after executing a TCL statement
IF {#}S proc-stmt	tests for select-list
IH{data}	copies data to the active input buffer
IN{r}	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IP{r}	prompts for input which is then stored in the active input buffer
IS{r}	selects the secondary input buffer and prompts for input which is then stored in the secondary input buffer
IT	reads current tape label from attached tape unit and stores it in the primary input buffer
O{text}{+}	displays text

P	executes the TCL statement in the primary output buffer
PH	executes the TCL statement in the primary output buffer, suppresses terminal output associated with the command
PP	displays both output buffers, then executes the TCL statement in the primary output buffer
PW	displays both output buffers, then waits for terminal input before conditionally executing the TCL statement in the primary output buffer
PX	executes the TCL statement in the primary output buffer; control is transferred, upon completion of the TCL statement, the same as if an X command were executed next in the PROC
RI{p}	resets (clears) both input buffers and selects the primary input buffer
RO	resets (clears) both output buffers and selects the primary output buffer
S(m)	selects primary input buffer and positions pointer at column position m
Sn	positions input pointer at parameter n in the active input buffer
SP	selects the primary input buffer
SS	selects the secondary input buffer
STOFF	selects the primary output buffer
STON	selects the secondary output buffer

T {f{,f{,f{...}}}}{,}	terminal control command; outputs text, bell, or cursor positioning control codes
X{text}{+}	displays text, exits current PROC, and transfers control to TCL, the calling PROC, or the BASIC program

* – Comment

The * command allows comments within the body of a PROC.

Syntax

***{text}**

text	Specifies the comment; it can contain any characters, including blanks.

Description

The text following the * command can contain strings of any number of characters. This text is ignored by the PROC processor during execution of the PROC.

The C command can also be used to place comments in a PROC. (See the C command listed alphabetically in this chapter.)

```
001 PQ
002 * This is an example of a comment line
.
.
```

+ and - – Arithmetic Functions

The + command adds the specified number to a decimal parameter in the active input buffer. The - command subtracts the specified number from the decimal parameter in the active input buffer.

Syntax

+n
-n

n	The specified number (decimal integers); must be between $\pm(2^{31}-1)$.

Description

Prior to executing a + or - command, the input pointer must be pointing to the first digit of the decimal parameter. If the decimal parameter is negative, the input pointer must be pointing to the - (negative) sign.

The + and - commands do not change the location of the input pointer. If the input pointer is currently at the end of the buffer, the + and - commands have no effect.

Note: *When initializing a decimal parameter, leading zeroes should be used to make sure the initial value contains the same number of digits as all anticipated values created by subsequent + and - commands; otherwise, other parameters may be overwritten. Allow for the - sign for negative numbers.*

No arithmetic overflow is reported. Values used by arithmetic operations are integers between $\pm(2^{31}-1)$.

If a parameter begins with non-numeric characters, it will be replaced by argument n. For example, assume the input buffer contains XYZ and the pointer is pointing to X; +1 will yield 1YZ.

```
PQ
*Reset input buffers to null
RI
*Copy data into primary input buffer
IHABC 001 XYZ
*Move input pointer forward
F
*Add 99 to current parameter
+99
```

```
Statement                Primary input buffer

IHABC 001 XYZ            ABC_001_XYZ
                         ↑
F                        ABC_001_XYZ
                             ↑
+99                      ABC_100_XYZ
                             ↑
```

```
PQ
*Reset input buffers to null
RI
*Copy data into primary input buffer
IHABC 001 XYZ
*Move input pointer forward
F
*Subtract 99 from current parameter
-99
```

```
Statement                Primary input buffer

IHABC 001 XYZ            ABC_001_XYZ
                             ↑
F                        ABC_001_XYZ
                             ↑
-99                      ABC_-98_XYZ
                             ↑
```

(...) – Transfer Control

The (...) command transfers control to another PROC (specified within the parentheses), with no provision for returning to the current PROC.

Syntax

(filename {item-ID}) {n}

filename	Specifies the file in which the PROC to be executed is stored.
item-ID	Specifies the name of the PROC to be executed. If omitted, the data starting at the current position of the input pointer to the end of the parameter is used as the item-ID.
n	Specifies a label in the PROC where control should be transferred. If omitted, control is transferred to the second line of the PROC (it is assumed that the first line contains the PQ code).

Description

The (...) command transfers control from the current PROC to the specified PROC. Once control has been transferred to the specified PROC, control does not return to the calling PROC.

The specified PROC can exist in any dictionary or data file. This allows the storage of PROCs outside of the master dictionary.

When a (...) command is executed, the PROC buffers remain unchanged.

```
      LINK
001 PQ
002 OEnter the item-ID of the PROC to be executed +
003 C User specifies the PROC to execute by entering its item-ID
004 IN?
005 C Execution is transferred to the user specified PROC in the
006 C PROCLIB file
007 (PROCLIB)
```

Output to terminal

```
:LINK↵
Enter the item-ID of the PROC to be executed? LISTUSERS↵

 CH#. PCBFID NAME........... TIME.. DATE.... LOCATION

 000 000200 P                08:00  01/01/92 Paul's Office
 002 000240 CM               09:10  01/01/92 Shop Floor
 003 000260 LC               07:30  01/01/92 Larry's Office
 004 000280 JP               10:14  01/01/92 East Wing
*006 0002C0 SAL              08:35  01/01/92 Sally's Office
 010 000340 JET              09:00  01/01/92 Lobby

:
```

[...] – Subroutine Call

The [...] command can be used to perform a call to an internal subroutine or a call to an external subroutine.

Syntax

[] {n}
[filename {item-ID}] {n}

filename	Specifies the file in which the PROC is stored.
item-ID	Specifies the name of the PROC to be executed. If omitted, the data starting at the current position of the input pointer to the end of the parameter is used as the item-ID.
n	Specifies a label in the PROC where control should be transferred. If omitted, control is transferred to the second line of the PROC (it is assumed that the first line contains the PQ code).

Description

The first form of the [...] command performs a call to an internal subroutine. The second form of the [...] command performs a call to an external subroutine.

Both forms of the command store the location of the next PROC command in the PROC subroutine's stack and then transfers control to the specified location or PROC. Execution of PROC commands continues from that point until an X command or the end of the PROC is encountered. Once the call finishes, control returns to the next line following the call in the calling PROC.

In both internal and external PROC calls, neither the input nor output buffers are affected by the call itself or a subsequent return.

The following example shows an internal subroutine call.

```
    SUB.EXAMPLE
PQ
C Control transfers to label 3
[] 3
C Control continues here after return
OFIRST
3 OSECOND
```

Output to terminal

```
:SUB.EXAMPLE↵

SECOND
FIRST
SECOND

:
```

The next example shows an external subroutine call using a menu. If the user chooses option one or two, the current PROC transfers control to the external PROC. If the user chooses option three, the PROC is terminated.

```
    GO.EXAMPLE
PQ
C This PROC shows the use of menus
C with the [] command
C Display menu
10 OMain Menu
O1. List users on the system
O2. Enter new customer
O3. Exit
O
OEnter option+
S1
IP?
IF A = 1 [MD LISTU]
IF A = 2 [PROCLIB OPT2]
IF A = 3 XGOOD BYE
OPress return to redisplay main menu
IP
G 10
```

Output to terminal

```
:GO.EXAMPLE↲

Main Menu
1. List users on the system
2. Enter new customer
3. Exit

Enter option?1↲

 CH#. PCBFID NAME......... TIME.. DATE.... LOCATION

 000  000200 P             08:00  01/01/92 Paul
 002  000240 CM            09:10  01/01/92 Shop Floor
*006  0002C0 SAL           08:35  01/01/92 Sally
 010  000340 JET           09:00  01/01/92 Lobby

Press return to redisplay main menu:
```

A – Copy Data

The A command copies data from the active input buffer to the active output buffer.

Syntax

```
A{c}{p}{,m}
A{c}({n}{,m})
```

c Specifies a surround character to be placed before and after the copied data in the primary output buffer. Surround characters are not used in the secondary output buffer. If argument **c** is omitted when the primary output buffer is active, a blank is used as the surround character. If argument **c** is a backslash [\], data is copied without any surrounding characters. Argument **c** can be any non-numeric character except a left parenthesis [(] or a comma [,]. However, if the second form of the command is used, argument **c** can be a comma.

Note: *Surround characters are only placed around data in the primary output buffer, never around data in the secondary output buffer. Therefore, if the secondary output buffer is the active buffer, argument **c** has no effect.*

p Specifies the parameter to be copied. The input pointer is moved to the parameter. If argument **p** is omitted, copying begins at the current position of the input pointer.

n Selects the primary input buffer as the active input buffer and specifies the starting column position in that buffer to be copied; the primary input buffer remains active after the command is executed. If argument **n** is omitted, copying begins at the current position of the input pointer in the active input buffer.

m	Specifies the number of characters in the active input buffer to be copied (up to the end of data in the active input buffer) and parameter boundaries are ignored. If argument **m** is omitted, data is copied up to the end of the current parameter.

Description

Data can be copied either by parameter or by columns; this is determined by arguments **p**, **n**, and **m**.

If a parameter in an input buffer has embedded semicolons and that parameter is to be copied to the primary output buffer, the A command treats the parameter in a special manner. Each string that is delimited by a semicolon is treated as a sub-parameter. Each sub-parameter is copied to the output buffer enclosed by:

- the surround character, if specified
- by blanks if no surround character is specified
- by nothing if the specified surround character is a backslash

The effect is as if multiple A commands were executed. The semicolons are not copied.

If the parameter is enclosed in single quote marks or is being copied into the secondary output buffer, embedded semicolons are treated as ordinary characters.

A parameter that contains embedded semicolons is treated as a single parameter for purposes of positioning with the argument **p** prior to starting a copy.

At the conclusion of each A command, the input pointer ends up pointing to the next character after the last one copied. If two A commands are executed with no intervening input pointer positioning commands, and the second A command contains no specification for where the copy is to start (that is, neither arguments **p** nor **n** is specified), copying starts at the input pointer position resulting from the first A command. If the input pointer is at the end of the buffer and no starting position is specified, no operation occurs.

Blanks used to separate parameters are never copied to the output buffer. Blanks embedded within a parameter that is delimited by single quotes are copied to the output buffer. Single quotes used to enclose parameters are always copied to the output buffer.

If the primary output buffer is active and argument **c** is not specified, a blank is used as the surround character. If a backslash [\] is used as the surround character, data is copied to the primary output buffer without any surrounding characters. The primary output pointer points to the end of the last parameter if there is not a surround character present. If a surround character is present, the primary output pointer points to the surround character.

Surround characters are not used if the secondary output buffer is active. Argument **c**, if specified, has no effect.

If argument **m** is used to copy a specific number of characters, subsequent parameter boundaries may be different than if the data had been copied on a parameter basis. For example, assume the active input buffer contains the following:

```
'A QUICK BROWN FOX' JUMPED OVER THE LOG
```

Parameter number 1 is 'A QUICK BROWN FOX'; parameter 2 is JUMPED; parameter 3 is OVER, and so on. After the following command, the input pointer is pointing at the blank preceding BROWN:

```
A(,8)
```

A subsequent A command that does not specify any arguments will copy BROWN as the next parameter.

The following example shows the use of the A command with and without arguments.

```
PQ
IH'ABC DEF' XYZ
A
A\,2
A
```

```
                    Primary           Primary
Statement           input buffer      output buffer

IH'ABC DEF' XYZ     'ABC_DEF'_XYZ
                    ↑

A                   'ABC_DEF'_XYZ     _'ABC_DEF'_
                             ↑                    ↑

A\,2                'ABC_DEF'_XYZ     _'ABC_DEF'_ _ X
                               ↑                     ↑

A                   'ABC_DEF'_XYZ     _'ABC_DEF'_ _ X_ YZ_
                                 ↑                        ↑
```

B – Input Pointer Back

The B command moves the input pointer back one parameter in the active input buffer.

Syntax

B

Description

After execution of the B command, parameters between the new position of the input pointer and the end of the buffer still exist. If the input pointer is at the beginning of the active input buffer, the B command has no effect.

```
001 PQ
002 RI
003 IHABC 123 DEF
004 C Position input pointer to parameter DEF
005 S3
006 C Move input pointer back
007 B
```

```
Statement               Primary input buffer

IHABC 123 DEF           ABC_123_DEF
                        ↑

S3                      ABC_123_DEF
                                ↑

B                       ABC_123_DEF
                            ↑
```

BO – Output Pointer Back

The BO command moves the output pointer back one parameter in the active output buffer.

Syntax

BO

Description

After execution of a BO command, any parameters beyond the new position of the output pointer are lost. If the output pointer is at the beginning of the buffer, the BO command has no effect.

```
PQ
C Reset output buffers to null
RO
C Copy data into the primary output buffer
HXXX YYY
*Move output pointer back
BO
```

```
Statement                Primary output buffer

HXXX YYY                 XXX_YYY
                                ↑

BO                       XXX
                            ↑
```

C – Comment

The C command allows comments within the body of a PROC.

Syntax

C{text}

text	Specifies the comment; it can contain any characters, including blanks.

Description

The text following the C command can contain strings of any number of characters. This text is ignored by the PROC processor during execution of the PROC.

The * command can also be used to place comments in a PROC. (See the * command listed alphabetically in this chapter.)

```
001 PQ
002 C This is an example of a comment line
.
.
```

D – Display Data

The D command displays data from an input buffer to the terminal.

Syntax

```
D{p}{,m}{+}
D({n}{,m}){+}
```

p	Specifies the parameter to display. If argument **p** is omitted, the display begins at the current position of the input pointer. If argument **p** is zero, the entire contents of the active input buffer are displayed, even if argument **m** is specified.
n	Specifies the starting column position in the primary input buffer to be displayed. If argument **n** is omitted, copying begins at the current position of the input pointer.
m	Specifies the number of characters to be displayed (up to the end of data in the input buffer) and parameter boundaries are ignored. If argument **m** is omitted, data is displayed up to the end of the current parameter.
+	Suppresses the carriage return so that the cursor does not move to the next line after the data is displayed.

Description

Data can be displayed either by parameter or by columns; this is determined by arguments **p**, **n**, and **m**.

If the input pointer points at the end of the buffer and no starting position is specified, no characters are displayed; however, a carriage return and linefeed are sent to the terminal.

If a parameter in an input buffer has embedded semicolons and that parameter is to be displayed, the parameter is treated as a single parameter for purposes of display.

The D command has no effect on the active input buffer or its pointer; whichever buffer was the active input buffer before the command remains the active input buffer after the command.

```
PQ
RI
C Copy data into primary input buffer
IH'ABCD' EFG
C Display first two characters
D,2
C Display current parameter
D
C Display buffer contents
D0
```

Statement	Primary input buffer	Terminal
`IH'ABCD' EFG`	`'ABCD'_EFG`	
	↑	
`D,2`		`'A`
`D`		`'ABCD'`
`D0`		`'ABCD'_EFG`

F – Input Pointer Forward

The F command moves the input pointer forward one parameter in the active input buffer.

Syntax

F

Description

The F command has no effect on the contents of the active input buffer. If the input pointer is already at the end of the buffer, the F command has no effect.

```
PQ
C Reset input buffers to null
RI
C Copy data into the primary input buffer
IHABC 123
C Move input pointer forward
F
```

```
Statement                Primary input buffer

IHABC 123                ABC_123
                         ↑

F                        ABC_123
                             ↑
```

G{O} – Transfer Control

The G{O} command unconditionally transfers control (that is, branches) to a specified label in the PROC or to a label specified by a parameter in the active input buffer.

Syntax

G{O} n
G{O} A{m}

n	Specifies the numeric label of the PROC command to which control is to be transferred.
A	Specifies a parameter in the active input buffer.
m	Specifies the ordinal number of the parameter; if m is omitted, the current parameter is used.

Description

The G{O} n form of the command transfers control to PROC label n.

The G{O} A{m} form of the command allows for variable branching. It causes a branch to the label referred to by the specified data in the active input buffer.

If the referenced label does not exist, the following error message is displayed and the PROC is terminated:

```
[268] The destination of the PROC GO statement:
    GO n, cannot be found
```

If several PROC commands begin with the same label, the GO command transfers control to the first PROC command which begins with the specified label (scanning from the top).

Note: *For information to conditionally transfer control within a PROC, see the IF command listed alphabetically in this chapter.*

```
PQ
10 C Menu is displayed
O
O 1. List vendors
O 2. Enter new vendors
O 3. End program
O
O    Enter job #+
C Get response
IP
C If the response is greater than 3 or less than
C one, then branch to label 10
IFN A > 3 GO 10
IFN A < 1 GO 10
C Branch based on response
GO A
C Label 1; response is 1
1 HSORT VENDORS
C Execute output buffer and terminate
PX
C Label 2; response is 2
2 HRUN PROGRAMS VENDOR.UPD
PX
C Label 3; response is 3
3 XBYE
```

Output to terminal

```
1. List vendors
2. Enter new vendors
3. End program

   Enter job #:3↵
   BYE
:
```

H – Copy Data to Output Buffer

The H command copies specified data to the active output buffer.

Syntax

H{data}{<}

data	Specifies the data to be copied to the active output buffer; it can contain any characters, including blanks. If omitted, a null string is used.
<	Inserts a carriage return when the active output buffer is the secondary output buffer. < as the last character in an H command marks the end of one data line in the secondary output buffer and the beginning of another; it is not required after the last data line. When the primary output buffer is active, < at the end of an H command has no special meaning, and is considered part of the data.

Note: The line continuation character (<<) is no longer necessary.

Description

The H command causes the data (including any blanks) which immediately follows the H command to be placed in the active output buffer at the current position of the output pointer. Thus, a blank after the H command would place a blank into the output buffer. The primary output buffer is active unless a STON command is executed prior to the H command.

When a data line is copied to the secondary output buffer, a carriage return (<) must be explicitly placed in the secondary output buffer or the next data line will be appended to the current line. For example, assume the secondary output buffer contains ABC. The following H statement:

```
HXYZ
```

causes XYZ to be appended to ABC in the secondary output buffer:

```
ABCXYZ
```

The following H statement will place a carriage return only in the secondary output buffer:

```
H<
```

Note: *After a SELECT, SSELECT, QSELECT, or GET-LIST statement is executed, and no items are present, PROC does not go to the secondary output buffer to execute the TCL statement. PROC continues to execute the rest of the PROC. For an example, see the STON command listed alphabetically in this chapter.*

```
PQ
C Select secondary input buffer
SS
C Copy data into input buffer
IHXYZ ABC
C Select secondary output buffer
STON
C Copy parameter XYZ to output buffer
A
C Place a blank, the text DE and a carriage return
C into the secondary output buffer
H DE<
```

```
                    Secondary           Secondary
Statement           input buffer        output buffer

IHXYZ ABC           XYZ_ABC
                    ↑

A                                       _XYZ_
                                            ↑

H DE<                                   XYZ_DE<CR>
                                                  ↑
```

IF – Conditional Branch

The IF command transfers control to another statement within a PROC, based on a specified condition.

Syntax

IF {#}a-cmd proc-stmt	existence test
IF{N} a-cmd op string proc-stmt	relational test
IF a-cmd op (pattern) proc-stmt	pattern match test
IF{N} {#}E {op string} proc-stmt	system message test
IF {#}S proc-stmt	select-list test

Description

The IF command provides the following tests:

• existence test	the existence of a specified parameter in the active input buffer
• relational test	the value of a parameter in the active input buffer
• pattern match test	the pattern element of a parameter in the active input buffer
• system message test	a previous TCL system message condition
• select-list test	the existence of a select-list

For any of the IF forms, the PROC statement that is conditionally executed can in turn be another IF statement; that is, IF statements can be nested. For example, the following statement transfers control to label 99 if the current parameter consists of two numerals in the range 10 through 19 (inclusive):

```
IF A = (2N) IF A > 09 IF A < 20 GO 99
```

The programmer can visualize nested IF statements as though implied AND operators were placed between them.

The tests provided by the IF command are described in detail on the following pages.

Table 2-1 lists the relational operators that can be used with IF statements.

Table 2-1. Relational Operators

Operator Symbol	Operation
=	tests for equal
#	tests for not equal
<	tests for less than
>	tests for greater than
[	tests for less than or equal to
]	tests for greater than or equal to

Existence Test

The existence test form of the IF command tests for the existence of a specified parameter in the active input buffer. If the parameter exists, control passes to the PROC statement specified. If the # option is used, the test is reversed; that is, if a parameter does not exist, control is transferred to the PROC statement.

Syntax

IF {#}a-cmd proc-stmt

#	(Not) specifies that the condition is true if the parameter indicated by the a-cmd does not exist in the active input buffer. If omitted, the condition is true if the parameter does exist in the active input buffer.
a-cmd	Valid A command (of the form A{p}) specifies the parameter to be tested. (Other forms of the A command are not meaningful.)
proc-stmt	Any valid PROC statement.

Description

When an A command is used in a test condition for an IF command, the parameter is not copied. Instead, the A command is used to specify which parameter in the input buffer is to be tested. Prior to the test, the input pointer is positioned to the beginning of the parameter specified by the A command.

A parameter exists in the active input buffer if the input pointer points to a character. Note that in positioning the input pointer, intervening spaces will be skipped over.

A parameter does not exist if the input pointer points to the end of the input buffer.

```
PQ
C Reset input buffers to null
RI
C Copy data into input buffer
IHABC AAA XYZ
C If parameter exists control is transferred to
C label 27
IF A2 GO 27
OA does not exist
X
C Label 27; display message on terminal
27 OA exists
C If parameter does not exist, the PROC terminates
IF #A4 XEND
C If parameter does exist, the message is displayed
OA4 exists
```

```
Statement          Primary input buffer       Terminal

IHABC AAA XYZ      ABC_AAA_XYZ
                   ↑

IF A2 GO 27        ABC_AAA_XYZ                A exists
                       ↑

IF #A4 XEND        ABC_AAA_XYZ                END
                              ↑
```

Relational Test

The relational form of the IF command compares data in the active input buffer with a specified string.

Syntax

IF{N} a-cmd op string proc-stmt

N	Specifies that a numeric comparison is to be made. If omitted, a string comparison is made.
a-cmd	Valid A command (of the form A{p}{,m}) specifies the parameter to be compared. (The Ac and A{c}({n}{,m}) formats are not meaningful.)
op	Specifies the relation to compare for; the operator can be any of the relational operators listed in Table 2-1.
string	Specifies the literal string of characters or the number to which the specified parameter is to be compared.
proc-stmt	Any valid PROC statement.

Description

To resolve a string relational condition, character pairs (one from the selected parameter and one from the literal string) are compared one at a time from leftmost characters to rightmost.

If no unequal character pairs are found, the strings are considered to be equal. If an unequal pair of characters is found, the characters are ranked according to their numeric ASCII code equivalents (refer to Appendix A, List of ASCII Codes). The character string contributing the higher numeric ASCII code equivalent is considered to be greater than the other string. For example, string AAB is considered to be greater than AAAA, and 02 is considered greater than 005.

If the selected parameter and the literal string are not the same length, but the shorter of the two is otherwise identical to the beginning of the longer one, then the longer string is considered greater than the shorter string. For example, the string WXYZ is considered to be greater than the string WXY.

Numeric comparisons are in the range -2^{48} to $+2^{48}-1$. A leading + or - sign is handled, but decimal points are not. Numbers are terminated at the first non-numeric character or at the end of the parameter. For example, +35AD will compare equal to 035BC.

To resolve a numeric relational condition (IFN form), the integer values of the selected parameter and the literal string are compared. For example, 102 is considered numerically less than 1005, whereas 102 would be considered greater than 1005 in a string comparison. Non-numeric strings evaluate numerically to zero.

```
PQ
RI
IHABC XYZ 012
C If parameter is equal to ABC, control is
C transferred to label 3
IF A = ABC GO 3
C If parameter is not equal to ABC, message is
C displayed and PROC is terminated
ONot equal
X
C Label 3; display message
3 OEqual
C If parameter A3 is equal to 12, control is
C transferred to label 100
IFN A3 = 12 GO 100
C If parameter A3 is not equal to 12, message is
C displayed and PROC is terminated
ONot equal
X
C Label 100; display message
100 OEqual
```

```
Statement            Primary input buffer      Terminal

IHABC XYZ 012        ABC_XYZ_012
                     ↑

IF A = ABC GO 3      ABC_XYZ_012               Equal
                     ↑

IFN A3 = 12 GO 100   ABC_XYZ_012               Equal
                             ↑
```

Pattern Match Test

The pattern matching form of the IF command compares a parameter in an input buffer with a specified pattern.

Syntax

IF a-cmd op (pattern) proc-stmt

a-cmd	Valid A command (of the form A{p}{,m}) specifies the parameter to be tested. (The Ac and A{c}({n}{,m}) formats are not meaningful.)
op	Specifies the relation to compare for. Although the operator can be any of the relational operators listed in Table 2-1, only = and # are meaningful.
(pattern)	Pattern used to test a parameter for a specified combination of numeric characters, alphabetic characters, alphanumeric characters, or literals; see Table 2-2.
proc-stmt	Any valid PROC statement.

Description

A pattern is composed of one or more pattern elements concatenated together. If a pattern is composed of the single element 0A or 0N, the test is true only if all the characters in the parameter conform to the character type. For example, the following statement outputs the message -OK- if the characters of the current parameter are all alphabetic characters:

```
IF A = (0A) O-OK-
```

If another pattern follows a 0A or 0N pattern, that pattern is used in the match as soon as a character is encountered that does not match the 0A or 0N pattern.

Table 2-2. Patterns

Pattern Element	Explanation
nN	tests for n numeric characters.
nA	tests for n alphabetic characters.
nX	tests for n characters of any type.
string 'string' "string"	tests for literal string of characters; delimiter is not required, but can be single or double quotes. Use of quote marks is useful in cases where the literal would otherwise look like a pattern element. If quotes are used, embedded blanks will not work because pattern matches stop on the first blank encountered. If quotes are not used, blanks will work.

If the integer value used in the nX form of the IF command is 0 (zero), the command will evaluate any number of characters of any type until encountering a blank or the end of the input buffer; if a pattern, other than 0A, 0N, or 0X, is appended to 0X, the match returns false. For example, the following statement outputs CORRECT if the characters of the current parameter are one character of any type followed by a - (hyphen) and then followed by any number of characters of any type:

```
IF A = (1X-0X) O CORRECT
```

However, if the 1X in the above statement is replaced by a 0X, it will not output CORRECT because the command will evaluate the current parameter as one string (that is, it will evaluate any number of characters of any type until encountering a blank or a right parenthesis) and then go back to the current parameter and look for a - (hyphen). However, the - (hyphen) will not be found because it has already been evaluated by the first 0X.

The following example outputs to the terminal EQUAL or NOT EQUAL based on the pattern match.

```
PQ
RI
IHABC 10/09/91
C If the first parameter is three alpha characters,
C then branch to label 7
IF A = (3A) GO 7
C If the first parameter is not equal to three
C alpha characters, a message is displayed and PROC
C is terminated
ONot equal
X
C Label 7; display message to terminal
7 OEqual
C If the second parameter is in the specified form,
C then branch to label 5
IF A2 = (2N/2N/2N) GO 5
C If the second parameter is not in the specified
C form, a message is displayed and PROC is terminated
ONot equal
X
5 C Label 5; display message
OEqual
```

```
Statement              Primary input buffer       Terminal

IHABC 10/09/91         ABC_10/09/91
                       ↑
IF A = (3A) GO 7       ABC_10/09/91               Equal
                       ↑
IF A2 = (2N/2N/2N) G 5
                       ABC_10/09/91               Equal
                           ↑
```

The next example outputs to the terminal OK if the user inputs a string in the pattern of four characters, followed by a hyphen, followed by 0 (zero) or more characters.

```
     STRING.EXAMPLE
PQ
O Enter a string to test
C If the string entered starts with four characters
C followed by a hyphen, then branch to label 10
IP?
IF A = (4X'-'0X) GO 10
C If the string is not in the specified form,
C message is displayed and PROC is terminated
O DOESN'T WORK
X
10 C Label 10; display OK to terminal
OOK
```

Output to terminal

```
:STRING.EXAMPLE↵

Enter a string to test?ABCD-EF↵
OK

:
```

System Message Test

The system message testing form of the IF command is used to test for system messages generated by a preceding PROC-generated TCL statement.

Syntax

IF{N} E op string proc-stmt

N	(IFN) specifies that a numeric comparison is to be made. If omitted, a string comparison is made.
op	Specifies the relation to compare for; the operator can be any of the relational operators listed in Table 2-1.
string	System message identifier; that is, item-ID of a system message in the ERRMSG file.
proc-stmt	Any valid PROC command.

Description

The IF E command allows a PROC to test for a system message identifier returned by a PROC-generated TCL statement. This command is relevant only after any form of the P command except for the PX form.

Note: *When control is passed to TCL via a PX command, the PROC terminates after the TCL statement has been executed (see the P command listed alphabetically in this chapter).*

Any message generated by processing the TCL statement is reported by placing the identifier of that system message in the secondary input buffer. Since the IF E command uses the secondary input buffer, it is valid until a command using the secondary input buffer is executed.

The command tests for the identifier of the system message.

The following example shows a numeric comparison. If a message identifier in the range of 92 through 98 has been encountered, control transfers to TCL and the text is printed.

```
015 IFN E > 91 IFN E < 99 X TAPE ERROR!
```

If system message number 82 has been encountered in the following command, control transfers to TCL and the text NO ACCESS is printed (user has insufficient level).

```
IF E = 82 X NO ACCESS
```

Select-List Test

The select-list form of the IF command tests for the existence of a select-list generated by a preceding SELECT, SSELECT, QSELECT or GET-LIST statement. This test can be used to prevent processing a non-existent select-list.

Syntax

IF {#}S proc-stmt

#	Specifies that the condition is true if a select-list is not available.
proc-stmt	Any valid PROC command.

Description

If a result such as NO ITEMS PRESENT or ITEM NOT ON FILE has occurred, the select-list will not exist, and the IF S command can be used to test for this condition.

Certain system commands create select-lists; some of these select-list system commands are SELECT, SSELECT, QSELECT and GET-LIST. (For more information about select-lists, see the specific command in the *Ultimate System Commands Guide* or the *Ultimate RECALL and Ultimate UPDATE User Guide*.)

A PROC can be written such that:

- it expects the user to execute one of the select-list system commands prior to executing the PROC
- it can be called from several other PROCs, some of which that execute select-list system commands prior to the call and some of which may not
- it executes a select-list system command with the secondary output buffer containing only a carriage return (that is, a H< statement)

The PROC can then use the IF S command to determine if anything was selected before continuing execution.

In the following example, the PROCs TEST1 and TEST2 operate identically if the GET-LIST statement executes without an error (that is, if the list exists on file). However, TEST2 continues with PROC execution even if the list is not on file. An IF S test would be meaningless following the P command in TEST2, since no select-list will exist at that point regardless of whether the GET-LIST was successful. TEST1, on the other hand, has a null line in the secondary output buffer when the GET-LIST executes; therefore, control is returned to the PROC, which can test to see if it successfully retrieved a select-list. If the list is not on file, the PROC requests the user to enter the list name and tries again.

```
    TEST1                               TEST2

001 PQ                              001 PQ
002 10 HGET-LIST                    002 HGET-LIST
003 OEnter list-name+               003 OEnter list-name+
004 IP?                             004 IP?
005 A                               005 A
006 STON                            006 STON
007 H<                              007 HLIST INVENTORY LPTR
008 PH                              008 P
009 IF S G 20                       .
010 OILLEGAL LIST-NAME.             .
011 OPLEASE RE-ENTER
012 G 10
013 20 HLIST INVENTORY LPTR
014 P
```

IH – Copy Data to Input Buffer

The IH command copies specified data to the active input buffer.

Syntax

IH{data}

data	Specifies the data to be copied to the active input buffer; it can contain any characters, including blanks. If omitted, no operation is performed.

Description

The IH command causes the specified data to replace the current parameter in the active input buffer.

Note: *The current parameter of an active buffer is the data from the current position of the pointer to the end of the parameter. If the pointer does not point at the first character of data delimited by blanks, the characters prior to the pointer are not copied.*

The input buffer pointer location is not changed. If the specified data is omitted, no operation is performed.

```
PQ
RI
C AAA BBB CCC is copied into the input buffer
IHAAA BBB CCC
C Move pointer to BBB.
F
C XX YY is copied into the input buffer
C and replaces BBB.
IHXX YY
```

```
Statement               Primary input buffer

IHAAA BBB CCC           AAA_BBB_CCC
                        ↑

F                       AAA_BBB_CCC
                            ↑

IHXX YY                 AAA_XX_YY_CCC
                            ↑
```

IN – Terminal Input to Secondary Input Buffer

The IN command selects the secondary input buffer and accepts data input from the terminal and puts it into the input buffer.

Syntax

IN{r}

r	Specifies a prompt character to be displayed at the terminal for operator input; it can be any character, including a blank. The prompt character remains in effect until another IN, IP, or IS command with a different prompt character is executed or a () (transfer control) or [] (subroutine call) is executed. If omitted, the previously specified prompt character, if any, is used. If no prompt character has been specified previously, a colon (:) is used as the prompt character.

Description

The IN command replaces all data in the secondary input buffer with the new data input from the terminal. The input pointer location is left pointing at the beginning of the data that was input.

Data input by this command can then be copied to the active output buffer by an A command. (See the A command listed alphabetically in this chapter.)

The IS command can also be used to accept data input from the terminal. The IS command is a synonym for the IN command.

```
     IN.EXAMPLE
PQ
OInput a letter of the alphabet+
C Data is accepted from the terminal and put in the
C secondary input buffer
IN=
C If input is an alpha character, control transfers
C to label 9
IF A = (1A) GO 9
XIllegal Response
C Label 9; message is displayed
9 OYou entered correctly
```

Output to terminal

```
:IN.EXAMPLE↵

Input a letter of the alphabet=E↵

You entered correctly

:
```

IP – Terminal Input to Active Input Buffer

The IP command accepts data input from the terminal and inserts it into the active input buffer.

Syntax

IP{B}{r}

B	Replaces all blanks input at the terminal with backslashes (\).
r	Specifies a prompt character to be displayed at the terminal for operator input; it can be any character, including a blank. The prompt character remains in effect until another IN, IP, or IS command with a different prompt character is executed or a () (transfer control) or [] (subroutine call) is executed. If omitted, the previously specified prompt character, if any, is used. If no prompt character has been specified previously, a colon (:) is used as the prompt character.

Description

The data input from the terminal replaces the current parameter in the active input buffer.

Note: *The current parameter of an active buffer is the data from the current position of the pointer to the end of the parameter. If the pointer does not point at the first character of data delimited by blanks, the characters prior to the pointer are not replaced.*

If several parameters are input at the terminal, they only replace the current parameter in the buffer. If the input pointer is at the end of the data, the new input data is appended after the last parameter.

The input pointer location is not changed; it remains pointing at the beginning of the data that was input.

Data input by the IP command can then be copied to the output buffer by an A command. (See the A command listed alphabetically in this chapter.)

```
001 PQ
002 IHAAA BBB CCC
003 A2,1
004 IP
005 F
006 F
007 IP=
```

Statement	User is prompted with	User responds with	Primary input buffer
`IHAAA BBB CCC`			`AAA_BBB_CCC` `↑`

`A2,1`			`AAA_BBB_CCC`
			`    ↑`

`IP`	`:`	`XX YY↵`	`AAA_BXX_YY_CCC`
			`     ↑`

`F`			`AAA_BXX_YY_CCC`
			`        ↑`

`F`			`AAA_BXX_YY_CCC`
			`           ↑`

`IP=`	`=`	`ZZ↵`	`AAA_BXX_YY_ZZ`
			`           ↑`

IS – Terminal Input to Secondary Input Buffer

The IS command is a synonym for the IN command. Refer to the IN command in this chapter for complete information.

IT – Tape Input to Primary Input Buffer

The IT command inputs the next tape label from the currently attached tape unit.

Syntax

IT

Description

The IT command clears the input buffers, reads the current tape label from the attached tape unit, stores it in the primary input buffer, and positions the input pointer at the beginning of the tape label. Thus, any data placed in the primary input buffer after the IT command will be placed at the beginning of the input buffer unless the input pointer is positioned at a specified location.

Note: *Any data in the input buffers prior to the IT command will be lost.*

Data input by the IT command can then be copied to the output buffer by an A command (see the A command listed alphabetically in this chapter) or it can be tested with an IF command.

The program below will T-FWD a tape, read the label with the IT command, check to see if it is the file you are looking for using an IF command; if it is the file specified, the file is T-LOADed; otherwise, the tape is T-FWDed to the next file and the check is repeated.

```
    IT.TEST
PQ
HT-REW
P
10 IT
DO
IF A(30) # DOCUMENT G 20
HT-LOAD DOCUMENT (O
P
X
20 HT-FWD
P
G 10
```

O – Output Text

The O command displays specified text to the terminal.

Syntax

O{text}{+}

text	Specifies the text to output. A carriage return is appended to the text unless the + parameter is present.
+	Suppresses the carriage return so that the cursor does not move to the next line after the text is displayed. This feature is useful when the O command is used in conjunction with an input (IP, IN, or IS) command.

Description

The O command is useful to prompt the user for input from the terminal.

The O command has no effect on the buffers.

```
001 PQ
002 C Output text to terminal
003 OPart-Number+
004 C Data is accepted from the terminal
005 IS=
```

Output to terminal

```
Part-Number=
```

P – Execute Primary Output Buffer

The P command executes the command string in the primary output buffer.

Syntax

P{c}

c Specifies alternative actions associated with the PROC execution. One of the following can be specified:

H	Hushes (suppresses) output to the terminal.
P	Displays the content of the output buffers before execution.
W	Displays the content of the output buffers and waits for user response. The user can enter the following options at the prompt:

N	PROC execution is aborted and the system exits to TCL.
S	PROC execution continues at the command following PW.
X	PROC execution is aborted and the system exits to TCL.

Any other character causes PROC action to continue.

X	Terminates PROC control after execution.

Note: *If the argument c is omitted, no alternative action is taken.*

Description

The P command submits the contents of the primary output buffer to TCL for processing.

The secondary output buffer can be used to feed interactive processors such as BASIC or the line Editor; or the secondary output buffer can contain another TCL statement if the primary output buffer contains SELECT, SSELECT, QSELECT, or GET-LIST. If the secondary output buffer contains data but the last character of the last line is not a carriage return (<), the P command places a carriage return character at the end of the secondary output buffer.

After execution of a P command, if the statement executed from the primary output buffer was a PROC, at the conclusion of that statement, control is returned to TCL, not to the current PROC. To return to the current PROC after executing another PROC, use the [] command to execute the other PROC.

After execution of a P command (except for PX), if the statement executed was a statement other than a PROC, the current PROC regains control at the statement immediately following the P command. The primary input buffer is selected as the active input buffer, and the input pointer is positioned to the beginning of the buffer on return of control to the PROC from TCL. The secondary input buffer and both output buffers are cleared. Data in the primary input buffer is not cleared.

An automatic P command is executed if the following two conditions are met:

- the last line of the PROC is reached and it is not an X command
- the primary output buffer contains data

The PW form of the P command operates the same as the PP command, except after the buffer contents are displayed, the system displays a question mark (?) prompt character and waits for terminal input. PW is normally used as a debugging tool and can be replaced by P once the user has determined that the PROC is functioning properly.

Note: *Any system message identifier returned by the system command is placed in the secondary input buffer and can be tested with the IF E command. (For system message checking procedures, see the IF E command listed alphabetically in this chapter.)*

```
PQ
RI
C The DATE command is copied in the input buffer
IHDATE
C The WHO command is copied in the output buffer
HWHO
C The output buffer is executed
P
C PROC regains control and copies the DATE command
C to the output buffer
A
C The output buffer is automatically executed
```

Output to terminal

```
13 SUSAN

11:27:23   06 MAY 1992

:
```

RI – Clear Input Buffers

The RI command clears the input buffers to null, resets the input pointers, and selects the primary input buffer as the active input buffer.

Syntax

```
RI{p}
```

p	Specifies the parameter in the primary input buffer at which to start clearing. The primary input buffer is cleared from the pth parameter to the end of the buffer. If p is omitted, the primary input buffer is cleared from the beginning of the buffer. The secondary input buffer is always cleared from the beginning of the buffer to the end.

Description

After clearing the input buffers to null, RI selects the primary input buffer as the active input buffer and the input pointer is positioned at the pth parameter if specified; if p is not specified, the input pointer is at the beginning of the buffer.

```
PQ
RI
IHABC DEF GHI JKL
SS
IHXX YY
C Clear to the end of the buffer starting with
C parameter 3
RI3
```

```
                       Primary               Secondary
Statement              input buffer          input buffer

RI
                       ↑
IHABC DEF GHI JKL      ABC_DEF_GHI_JKL
                       ↑
SS                     ABC_DEF_GHI_JKL
                                             ↑
IHXX YY                ABC_DEF_GHI_JKL       XX_YY
                                             ↑
RI3                    ABC_DEF
                               ↑
```

RO – Clear Output Buffers

The RO command clears the output buffers to null, selects the primary output buffer as the active output buffer, and sets the output pointer to the beginning of the buffer.

Syntax

RO

Description

The RO command clears both output buffers to the empty (null) condition, selects the primary output buffer, and sets the output pointer to the beginning of the buffer.

The RO command has no effect on the input buffer.

```
001 PQ
002 HABC DEF GHI JKL
003 STON
004 HXXX YYY
005 C Clear the output buffer
006 RO
```

```
                    Primary               Secondary
Statement           output buffer         output buffer

HABC DEF GHI JKL    ABC_DEF_GHI_JKL
                                    ↑

STON                ABC_DEF_GHI_JKL

                                          ↑

HXXX YYY                                  XXX_YYY
                                                 ↑

RO

                    ↑
```

S(m) – Position Primary Input Pointer

The S(m) command positions the input pointer at a specified column position in the primary input buffer.

Syntax

S(m)

m	Specifies the column position m at which the input pointer is to be placed.

Description

The S(m) command selects the primary input buffer, and positions the input pointer to column position m. If there are fewer than m column positions in the primary input buffer, the input pointer is set to the end of the buffer.

The S(m) command has no effect on the data content of either input buffer.

```
001 PQ
002 RI
003 IHABC DE FGHIJ
004 SS
005 IHXX YY
006 F
007 C Move pointer to column position 2
008 S(2)
```

```
                    Primary             Secondary
Statement           input buffer        input buffer

IHABC DE FGHIJ      ABC_DE_FGHIJ
                    ↑

SS                  ABC_DE_FGHIJ
                                        ↑

IHXX YY             ABC_DE_FGHIJ        XX_YY
                                        ↑

F                   ABC_DE_FGHIJ        XX_YY
                                           ↑

S(2)                ABC_DE_FGHIJ
                     ↑
```

Sn – Position Active Input Pointer

The Sn command positions the input pointer at a specified parameter in the active input buffer.

Syntax

Sn

n	Specifies the ordinal number of the parameter at which the input pointer is to be placed. If n is zero (S0) or one (S1), the pointer is set at the beginning of the buffer. The parameters are assumed to be separated by blanks or enclosed in single quotes. If there is no nth parameter, the pointer is set to the end of the input buffer.

Description

The Sn command positions the input pointer in the active input buffer at the specified parameter.

The Sn command has no effect on the data content of either input buffer.

```
PQ
RI
IHABC DE FGHIJ
C Move pointer to third parameter
S3
C Move pointer to the end of the buffer because a
C fifth parameter is not present
S5
```

```
Statement                Primary input buffer

IHABC DE FGHIJ           ABC_DE_FGHIJ
                         ↑

S3                       ABC_DE_FGHIJ
                                ↑

S5                       ABC_DE_FGHIJ
                                     ↑
```

SP – Select Primary Input Buffer

The SP command selects the primary input buffer as the active input buffer.

Syntax

```
SP
```

Description

Only one of the two input buffers can be active. Upon initial entry to a PROC, the primary input buffer is active.

The SP command selects the primary input buffer to be active, and sets the input pointer to the beginning of the buffer. The primary input buffer can be selected as active at any point within a PROC.

The SP command has no effect on the data content of either input buffer.

SS – Select Secondary Input Buffer

The SS command selects the secondary input buffer as the active input buffer.

Syntax

```
SS
```

Description

Only one of the two input buffers can be active. Upon initial entry to a PROC, the primary input buffer is active.

The SS command selects the secondary input buffer to be active, and sets the input pointer to the beginning of the buffer. The secondary input buffer can be selected as active at any point within a PROC.

The SS command has no effect on the data content of either input buffer.

STOFF – Select Primary Output Buffer

The STOFF command selects the primary output buffer as the active output buffer.

Syntax

```
STOFF
```

Description

Only one of the two output buffers can be active. Upon initial entry to a PROC, the primary output buffer is active.

The STOFF (STack OFF) command selects the primary output buffer as the active output buffer and sets the output pointer to the end of the buffer. The primary output buffer can be selected as active at any point within a PROC.

The STOFF command has no effect on the data content of either output buffer.

STON – Select Secondary Output Buffer

The STON command selects the secondary output buffer as the active output buffer.

Syntax

STON

Description

Only one of the two output buffers can be active. Upon initial entry to a PROC, the primary output buffer is active.

The STON (STack ON) command selects the secondary output buffer as the active output buffer and sets the output pointer to the end of the buffer. The secondary output buffer can be selected as active at any point within a PROC.

The STON command has no effect on the data content of either output buffer.

```
     STON.TST
PQ
C Selects a list and puts it in the primary output
C buffer
HSELECT MD WITH 1 = "9"
C Secondary output buffer is active
STON
C Save the list to the secondary output buffer. If
C a list is not present, a system message will be
C placed in the buffer
HSAVE-LIST ITEMS

:STON.TST↵

[401] No items present.

:
```

T – Terminal Control

The T command is used to specify cursor positioning and visual attributes, to output literals, or to output non-keyable character codes. The cursor functions are terminal independent; special terminal function codes are available.

Syntax

```
T {f{,f{,f{...}}}}{,}
```

f can be any of the following:

"text"	Causes the literal text to be output at the current position.
B	Causes a bell code to be output.
C	Causes a clear screen code to be output.
Inn	Displays the ASCII character whose decimal value is nn.
Xxx	Displays the ASCII character whose hexadecimal value is xx.
(c{,r})	Causes the terminal cursor to position to column c, row r (if r is present). If only c is present, the cursor is positioned to column c of the current row on terminals with column-only addressing. However, since not all terminals support this addressing mode, its use is not recommended.
(-n)	Generates a cursor control function as shown in Table 2-3. The terminal must support this function.
,	Allows continuation of the command to multiple lines.

Description

Columns and rows on the terminal screen are numbered starting with zero (0), left to right and top to bottom. For example, (0,0) is the upper left corner of the terminal screen. Users can create formatted screens in PROCs with the T command.

The cursor is left where the T command positions it (that is, a carriage return or line feed is not automatically output at the end of the T command).

There must be a blank separating the T command and the first function code. Comments can be placed immediately after any argument. For example, the following two statements are equivalent:

```
T C

T CLEAR
```

Comments cannot contain commas since the comma is used to terminate the comment. Another restriction exists with the Inn and Xxx arguments; a comment cannot start with a character that would be interpreted as a decimal or hexadecimal digit, respectively. Starting a comment with a blank is an easy way to ensure this.

The T command generates terminal control codes based on the current terminal type for the line on which the PROC is executing. The terminal type is determined by the following:

- most recent TERM or TERM-INIT command executed for the line
- a TERM type logon parameter set up in DICT ACC for the line
- the system-wide default terminal type, which setup by the SET-TERM command

For more information on these commands, please refer to the *Ultimate System Commands Guide*.

It is recommended that the programmer employ the terminal independent (-n) function codes instead of hard coding these functions for a single terminal type. See Table 2-3 for the available codes. Note that these codes are the same as those for the BASIC @ Function (see the @ Function in the Ultimate *BASIC Language Reference Guide*); although all can be specified, some codes are intended primarily for use in BASIC and do not have much value in PROC, specifically true for codes (-46) through (-51).

Table 2-3. Cursor Control Values (1 of 5)

Code	Description
(-1)	Clear the screen and positions the cursor at 'home' (upper left corner of the screen).
(-2)	Position the cursor at 'home' (upper left corner).
(-3)	Clear from cursor position to the end of the screen
(-4)	Clear from cursor position to the end of the line
(-5)	Start blink
(-6)	Stop blink
(-7)	Start low intensity
(-8)	Stop low intensity
(-9)	Backspace the cursor one character
(-10)	Move the cursor up one line
(-11)	Move the cursor down one line
(-12)	Move the cursor right one column
(-13)	Enable auxiliary (slave) port
(-14)	Disable auxiliary (slave) port
(-15)	Enable auxiliary (slave) port in transparent mode
(-16)	Initiate slave local print
(-17)	Start underline
(-18)	Stop underline
(-19)	Start reverse video
(-20)	Stop reverse video
(-21)	Delete line
(-22)	Insert line

Table 2-3. Cursor Control Values (2 of 5)

Code	Description
(-23)	Scroll screen display up one line
(-24)	Start boldface type
(-25)	Stop boldface type
(-26)	Delete one character
(-27)	Insert one blank character
(-28)	Start insert character mode
(-29)	Stop insert character mode
(-30,c)	Set foreground and background color:

c	background	foreground
1	black	cyan
2	black	red
3	black	blue
4	black	green
5	black	magenta
6	black	yellow
7	black	white
8	blue	red
9	blue	green
10	blue	white
11	blue	yellow
12	blue	red
13	blue	cyan
14	blue	magenta
15	white	red
16	white	green
17	white	blue
18	white	cyan
19	white	magenta
20	white	black
21	red	white
22	red	green

Table 2-3. Cursor Control Values (3 of 5)

Code	Description		
(-31,f)	Set foreground color:		
	f	**foreground**	
	1	brown	(can vary on some terminals)
	2	white	
	3	red	
	4	magenta	
	5	yellow	
	6	green	
	7	cyan	
	8	blue	
(-32,b)	Set background color:		
	b	**background**	
	1	brown	
	2	white	
	3	black	
	4	red	
	5	blue	
	6	cyan	
	7	magenta	
(-33)	Set 80 columns		
(-34)	Set 132 columns		
(-35)	Set 24 rows		
(-36)	Set 44 rows		
(-37)–(45)	Reserved		

Table 2-3. Cursor Control Values (4 of 5)

Code	Description
(-46)	Returns function key default values; intended for BASIC
(-47)	Returns character sequence needed to set the overall characteristics for the label line (bottom line of terminal); intended for BASIC
(-48)	Returns character sequence needed to set the overall characteristics for the status line (top line of terminal); intended for BASIC
(-49)	Returns string that defines the graphic character codes for the current terminal; intended for BASIC
(-50)	Start graphics; intended for BASIC
(-51)	Stop graphics; intended for BASIC
(-52)	Start blink
(-53)	Stop blink
(-54)	Start reverse video
(-55)	Stop reverse video
(-56)	Start reverse video and blink
(-57)	Stop reverse video and blink
(-58)	Start underline
(-59)	Stop underline
(-60)	Start underline and blink
(-61)	Stop underline and blink
(-62)	Start underline and reverse video
(-63)	Stop underline and reverse video

Table 2-3. Cursor Control Values (5 of 5)

Code	Description
(-64)	Start underline, reverse video, and blink
(-65)	Stop underline, reverse video, and blink
(-66)	Start dim
(-67)	Stop dim
(-68)	Start dim and blink
(-69)	Stop dim and blink
(-70)	Start dim and reverse video
(-71)	Stop dim and reverse video
(-72)	Start dim, reverse video, and blink
(-73)	Stop dim, reverse video, and blink
(-74)	Start dim and underline
(-75)	Stop dim and underline
(-76)	Start dim, underline, and blink
(-77)	Stop dim, underline, and blink
(-78)	Start dim, reverse video, and underline
(-79)	Stop dim, reverse video, and underline
(-80)	Set 80 columns
(-81)	Reserved
(-82)	Set 132 columns

`T C,B,(10,5),"TITLE"`	This sequence first clears the screen, outputs a bell code to the terminal, positions to column 10 row 5, and displays the text TITLE.
`T (0,8),(-4)`	This sequence positions the cursor at column 0 row 8, and clears the entire line assuming that the terminal used supports that function.
`T (-5),"twinkle",(-6)`	This sequence starts a blinking field, prints the word twinkle, and ends the blinking field. This assumes the terminal supports blinking.

`T CLEAR,"TITLE",`
`(5,5) Comment,"TEXT"`
	This sequence continues over a line boundary and inserts a comment in the line.

X – PROC Termination

The X command terminates execution of a PROC, with the optional display of text and returns control to TCL, to the calling PROC, or to the BASIC program that invoked this PROC with an EXECUTE statement.

Syntax

```
X{text}{+}
```

text	Contains a message to be displayed on the terminal when the PROC is terminated.
+	Can be appended to the text to suppress the carriage return and line feed that is normally appended to text displayed on the terminal. If you return to a calling PROC, the cursor remains at the character position after the last character in the message. If you exit to TCL, the cursor goes to the next line, but the blank line between the text and the TCL prompt is suppressed.

Description

PROC control is terminated after the final PROC command has been executed. Therefore, the X command is not needed in this case. The X command is intended as a method of terminating a PROC at an intermediate point.

If the PROC was invoked from the TCL level, it exits to TCL. If the PROC was called as a subroutine, the X command causes a return to the calling PROC and continuation at the next command in that PROC.

If control passes to a TCL level that was active due to an EXECUTE statement in a BASIC program, control then passes back to the BASIC program. (See the BASIC Language Reference Guide for more details.)

`X`	PROC terminates and returns to TCL or to the calling PROC.
`X***EXIT TO TCL**`	Same as above; in addition, a message is printed with a carriage return and a line feed appended.
`XHURRY BACK+`	Same as above; however, a message is printed without a carriage return or a line feed appended.

3 Reference for PROC Programmers

This chapter contains PROCs that perform the following functions:

- File updating
- Tape positioning
- Running a BASIC program
- Using SSELECT and COPY commands
- Reading a tape file
- Using two operator inputs in one command
- Printing a mailing list
- Displaying main menu for job selection

File Updating

The following example shows a line editor operation which changes attribute 3 of item 11115 in file ACCOUNT to 101 AVOCADO. The item and the file must already exist.

```
:EDIT  ACCOUNT  11115↵
Top
.G3↵                    Go to line 3
003 100 AVOCADO         Displays line 3
.R/100/101↵             Replace 100 with 101
003 101 AVOCADO         New  replacement line
.FI                     Saves item using the same name
'11115' filed
:
```

The following PROC named CHANGE performs the identical operation as the above example. It is written in such a manner that it updates any specified attribute in any specified item in any specified file.

CHANGE	**Explanation**
`001 PQ`	PROC code.
`002 HED`	System command to output buffer.
`003 A2`	2nd parameter to output buffer.
`004 A3`	3rd parameter to output buffer.
`005 STON`	Secondary output buffer active.
`006 HG`	'G' to output buffer
`007 A4`	4th parameter to output buffer.
`008 H<`	Carriage return to output buffer.
`009 HR/`	'R' and / to output buffer.
`010 A5`	5th parameter to output buffer.
`011 H/`	/ to output buffer.
`012 A6`	6th parameter to output buffer.
`013 H<`	Carriage return output buffer.
`014 HFI`	'FI' to output buffer.
`015 PP`	Display and execute PROC.

`:CHANGE ACCOUNT 11115 3 100 101↵`

Output to terminal

```
EDIT ACCOUNT 11115
G3
R/100/101_
FI
Top
003 100 AVOCADO
003 101 AVOCADO
'11115' filed.

:
```

Tape Positioning

The following PROC named TAPE-FWD positions a tape. This PROC illustrates the use of numeric labels for transfer of PROC control via the GO command. It also uses the arithmetic commands, + and -.

To execute this PROC from the TCL level, enter the following:

```
TAPE-FWD or TAPE-FWD n
```

where *n* is the number of files to be spaced over. If *n* is omitted or non-numeric, the PROC prompts for NO. OF FILES.

During execution, the active input buffer contains:

```
TAPE-FWD_n_xx
```

where:

n	is the number of files left to be spaced over.
xx	is the count of such files already spaced over, initialized to 000 at line 5 of the sample PROC.

The PROC attaches tape unit 0 (T-ATT). On multi-drive systems, if unit 0 is not the desired unit, the desired tape unit must already be attached. If the tape drive is already attached to another line (error 95 on line 8 of the sample PROC), the PROC terminates execution.

The PROC then does a T-RDLBL to read the tape label and print it on the terminal; if an end-of-file (EOF) error (this is error 94 on line 12) occurs on this statement, the end of tape data has been reached; the message End of recorded data (on line 23) is printed, along with the file-count from parameter 3.

If T-RDLBL executes successfully, the tape file is spaced over by executing a T-FWD. This is repeated until parameter 2 goes to 0.

Note: *The IFN command is used to test for a value of zero regardless of the number of 0 digits in the parameter. Alternatively, multiple tests could be used to test for 0, 00 , or 000, since the - command doesn't change the parameter size.*

TAPE-FWD	**Explanation**
`001 PQ`	
`002 4 IF #A2 G 3`	
`003 IF A2 # (0N) G 3`	Don't accept non-numeric data.
`004 S3`	Position pointer at 3rd parameter.
`005 IH000`	Initialize 3rd parameter to 0.
`006 HT-ATT`	System command to output buffer.
`007 P`	
`008 IF E = 95 X`	Test for tape attached to another line; if so, exit.
`009 IFN A = 0 X`	If 2nd parameter is 0, exit.
`010 2 HT-RDLBL`	Copy system command to output buffer.
`011 P`	
`012 IF E = 94 G 9`	Test for End of file; if so, go to label 9.
`013 HT-FWD`	System command to output buffer.
`014 P`	
`015 S3`	Position pointer at 3rd parameter.
`016 +1`	Increment parameter by 1.
`017 S2`	Position pointer at 2nd parameter.
`018 -1`	Decrement parameter by 1.
`019 G 2`	Go to 2 (continue PROC).
`020 3 ONo. of files+`	
`021 IP?`	
`022 G 4`	
`023 9 OEnd of recorded data - (+`	
`024 D3+`	Display 3rd parameter.
`025 X FILES)`	Display text and exit PROC.

Running a BASIC Program

The following PROC named RUN.BP puts two lines of data in the secondary output buffer and runs a BASIC program named INPUTTER. Both data lines are terminated with a carriage return; the second < is optional (that is, if the explicit carriage return, <, were omitted, PROC would automatically append one to the end of the secondary output buffer).

RUN.BP

```
PQ
HRUN BP INPUTTER
STON
H100<
H200<
PP
```

The following BASIC program named INPUTTER expects three inputs. When INPUTTER is executed, variable A is assigned the value 100 and variable B is assigned the value 200. Then, since the secondary output buffer is now empty, a prompt message is displayed on the terminal and C is assigned the value entered in from the keyboard.

INPUTTER

```
INPUT A
INPUT B
PRINT "Enter department code":
PROMPT ":"
INPUT C
```

```
:RUN.BP↵
RUN BP INPUTTER
100_
200_
Enter department code:
```

Using SSELECT and COPY Commands

The following example shows a sample operation at the TCL level using a SSELECT command and then a COPY command.

```
:SSELECT  INVENTORY  WITH  QOH  >  "900"  BY-DSND  QOH↵
19 items selected
:COPY  INVENTORY↵
TO:(HOLD-FILE)↵
19 items copied
:
```

The following PROC named SELECT.COPY will perform the sample operation as above. An IF S test after line 5 is used to avoid executing the COPY statement if no items were selected. For more information on the IF S test, see the IF command listed in Chapter 2 of this manual.

SELECT.COPY

```
001 PQ
002 HSSELECT INVENTORY WITH QOH > "900" BY-DSND QOH
003 STON
004 H<
005 P
006 IF #S X
007 HCOPY INVENTORY
008 STON
009 H(HOLD-FILE)<
```

For more information using the SSELECT verb, please refer to the *Ultimate RECALL and Ultimate UPDATE User Guide*. For further information about the COPY and SSELECT command formats, refer to the *Ultimate System Commands Guide*.

Reading Tape Files

The following PROC named T-READER reads the first record of successive tape files using the T-READ command.

T-READER

```
PQ
C PROC to T-READ the first record of
C successive tape files
C Read 1 record
1 HT-READ 1
P
O
OContinue to read the next record (Y/N)+
IN?
C If user does not continue, exit
IF A = N X
C If user continues, tape is forwarded to next file
HT-FWD
P
C Go to label 1 to read another record
G 1
```

For more information using the T-READ and T-FWD commands, please refer to the *Ultimate System Commands Guide.*

Using Two Operator Inputs in One Command

The following PROC combines two parameters entered by the operator with a literal string in the primary output buffer that directs the system to list a record that has the required values. The operator must enter a balance equal to or greater than 0; otherwise, the PROC will prompt the operator for another balance.

ACCT.BAL

```
PQ
T C
O
OEnter filename to list current balance from +
S1
IP
10 O
OList only accounts with a balance greater than +
S2
IP
IF A = (0N) G 20
G 30
20 H LIST
A1
H WITH CURR.BAL > "
A\2
H" NAME CURR.BAL (I
C Display input buffer parameters
OInput parameters are:
D0
O
C Display output buffer parameters
OOutput parameters are:
PW
X
30 O***BALANCE MUST BE EQUAL TO OR GREATER THAN 0***
G 10
```

```
:ACCT.BAL↵
Enter filename to list current balance from :CUSTOMERS↵

List only accounts with a balance greater than :200↵

Input parameters are:
CUSTOMERS 200

Output parameters are:
LIST CUSTOMERS WITH CURR.BAL > "200" NAME CURR.BAL (I
?↵

PAGE 1                                    time   date

NAME................ CURR.BAL........

ROGERS LTD.                 $250.00
THE PRINT SHOP            $1,500.00
CITY OF E. HANOVER        $3,000.00
QUICK PRINT               $1,000.00

4 items listed.

:
```

Printing Mailing Lists

The following PROC produces a mailing list of selected item-IDs. It assumes that the item-IDs in the CUSTOMERS file are right justified numeric.

MAIL.LIST

```
PQ
O Print a mailing list in alphabetical order
1 O
OEnter starting item-ID (X to end) +
S2
IP
IF A = X X
IF #A G 1
IF A # (0N) G 1
2 O
OEnter ending item-ID (X to end) +
S3
IP
IF A = X X
IF #A G 2
IF A # (0N) G 2
H SORT CUSTOMERS
H >= "
A\2
H" AND <= "
A\3
H"
H BY NAME
H ID-SUPP
H NAME ADDRESS CITY STATE ZIP
P
```

```
:MAIL.LIST↵
Print a mailing list in alphabetical order

Enter starting item-ID (X to end) :300↵

Enter ending item-ID (X to end) :305↵

PAGE 1                                  time   date

NAME........... ADDRESS......... CITY...... STATE ZIP..

QUICK PRINT     4 CARTER AVE.    NEWARK       NJ  07111
ROGERS LTD.     80 MAIN ST.      W. ORANGE    NJ  07052
SLONA MANUF.    103 HILLTOP RD.  WHITING      NJ  08759
THE PRINT SHOP  10 MAIN AVE.     W. HANOVER   NJ  07936
ULTIMATE CORP.  RIDGEDALE AVE.   E. HANOVER   NJ  07936

5 items listed.

:
```

Displaying Main Menu For Job Selection

The following PROC uses T commands with cursor positioning values to display a menu.

MENU

```
PQ
C ---- MAIN MENU -----
100 T C
T,(26,2),"----- MAIN MENU -----"
T,(14,5),"---- Accounting ----"
T,(14,12),"----Distribution----"
T,(47,5),"--- System ---"
T,(10,7)," 1. Accounts Receivable"
T,(10,8)," 2. Accounts Payable"
T,(10,9)," 3. General Ledger"
T,(43,7)," 4. System Controls"
T,(43,8)," 5. Utility Menu"
T,(10,14)," 6. Inventory Control"
T,(10,15)," 7. Order Processing"
T,(10,16)," 8. Retail Point-of-Sale"
T,(10,17)," 9. Purchase Orders"
T,(10,18)," 10. Sales Analysis"
T,(10,19)," 11. Physical Inventory"
200 T,(27,22)," Enter Selection (X to end)"+
IN
IF #A G 200
IF A = X G 999
IF A = END G 999
IF A = TCL G 999
IF A = OFF G 900
IF A # (0N) G 200
IFN A > 11 G 200
G A
G 200
1 [MENUS MAR]
G 100
2 [MENUS MAP]
G 100
3 [MENUS MGL]
```

```
035 G 100
036 4 [MENUS MIC]
037 G 100
038 5 [MENUS MOP]
039 G 100
040 6 [MENUS MPOS]
041 G 100
042 7 [MENUS MPO]
043 G 100
044 8 [MENUS MSA]
045 G 100
046 9 [MENUS MPI]
047 G 100
048 10 [MENUS MSYS]
049 G 100
050 11 [MENUS MUTIL]
051 G 100
052 900 HOFF
053 P
054 999 X
```

```
:MENU↵
                 ----- MAIN MENU -----


      ---- Accounting ----           --- System ---

     1. Accounts Receivable         4. System Controls
     2. Accounts Payable            5. Utility Menu
     3. General Ledger

      --- Distribution ---

     6. Inventory Control
     7. Order Processing
     8. Retail Point-of-Sale
     9. Purchase Orders
    10. Sales Analysis
    11. Physical Inventory


               Enter Selection (X to end):
```

Notes

A List Of ASCII Codes

This appendix presents a list of ASCII codes for decimal number values from 0 through 255. The hexadecimal equivalent value and ASCII character generated are also given.

Decimal values 0-31 are assigned as non-printable functions; these codes may be specified by control key sequences. In the listing, the control key is indicated by a caret (^) in the first position in the Key column.

Decimal values greater than 127 (x'7F') are not defined in the ASCII character set. The functions or characters assigned to these values are dependent on the terminal being used. Special file structure functions and control key sequences have been assigned to decimal values 28 through 31 (X'1C' through X'1F') and 251 through 255 (x'FB' through x'FF').

Decimal	Key	Hexadecimal	Name
0	^@	00	NUL
1	^A	01	SOH
2	^B	02	STX
3	^C	03	ETX
4	^D	04	EOT
5	^E	05	ENQ
6	^F	06	ACK
7	^G	07	BEL
8	^H	08	BS
9	^I	09	HT
10	^J	0A	LF
11	^K	0B	VT
12	^L	0C	FF
13	^M	0D	CR
14	^N	0E	SO
15	^O	0F	SI
16	^P	10	DLE
17	^Q	11	DC1
18	^R	12	DC2
19	^S	13	DC3
20	^T	14	DC4
21	^U	15	NAK
22	^V	16	SYN
23	^W	17	ETB
24	^X	18	CAN
25	^Y	19	EM
26	^Z	1A	SUB
27	^[	1B	ESC
28		1C	FS
29		1D	GS
30		1E	RS
31		1F	US

Decimal	Key	Hex
32		20
33	!	21
34	"	22
35	#	23
36	$	24
37	%	25
38	&	26
39	'	27
40	(	28
41	)	29
42	*	2A
43	+	2B
44	,	2C
45	-	2D
46	.	2E
47	/	2F
48	0	30
49	1	31
50	2	32
51	3	33
52	4	34
53	5	35
54	6	36
55	7	37
56	8	38
57	9	39
58	:	3A
59	;	3B
60	<	3C
61	=	3D
62	>	3E
63	?	3F
64	@	40
65	A	41
66	B	42
67	C	43
68	D	44
69	E	45
70	F	46
71	G	47
72	H	48
73	I	49
74	J	4A
75	K	4B
76	L	4C
77	M	4D
78	N	4E
79	O	4F
80	P	50
81	Q	51
82	R	52
83	S	53
84	T	54
85	U	55
86	V	56
87	W	57
88	X	58
89	Y	59
90	Z	5A
91	[	5B
92	\	5C
93	]	5D
94	^	5E
95	_	5F
96	`	60
97	a	61
98	b	62
99	c	63
100	d	64
101	e	65
102	f	66
103	g	67
104	h	68
105	i	69
106	j	6A
107	k	6B
108	l	6C
109	m	6D
110	n	6E
111	o	6F
112	p	70
113	q	71
114	r	72
115	s	73
116	t	74
117	u	75
118	v	76
119	w	77
120	x	78
121	y	79
122	z	7A
123	{	7B
124	|	7C
125	}	7D
126	~	7E
127	DEL	7F

Decimal	Key	Hexadecimal	Symbol	Name
128 (x'80') thru 250 (x'FA')			not used	
251		FB	SB	Start buffer
252	^\	FC	SVM	Subvalue Mark
253	^]	FD	VM	Value Mark
254	^^	FE	AM	Attribute Mark
255	^_	FF	SM	Segment Mark

Index

Ultimate
THE ULTIMATE CORP.

ID Number____________ *for Ultimate use only*

Problem Identification Form

Name	Phone Number ( )	System Number	Date

At TCL, execute REV verb and enter the following information: Firmware rev. ________ Kernel rev. ________ Async rev. ________ Abs rev. ________ Diags rev. ________ ECOs ________ ________	Hardware Platform: (manufacturer, model no.) Host O/S and revision Dealer Name

At TCL, execute WHAT (LSWP) verb and attach listing to this report.

Description of what happened and steps necessary to recreate (attach listings, tapes, if available):

FROM:

Name:	________________	System Number:	________		
Company	________________				
Address:	________________				
City:	________________	State:	______	Zip:	______

Fold and tape. Please do not staple.

The Ultimate Corp.
717 Ridgedale Avenue
East Hanover, NJ 07936
Attn: Technical Support

Fold and tape. Please do not staple.

Ultimate
THE ULTIMATE CORP.

Reader Comment Form

Ultimate welcomes your comments. If you find a problem or error in this manual, or can suggest an improvement, please complete this form. Please attach additional sheets, if necessary.

Name	Phone Number ( )	System Number
Name of Manual	Document Number	Date
Comments:		

FROM:

Name:	______________________	System Number:	__________
Company	______________________		
Address:	______________________		
City:	______________________	State: ______	Zip: ________

Fold and tape. Please do not staple.

The Ultimate Corp.
717 Ridgedale Avenue
East Hanover, NJ 07936
Attn: Technical Support

Fold and tape. Please do not staple.

Ultimate

THE ULTIMATE CORP.

Suggestion Form

Ultimate welcomes your suggestions. If you have a suggestion or would like to recommend an enhancement, please complete this form. Please attach additional sheets, if necessary.

Name	Phone Number ( )	System Number	Date
Company Name	Hardware Platform: (manufacturer, model no.)		
Dealer Name	Host O/S and revision		
Suggestion:			

6936-3

FROM:

Name:	______________________	System Number:	__________
Company	______________________		
Address:	______________________		
City:	______________________	State: ______	Zip: ______

Fold and tape. Please do not staple.

The Ultimate Corp.
717 Ridgedale Avenue
East Hanover, NJ 07936
Attn: Technical Support

Fold and tape. Please do not staple.

6936-3

Ultimate

THE ULTIMATE CORP.

717 Ridgedale Avenue
East Hanover, NJ 07936
201/887-9222
FAX 201/887-9546

Ultimate

THE ULTIMATE CORP.

717 Ridgedale Avenue
East Hanover, NJ 07936
201/887 9222
FAX 201/ 887 6139

PROC Reference Guide 6936-3