REALITY®
(3.0 SERIES)

Proc & Batch
Programming Manual

771044

PROPRIETARY INFORMATION

The information contained herein is proprietary to and considered a trade secret of Microdata Corporation and shall not be reproduced in whole or part without the written authorization of Microdata Corporation.

Specifications Subject to Change Without Notice
Printed in U.S.A.
Price: $10.00

®Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764

CONTENTS

Title Topic

CONTENTS Cont'd.

HOW TO USE THE REALITY® MANUALS

The Reality® manuals are written in modular format with each pair of facing pages presenting a single topic.

This and other Reality manuals differ substantially from the typical reference manual format. The left-hand page of each topic is devoted to text, while the right-hand page presents figures referred to by the text. At the head of each text page are a pair of titles, the first title naming the section, the second the topic. Immediately below these titles is a brief summary (boxed) of the material covered in the topic.

The advantage of this format will become readily apparent to the reader as he uses this manual. First, the figures referred to in the text are always conveniently right in front of the reader at the point where the reference is made. Secondly, the reader knows that when he turns the page, he has completed one idea and is ready to encounter a new one.

Documentation for the Reality system includes the following:

Reality Programmer's Reference Manual, #1048

Reality EDITOR Operator's Guide, #1052

Reality ENGLISH$^{\text{T.M.}}$ Programming Manual, #1038

Reality DATA/BASIC$^{\text{T.M.}}$ Programming Manual, #1051

Reality PROC and BATCH Programming Manual, #1044

Reality Assembly Language Programming Manual, #1049

Reality Bisync Operator's Guide, #1043

The examples throughout this manual use certain conventions as defined in Figure A.

CONVENTION	MEANING
TEXT	*Shaded text represents the user's input.*
TEXT	*Standard text represents computer output printed by the system.*
TEXT	*Italicized text is used for comments and notes which help explain or describe the example.*
CR	*This symbol represents a carriage return.*
b̸	*This symbol represents a space (blank).*

Figure A. Conventions Used Throughout This Manual

1 INTRODUCTION

1.1 THE PROC AND BATCH PROCESSORS

This manual describes two processors which are implemented on Microdata's Reality® Computer System: the PROC (stored procedure) processor and the BATCH update processor.

The PROC processor allows the user to prestore a complex sequence of Terminal Control Language (TCL) operations (and associated processor operations) which can then be evoked by a single word command. Any sequence of operations which can be executed at the TCL level can also be prestored via the PROC processor. This prestored sequence of operations (called PROC) is executed interpretively by the PROC processor and therefore requires no compilation phase.

The PROC processor has the following features:

Four variable length I/O buffers

Parameter passing between buffers

Interactive terminal prompting

Extensive I/O and buffer control commands

Conditional and unconditional branching to optional command labels

Relational character testing and pattern matching

File accesses and updating

Free-field and fixed-field character manipulation

Formatted screens and printouts

PROC calculations

Inter-PROC linkage and subroutines

The BATCH processor provides a facility for inputting, updating, verifying, and deleting items (or attributes) within Reality files. The BATCH processor operates via a predefined "BATCH-string" and a subsequent input line. The BATCH-string provides the dictionary function for the file update. In other words, the BATCH processor ignores the attribute defining items defined for the designated files (i.e., attributes which are used by the ENGLISH® language processor), and instead relies on the BATCH-string to define the updating algorithm. This processor is largely replaced by DATA/BASIC, but is maintained for compatibility.

The BATCH processor includes the following features:

- Simultaneous update of multiple files
- Reversal of update function via same BATCH-string
- Input prompting
- Free-field and fixed-field character referencing
- BATCH-lock capability protecting against concurrent BATCH-string access
- Interactive update capability when used in conjunction with a PROC

It is assumed that the user has read and understood the document titled "INTRODUCTION TO REALITY" prior to referencing this manual. A general understanding of the material covered in the "REALITY PROGRAMMER's REFERENCE MANUAL" will also prove useful in comprehending the finer points of this manual.

1 INTRODUCTION

1.2 HOW TO USE THIS MANUAL

> This manual is written in modular format with each pair of facing pages presenting a single topic.

The approach in this and other Reality manuals differs substantially from the typical reference manual format. Here each pair of facing pages discusses an individual topic. Generally the left-hand page is devoted to text, while the right-hand page presents figures referred to by the text. At the head of each text page are a pair of titles, the first one naming the section and the second one naming the topic. Immediately below these titles is a brief summary of the material covered in the topic.

The advantage of this format will become readily apparent to the reader as he begins to use this manual. First of all, the figures referred to in the text are always conveniently right in front of the reader at the point where the reference is made. Secondly, there is a psychological advantage to the reader in knowing that, whe he has completed reading a topic and goes to turn the page, he is done with one idea and ready to encounter a new one.

Subsequent sections of this manual describe various PROC and BATCH elements. In presenting general formats and examples relating to these elements, certain conventions apply. Conventions used in presenting general formats are listed in Figure A, while conventions used in examples are listed in Figure B.

Marginal change bars are for the convenience of present Reality users and indicate significant additions or changes from prior Reality publications.

CONVENTION	MEANING
UPPER CASE	*Characters or words printed in upper case are required and must appear exactly as shown.*
lower case	*Characters or words printed in lower case are parameters to be supplied by the user (e.g., file name, column number, data, etc.).*
{}	*Braces surrounding a word and/or parameter indicate that the word and/or parameter is optional and may be included or omitted at the user's option.*
{}...	*If an elipses (i.e., three dots) follows the terminating bracket, then the enclosed word and/or parameter may be omitted or repeated an arbitrary number of times.*

Figure A. Conventions Used in General Formats

CONVENTION	MEANING
TEXT	*Shaded text represents the user's input.*
TEXT	*All other text represents output printed by the system.*
(CR)	*This symbol represents a carriage return.*

Figure B. Conventions Used in Examples

(Significant textual changes are indicated by a vertical bar such as the one at the right.)

Figure C. Change Bar Format

2 THE PROC PROCESSOR

2.1 AN INTRODUCTION TO PROC'S

> An integral part of the Reality Computer System is the ability to define stored procedures called PROC'S.

A PROC provides the applications programmer a means of cataloging a highly complex sequence of operations which can then be evoked from the terminal by a one word command. Any operation that can be executed by the Terminal Control Language can be performed in a PROC. This usage of a PROC is quite similar to the use of a Job Control Language (JCL) in some large-scale computer systems. The PROC language in Reality, however, is more powerful since it has additional capabilities, and can be used to interactively prompt the terminal user. Additionally, a PROC can test and verify input data as they are entered from the terminal keyboard. Furthermore, the ability to access and update files from PROC makes PROC an exceedingly flexible tool.

A PROC is stored as an item in a dictionary or data file. The first attribute (first line) 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 attribute values contain PROC statements that serve to generate TCL commands or insert parameters into a buffer for the interactive processors, such as the EDITOR or the BATCH processor. PROC statements consist of an optional numeric label, a one- or two-character command, and optional command arguments.

Typically, PROC's are created using the EDITOR. PROC's are executed interpretively by the PROC processor and therefore require no compilation phase. (An optional compilation can be used to speed up PROC's.) A PROC stored as an item in the user's Master Dictionary (M/DICT) is executed in the TCL environment by typing the item-ID of the PROC, any optional arguments, and a carriage return. This is illustrated in Figure A where the sample PROC named LISTU is evoked. The ability to pass arguments to a TCL level process via a PROC is illustrated in Figure B. Here LISTDICTS is the name of the pre-stored PROC, while POLICY is the argument being passed.

The ability to interactively prompt input data from the user (and subsequently verify these data) is illustrated in Figure C. Here the PROC named ENTER-DATA is evoked. The PROC then prompts the user for the required data. The PROC could then, for example, store these data in a buffer which would then be passed to the BASIC processor to update the file.

Once a PROC is evoked, it remains in control until it is exited. When the PROC temporarily relinquishes control to a processor such as the EDITOR or a user-supplied subroutine, it functionally remains in control since an exit from the called processor returns control to the PROC. TCL only regains control when the PROC is exited explicitly, or when all of the lines in the PROC have been exhausted.

```
:LISTU (CR)

CHANNEL   PCB-FID   NAME......   DATE.......  TIME..

THREE     0260      KARDEX       14 SEPT 1977 12:38

ELEVEN    0360      EARL         14 SEPT 1977 14:01

TWO       0240      EARL         14 SEPT 1977  14:02

ZERO      0200      SYSPROG      14 SEPT 1977  14:15
```

Figure A. Sample PROC Execution

```
:LISTDICTS  POLICY (CR)

POLICY.............  D/CODE..  A/AMC.  V/CONV.........  V/TYP  V/MAX

AUDIT-PERIOD         A         01                       L      4
POLICY-PERIOD-FROM   A         02      D                L      10
POLICY-PERIOD-TO     A         03      D                L      11
EXPIRES              A         03      D                L      12
```

Figure B. Sample PROC Execution (Argument Passing)

```
:ENTER-DATA (CR)

PART-NUMBER                =  3215-19 (CR)

DESCRIPTION                =  TRANSISTOR (CR)

QUANTITY                   =  FIFTY (CR)

ERROR:NUMERIC DATA ONLY!!

QUANTITY                   =  50 (CR)
```

Figure C. Sample PROC Execution (Interactive Prompting)

2.2 INPUT/OUTPUT BUFFER OPERATION

Operations specified within a PROC involve the movement of data from either of two input buffers to either of two output buffers.

PROC's utilize four input/output buffers: the primary input buffer, the secondary input buffer, the primary output buffer, and the secondary output buffer (called the stack). See Figure A. Essentially, the function of a PROC is to move data from either input buffer to either output buffer, thus forming the desired TCL and processor commands.

The primary input buffer contains the PROC name and any optional arguments, exactly as they were entered when the PROC was evoked. The contents of this buffer remain the same whenever processors are called. This buffer is typically used for terminal input, and for storing parameters which control the sequence of PROC execution. Commands in DATA/BASIC allow this buffer to be altered.

The primary output buffer holds the command which ultimately is submitted at the TCL level for processing. Any command which can be executed via the terminal at the TCL level can also be constructed and executed via a PROC.

The secondary input buffer contains data input from the user in response to an IN command. The data in this buffer are volatile and are overwritten by subsequent IN commands. This buffer has fallen into disuse, but is maintained for compatibility.

When all desired data have been moved to the secondary output buffer, control is passed to the primary output buffer via a P or PP command. The command which resides in the primary output buffer is executed at the TCL level and the data in the secondary output buffer (if any) is used to feed processors such as BATCH or EDITOR. When the process is completed, control returns to the PROC, at which time new data may be moved to the output buffers.

Data in the input and output buffers is in the form of a series of parameters separated by blanks or attribute marks. References to a parameter in one of these buffers is made via the sequential number of that parameter within the buffer or by the starting character within the buffer. Thus we refer to the sixth parameter in the primary output buffer or the data starting in column 16 of the secondary input buffer (counting from "1" in both cases).

There are three pointers, one into the input buffers and two into the output buffers. These are indicated by ↑ in Figure A. The pointer into the input buffers points into whichever is "active" (next paragraph). This pointer may be moved back and forth without affecting the data in the buffer. Each output buffer has its own pointer which always points to the end of data in those buffers. If the pointer is moved backward, the data to the end of the buffer will be cleared.

At any given time, one of the input buffers is specified as "currently active", and one of the output buffers is specified as "currently active." The input pointer always points into the input buffer which is active. When the PROC is entered, or after the TCL process returns to the PROC, the primary input buffer is active and the primary output buffer is active. Having the "stack on" or the "stack off" is an alternate way of referring to the secondary output buffer being active versus the primary output buffer being active. Buffers are selected as "currently active" via certain PROC commands. When data is moved by the A command, it is moved from the position of the input pointer to the position of the output pointer in the buffer which is active.

As an illustration of this concept, consider Figure A. Here the PROC has been evoked by the characters ABC XYZ, which are then automatically placed in the primary input buffer. PROC commands have then been processed which position the input pointer of the primary input buffer to the second parameter (XYZ), and then subsequently move that parameter to the primary output buffer as illustrated in Figure A. In this example the currently active buffers are the primary input buffer and the primary output buffer.

The user should note that each line of data placed in the secondary output buffer must be terminated by a carriage return which is explicitly placed in the stack as part of an H command (refer to the topic describing that command). This is not the case with the primary output buffer; a carriage return is automatically placed at the end of the TCL command in the primary output buffer upon execution of that buffer via the P or PP command.

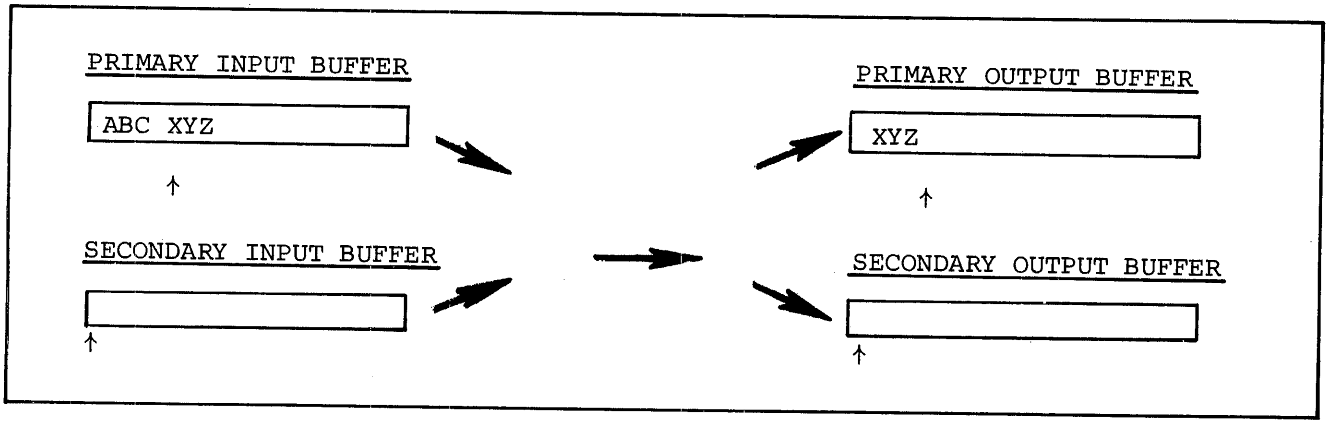


Figure A. Sample Inter-Buffer Transfer With Both Primary Buffers Currently Active

2.3 BUFFER DELIMITERS

PROC enhancements required a change in the buffer delimiter from blank to attribute mark. The old commands use blank to maintain compatibility.

Recent enhancements to the PROC processor have required the capability to delimit parameters with some character other than blank. The attribute mark has now been implemented as the new buffer delimiter. The major advantage is the ability to have null parameters, parameters which contain imbedded blanks, and parameters which match the format of REALITY file items.

In order to provide upward compatibility, the original PROC commands which use the blank as the parameter delimiter have not been changed. New commands (type 'N') have been added to take advantage of the attribute mark as the parameter delimiter. Commands which cause the buffer pointer to move leave the pointer pointed to an attribute mark. While combinations of the original commands and the new commands are allowed, the two sets are designed to work as two distinct alternatives.

See Figure A for some comparisons of the commands.

A *Moves from the first non-blank to the first blank.*

NA *Moves from one attribute mark to the next including imbedded blanks.*

F *Moves the pointer forward to the first character of the next parameter.*

NF *Moves the pointer forward to the next attribute mark.*

B *Moves the pointer backward to the first character of the previous parameter.*

NB *Moves the pointer backward one attribute mark.*

IP *Moves data into the current input buffer location with imbedded blanks preserved.*

NIP *Moves data into the current input buffer location but replaces one or multiple consecutive blanks with a single attribute mark.*

IH *Moves the character string which follows into the current pointer position.*

NIH *Moves the character string which follows into the current pointer position, replacing each set of one or more blanks with a single attribute mark.*

BO *Moves the output buffer pointer back to the first character of the previous parameter.*

NBO *Moves the output buffer pointer back to the previous attribute mark.*

Note: All positions are relative to the applicable buffer pointers.

Figure A. Comparison of Original and New Commands

2.4 FILE BUFFERS AND SELECT REGISTERS

The nine file buffers are used for reading and writing records. The five select registers are used to hold the results of SELECTS or SORT SELECTS.

The data within a file buffer is divided into attributes separated by attribute marks. Reference to an attribute beyond those already established will cause automatic construction of the required number of additional attributes.

File buffers are referenced by using an ampersand (&) followed by the file buffer number, a period (.) and a numeric value. The period is used as a separator and the numeric value is the attribute number; 0 references the item-ID. File buffers may not be used until opened. See the section on file commands.

The maximum size of a file buffer is 32,267 bytes.

Select registers are used when processing the results of a SELECT or SSELECT, or for handling multi-valued attributes. Any of the five select registers may contain only one attribute at any time. The attribute may be of any size and may contain multi-values. See the topic SPECIAL VERBS FOR USE WITH PROC: THE PQ-COMPILE AND PQ-SELECT VERBS.

The select registers are referenced using an exclamation mark (!) followed by the number of the register. Each reference to a select register obtains the next value (item-ID) for processing. If a value is to be used more than once, it should be moved to another buffer.

Any new values added to a select register cause all previously existing values to be deleted.

Figure A presents some examples of the file buffer concept. Figure B presents an example of the general form for select register reference.

PARAMETER #	BUFFER #1	BUFFER #2 ...	BUFFER #n
0	1234	100	...
1	SMITH	1234	...
2	1260	480	...
3	487-30-7914	80	...
4	12 SEPT 44	3235	...
5	M	3242	...
.	...	...	...
.	...	...	...
N	...	...	...

&1.1	*References the value "SMITH" in buffer #1.*
&1.4	*References the value "12 SEPT 44" in buffer #1.*
&2.3	*References the value "80" in buffer #2.*
&2.0	*References the value "100" in buffer #2.*

&F.A

& = File buffer reference.
F = File buffer number.
. = Separator.
A = Attribute number referenced.

Figure A. File Buffers Concept

!n ! = *Select register reference.*
n = *Select register number.*

SELECT REGISTER CONTENTS

BLUE]BROWN]GREEN]YELLOW	*1st ref (!n) obtains "BLUE"*
BROWN]GREEN]YELLOW	*2nd ref (!n) obtains "BROWN"*
GREEN]YELLOW	*3rd ref (!n) obtains "GREEN"*
YELLOW	

Figure B. General Form for Select Register Reference

2.5 BUFFER REFERENCING, DIRECT AND INDIRECT

Direct and indirect reference may be made to input, output and file buffers, and to select registers.

Four characters are reserved to indicate the buffer referenced:

% - Reference the primary input buffer
- Reference the currently active output buffer (see ST command)
& - Reference a file buffer
! - Reference a select register.

Direct referencing of the buffers is accomplished by following the appropriate buffer character with literal(s). See Figure B.

An indirect reference can be made by following the buffer reference character(s) with the input or output buffer symbol and literal. The indirect reference may only be from the primary input buffer (%) or the current output buffer (#). If the value referenced is non-numeric, then zero is used. See Figure C.

Buffer references may be used with many PROC commands. For a detailed description, see the descriptions of the individual commands.

PROC commands which use buffer references are:

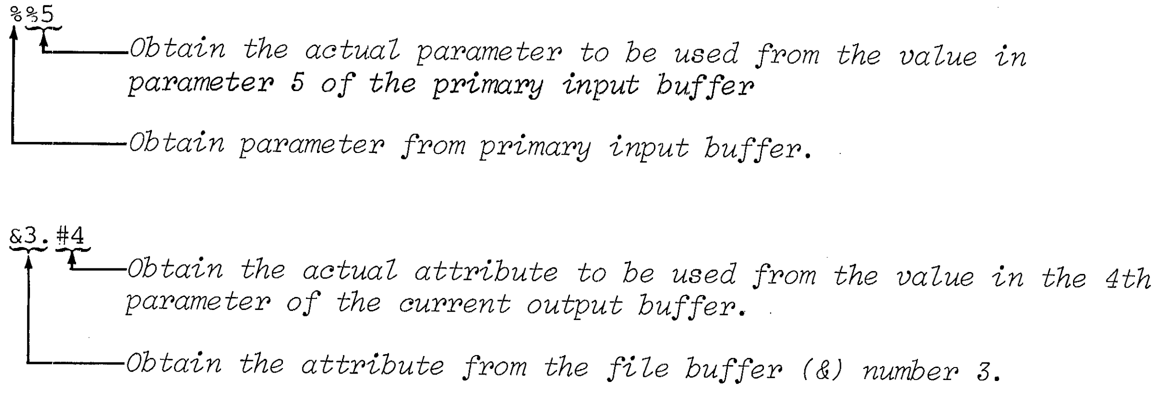
F-OPEN
F-READ
F-WRITE
F-CLEAR
F-DELETE
NIP
NIH
T
L
NH
MV
F;
IF

%n	*Obtain parameter "n" from primary input buffer.*
#n	*Obtain parameter "n" from "current" output buffer.*
&F.n	*Obtain attribute "n" from file buffer "F".*
!n	*Obtain the next value from select register 'n'.*

Figure A. Buffer Reference Syntax

%23	*Obtain the 23rd parameter from the primary input buffer.*
#4	*Obtain the 4th parameter from the current output buffer.*
&4.19	*Obtain attribute 19 from file buffer 4.*
!3	*Obtain the next value from select register 3.*

Figure B. Direct Reference



%%5

Obtain the actual parameter to be used from the value in parameter 5 of the primary input buffer

Obtain parameter from primary input buffer.

&3.#4

Obtain the actual attribute to be used from the value in the 4th parameter of the current output buffer.

Obtain the attribute from the file buffer (&) number 3.

Figure C. Indirect Reference

2.6 AN OVERVIEW OF PROC COMMANDS

> A PROC consists of any number of PROC commands, one command per line.

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 may contain any valid PROC commands. There is no limit to the number of lines in a PROC. However, each line may contain only one command, and each command must begin in column position one of the line. **If a comment appears on the second line, it will be printed by the LISTPROCS verb.**

Many different PROC commands are provided. Typical commands are listed in Figure A. A complete description of each command is presented in the remaining topics within this section.

Any PROC command may optionally be preceded by a numeric label. Such a label serves to uniquely identify its associated PROC command for purposes of branching or looping within the PROC. Labels may consist of any number of numeric characters (e.g., 5, 999, 72, etc.). When a label is used, the PROC command must begin exactly one blank beyond the label. For example:

```
1 GO 5
23 A
99 IF A = ABC GO 3
2 ST ON
```

As an introductory example to PROC commands, consider the following PROC stored as item 'DISPLAY' in the user's M/DICT:

```
PQ
HLIST ONLY
A2
P
```

Assume that the user types in the following:

```
:DISPLAY INVENTORY  CR
```

This input evokes the above PROC and places the words DISPLAY INVENTORY in the primary input buffer. The second line of the above PROC is an H command which causes the text LIST ONLY to be placed in the primary output buffer. The third line is an A command which picks up the second word (parameter) in the primary input buffer and places it in the primary output buffer. Thus the primary output buffer contains the words LIST ONLY INVENTORY. The last line of the PROC is a P command which submits the content of the primary output buffer to TCL for processing (i.e., LIST ONLY INVENTORY is an ENGLISH sentence which causes the item-ID's of the INVENTORY file to be listed; refer to the ENGLISH Reference Manual).

COMMAND	BRIEF DESCRIPTION
A, NA	*Moves data from input to output buffers.*
B, NB	*Back up input pointer.*
BO, NBO	*Backup up output pointer.*
C	*Specifies comment.*
D	*Outputs from either input buffer to terminal.*
F, NF	*Moves input pointer forward.*
F;	*Calculates a function.*
G, GO	*Unconditionally transfers control.*
GOSUB, RSUB	*Call local subroutine, return.*
H, NH	*Moves text string to either output buffer.*
IF	*Conditionally executes specified command.*
IH, NIH	*Moves text string to either input buffer.*
IN	*Inputs from terminal to secondary input buffer.*
IP, NIP	*Inputs from terminal to either input buffer.*
IT, NIT	*Inputs from tape to primary input buffer.*
L	*Formats spooler output.*
O	*Outputs text string to terminal.*
P	*Causes execution of PROC.*
PP	*Displays content of output buffers and executes PROC.*
RI	*Clears (resets) input buffer.*
RO	*Clears (resets) both output buffers.*
RTN	*Return from PROC subroutine.*
S, NS	*Positions input pointer and optionally selects primary input buffer.*
ST ON	*Selects secondary output buffer (stack).*
ST OFF	*Selects primary output buffer.*
T	*Formats console output.*
TR	*Turns PROC trace on.*
U	*Exits to user-defined subroutine.*
X	*Exits back to TCL level.*
+	*Adds decimal number to parameter in input buffer.*
-	*Subtracts decimal number from parameter in input buffer.*
( ), [ ]	*Links to another PROC.*

Figure A. Summary of PROC Commands

2.7 SELECTING BUFFERS AND POSITIONING POINTERS: THE ST, S, NS, F, NF, B, NB, BO, AND NBO COMMANDS

> The STON and STOFF commands turn the stack on or off, respectively. The S command positions the input pointer and/or selects the primary input buffer as the currently active input buffer. The F and B commands move the input pointer forward or backward one parameter, respectively. The BO command moves the output backward one parameter.

The STON command selects the secondary output buffer (the stack) as the currently active output buffer (i.e., turns the stack on). Its form is:

```
STON or ST ON
```

The STOFF command selects the primary output buffer as the currently active output buffer (i.e., turns the stack off). Its form is:

```
STOFF or ST OFF
```

When the stack is on, all data picked up by the A command is moved to the secondary output buffer. When the stack is off, these data are moved to the primary output buffer. The stack may be turned on or off at any point within the PROC. Upon initial entry to a PROC, the stack is off.

The S command positions the input pointer and/or selects the primary input buffer as the currently active input buffer. This command may be used in either of the following two general forms:

```
S(m)   or  NS(m)
Sp     or  NSp
```

The S(m) form selects the primary input buffer and moves the pointer to the m'th column, with columns numbered starting at 1. The Sp form moves the input pointer to the p'th parameter of the currently active input buffer.

The F command causes the input pointer for the currently active input buffer to move forward one parameter. If the input buffer pointer is currently at the beginning of the buffer, this command has no effect. The form of the F command is as follows:

```
F  or  NF
```

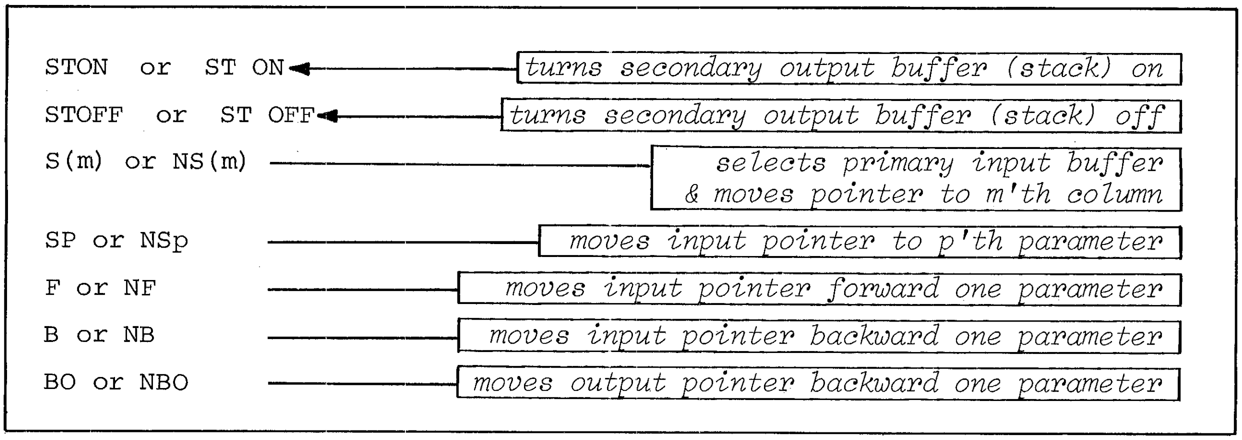
The B command causes the input pointer for the currently active input buffer to move backward one parameter. If the input buffer pointer is currently at the beginning of the buffer, this command has no effect. The form of the B command is as follows:

```
B  or  NB
```

The BO command causes the output pointer for the current output buffer to move backward one parameter. If the output buffer pointer is currently at the beginning of the buffer, this command has no effect. Moving the pointer back will cause data to be eliminated. The form of the BO command is as follows:

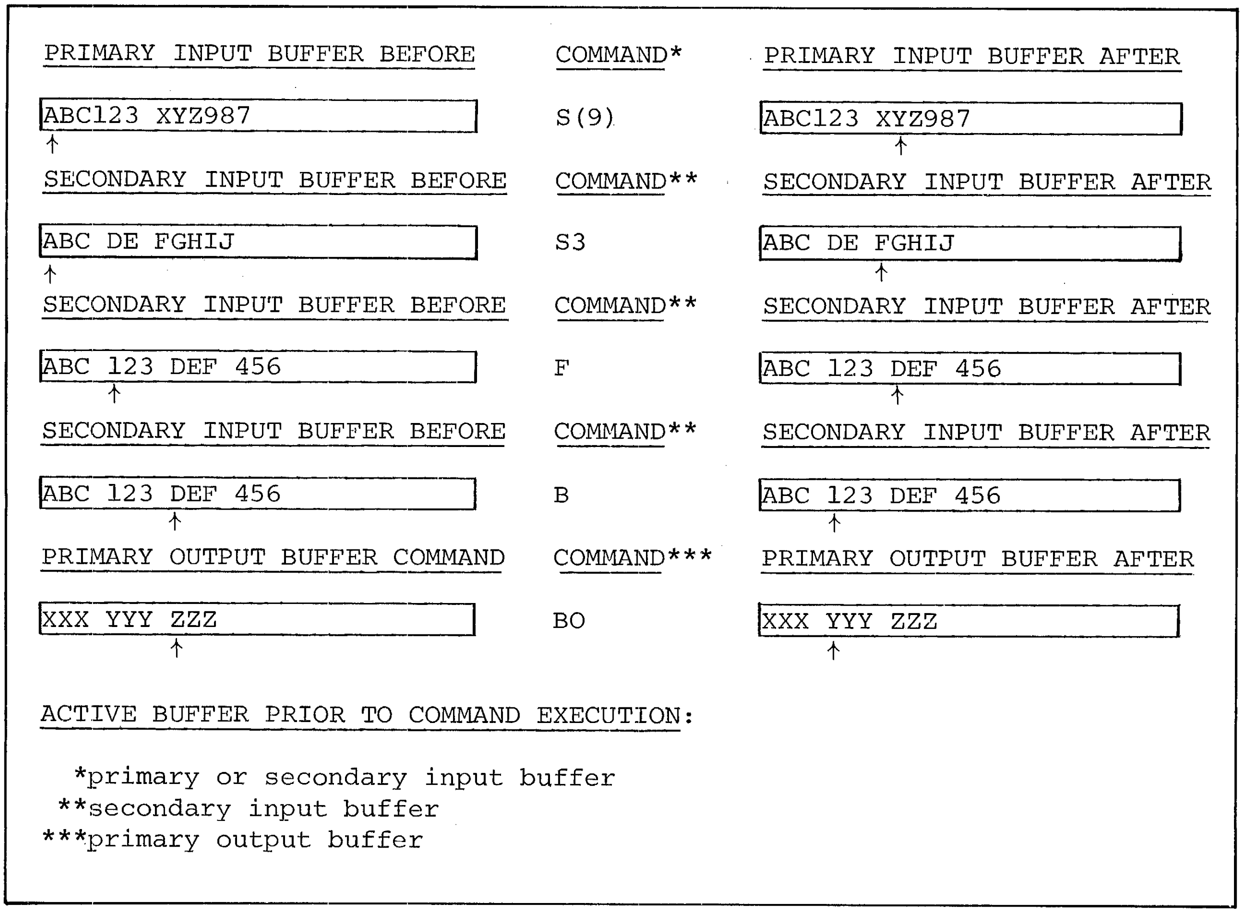
```
BO  or  NBO
```

Figure A summarizes the above commands. Figure B presents some examples.



STON or ST ON	*turns secondary output buffer (stack) on*
STOFF or ST OFF	*turns secondary output buffer (stack) off*
S(m) or NS(m)	*selects primary input buffer & moves pointer to m'th column*
SP or NSp	*moves input pointer to p'th parameter*
F or NF	*moves input pointer forward one parameter*
B or NB	*moves input pointer backward one parameter*
BO or NBO	*moves output pointer backward one parameter*

Figure A. General Form of ST, S, F, B, and BO Commands



```
PRIMARY INPUT BUFFER BEFORE      COMMAND*      PRIMARY INPUT BUFFER AFTER

ABC123 XYZ987                    S(9)          ABC123 XYZ987
↑                                                      ↑
SECONDARY INPUT BUFFER BEFORE    COMMAND**     SECONDARY INPUT BUFFER AFTER

ABC DE FGHIJ                     S3            ABC DE FGHIJ
↑                                                     ↑
SECONDARY INPUT BUFFER BEFORE    COMMAND**     SECONDARY INPUT BUFFER AFTER

ABC 123 DEF 456                  F             ABC 123 DEF 456
    ↑                                                  ↑
SECONDARY INPUT BUFFER BEFORE    COMMAND**     SECONDARY INPUT BUFFER AFTER

ABC 123 DEF 456                  B             ABC 123 DEF 456
        ↑                                          ↑
PRIMARY OUTPUT BUFFER COMMAND    COMMAND***    PRIMARY OUTPUT BUFFER AFTER

XXX YYY ZZZ                      BO            XXX YYY ZZZ
        ↑                                          ↑

ACTIVE BUFFER PRIOR TO COMMAND EXECUTION:

  *primary or secondary input buffer
 **secondary input buffer
***primary output buffer
```

Figure B. Sample Usage of S, F, B, and BO Commands

2.8 MOVING PARAMETERS: THE A OR NA COMMAND

> The A command is used to retrieve a parameter from the input buffer and move it to the output buffer. Either the primary or secondary input buffer may be used as the source, and either the primary or secondary output buffer may be used as the destination; the buffers used depend on commands executed prior to the A command. The NA form of this command uses an attribute mark as the parameter separator; otherwise NA operates the same as A.

The A command may be used in any of the following general forms:

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

If the parenthetical specification (i.e., m and/or n) *is not* used, then the parameter is obtained from the currently active input buffer, at the current position of the input pointer. Leading blanks are deleted from the parameter. The end of the parameter is designated by the first blank which is encountered.

If the parenthetical specification *is* used, then the parameter is always obtained from the primary input buffer. If the secondary input buffer is currently active, use of this option causes a switch back to the primary input buffer. If both m and n are specified (i.e., in the A(m,n) form), the input buffer pointer is set to the m'th column, and the next n characters (including any embedded blanks) is the parameter to be moved. The m specification may be used by itself in the form A(m). In this case the parameter is obtained from the m'th column and continues until the first delimiter after column m is encountered. The n specification may also be used by itself in the form A(,n). In this case the next n characters are moved, starting at the current position of the input pointer.

When the form Ap is used, the p'th parameter is moved, where parameters are separated by blanks, or attribute marks for NAp.

When the form Ac is used (where c is any non-numeric character except a left-parenthesis character) and the primary output buffer is active (i.e., stack is off), the character c surrounds the parameter. This feature is useful for picking up item-ID's (which require single quotes) and values (which require double quotes) for processing by the ENGLISH language processor. If not specified, the A command uses a blank as the default surround character.

Multiple parameters may be moved to the primary output buffer via a *single* A command if these parameters are separated by semicolons in the input buffer. The parameters will be moved to the primary output buffer with the semicolons replaced by blanks (or attribute marks). After the execution of an A command, the input buffer pointer is pointing to the very next character after the move.

Figure A summarizes the general form of the A command. Figure B presents a number of examples. Each example assumes that the output pointer is at the beginning of the buffer prior to the illustrated operation.

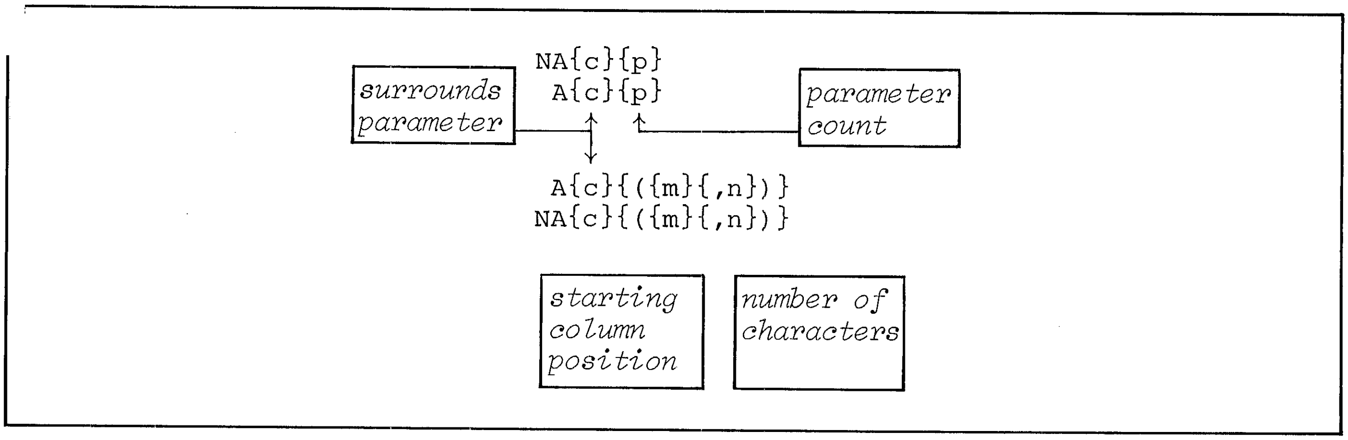
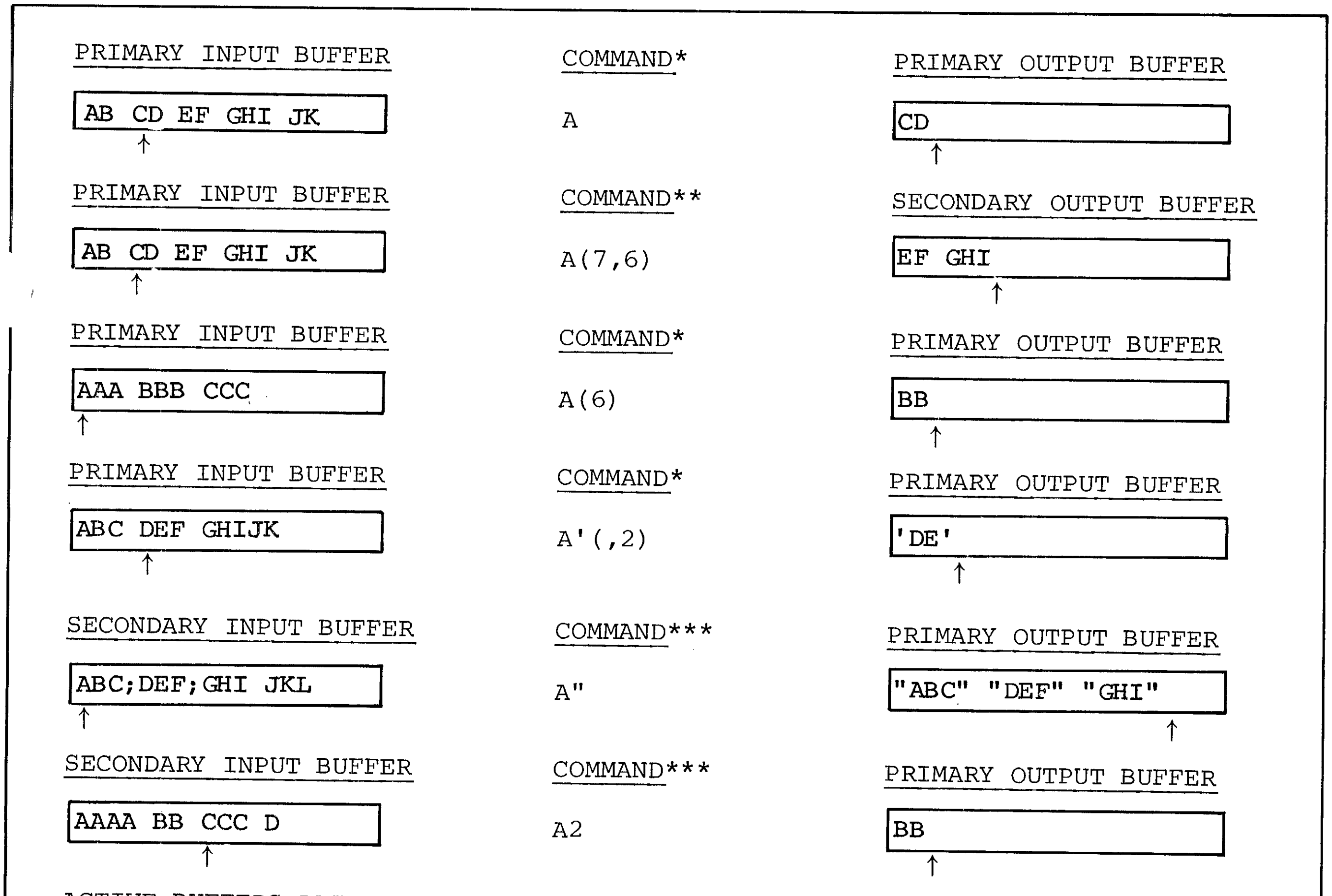


Figure A. General Form of A and NA Commands



ACTIVE BUFFERS PRIOR TO COMMAND EXECUTION:

*primary or secondary input; primary output
**primary or secondary input; secondary output
***secondary input; primary output

Figure B. Sample Usage of A Command

2.9 MOVING PARAMETERS: THE MV COMMAND

The MV command can be used to move data from any buffer or register to any buffer or register.

The MV command always has two operands. The first is the object of the move or the receiving field. The second is the source or the sending field. The receiving field may be any direct or indirect buffer reference. The sending field may be any direct or indirect buffer reference, or a single value, a string of values separated by commas, or a concatenation of values (see Figure A).

The simplest form is moving one value into the receiving field (Figure B).

A string of values may be moved into successive buffer locations by separating each value with a comma. If the sending field contains multiple contiguous commas, the effect is to *not* modify that number of buffer locations in the receiving field (Figure C). If an asterisk is used to separate values, then concatenation occurs.

An asterisk (*) as the last item in a string of values will move all remaining values in the sending field to successive receiving buffer locations. If the form n* is used, then n additional values will be moved.

The MV command will construct the required number of null attributes if the receiving field is beyond the current end of the buffer.

NOTE: *The MV command recognizes an attribute mark as the separator for buffer parameters.*

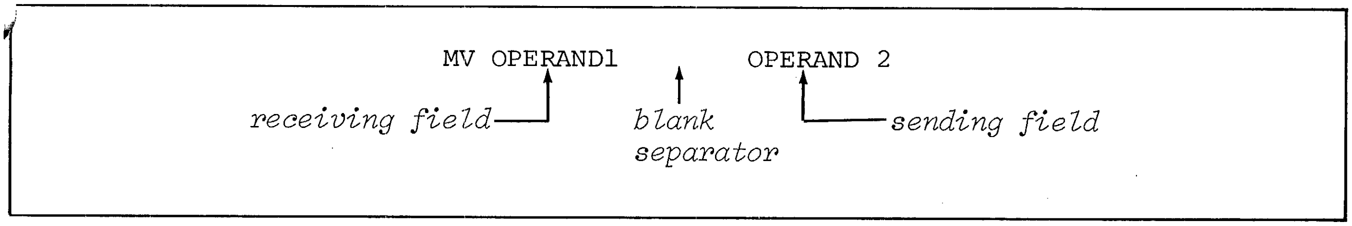


Figure A. General Form of the MV Command

MV &3.1 &5.2	*Move value from file buffer 5, parameter 2 to file buffer 3, parameter 1.*
MV %4 "SMITH"	*Move the literal "SMITH" to the 4th parameter of the primary input buffer.*
MV #1 %5	*Move value from primary input buffer parameter 5 to the 1st parameter of the current output buffer.*
MV !3 &3.18	*Move attribute 18 of file buffer 3 into select register 3.*
	NOTE: The single value move (MV) is the only valid form when a select register is specified in operand 1.

Figure B. Simple Forms of MV Command

MV &1.1 %3,%7	*Move the 3rd and 7th parameters from the primary input buffer into file buffer 1, parameter 1 and parameter 2. (Consecutive receiving fields.)*
MV &2.1 %3,%7,"SAM",#8	*Move the 3rd and 7th parameters from the primary input buffer, the literal "SAM" and the 8th parameter of the current output buffer into file buffer 2, parameters 1,2,3, and 4 respectively.*
MV %4 %2*%7	*Concatenate (*) the 2nd and 7th parameters of the primary input buffer and move into parameter 4 of the primary input buffer (%).*
MV &1.0 &2.0,*	*Move from file buffer 2 all parameters (*) into file buffer 1 starting in parameter 0.*
MV &9.4 "X",,,,"Y"	*Move the literal "X" into file buffer 9, parameter 4. Skip (don't modify) parameters 5,6,7 then move literal "Y" into parameter 8.*

Figure C. Complex Forms of the MV Command

2.10 MOVING PARAMETERS: THE MVA AND MVD COMMANDS

The MVA and MVD commands add and delete values in multi-valued attributes within the file buffers.

The MVA and MVD commands always contain two operands separated by a blank (see Figure A). Operand one *must* be a file buffer reference. Operand two may be any single value (i.e., a direct or indirect reference or a literal).

The MVA command moves the value from the sending field into the receiving field and stores it as a multi-value. The new value is stored in ascending ASCII sequence. If the value already exists, it will not be duplicated. If the sending field is itself multi-valued, it is stored as *one* multi-valued set and may, therefore, create duplicate values and destroy the ascending sequence of the attribute.

The MVD command deletes the value specified in the sending field from the values in the receiving field. If the receiving field is *not* in ascending ASCII sequence, the MVD command will not function properly.

NOTE: *The MVA and MVD commands use an attribute mark as the parameter separator.*

```
                MVA     OPERAND1 OPERAND2
                MVD     OPERAND1 OPERAND2
                           ↑        ↑
      receiving field──────┘        └──────sending field
```

Figure A. General Format of the MVA and MVD Commands

```
 BEFORE:        &1.0        [ABC↑DEF]GHI↑JKL↑FOR↑←

COMMAND:        MVA &1.1 &1.3

  AFTER:        &1.0        [ABC↑DEF]FOR]GHI↑JKL↑FOR↑←


 BEFORE:        &1.0        [ABC DEF]GHI↑JKL↑MNO↑←
                &3.0        [AAA BBB EXTRA]MONEY]WAGE↑END↑←

COMMAND:        MVA &1.1 &3.2

  AFTER:        &1.0        [ABC↑DEF]EXTRA]MONEY]WAGE]GHI↑JKL↑MNO←


 BEFORE:        &2.0        [ABC↑DEF]GHI]JKL↑MNO↑←

COMMAND:        MVD &2.1 GHI

  AFTER:        &2.0        [ABC↑DEF]JKL↑MNO↑←
```

Figure B. Examples of MVA and MVD Commands

2.11 INPUTTING DATA: THE IN, IP, IT, NIN, NIP AND NIT COMMANDS

The IN command selects the secondary input buffer and accepts input from the terminal. The IP command accepts input from the terminal to the currently active input buffer. The IT command accepts input from magnetic tape to the primary input buffer. The commands NIN, NIP and NIT use attribute marks as parameter delimiters but otherwise function identically to their counterparts.

The IN command selects the secondary input buffer as the currently active input buffer and inputs data from the terminal into the buffer (the original contents are lost). The general form of this command is:

```
IN{r}  or   NIN{r}
```

If the r specification is used, then that character is a prompt character at the terminal (r may be any character including a blank). The prompt character will remain in effect until a new IN command with a new r specification is executed. If r is omitted, then a colon (:) is used as a prompt. Data input by the user is placed into the secondary input buffer and may be moved to an output buffer by using the A command. However, when the primary input buffer is selected via the "A" command, or if the S(m) command is used, then the data in the secondary input buffer is lost.

The IP command inputs data from the terminal into the currently active input buffer. The general form of this command is:

```
IP{B}{r}   or   NIP{r}
```

Data input at the terminal in response to an IP command replaces the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer. If several parameters are input at the terminal, then they will all replace the current parameter in the buffer. If the input pointer is at the end of the data in the input buffer, then the new input data will be appended to the end. The r specification is identical to the r specification for the IN command (see above). If B is used in the command (i.e., IPB or IPBr), then all embedded blanks in the data input at the terminal will be replaced by backslash characters (\). This feature is useful if a parameter is to include embedded blanks (i.e., since blanks denote the end of a parameter).

The IT command reads one record (maximum size 300 characters) from the magnetic tape unit. The data are placed in the primary input buffer. (The original content of the primary input buffer is lost.) The general form of the IT command is:

```
IT{C}{A}   or   NIT{C}{A}
```

If the C option is used, an EBCDIC to ASCII conversion is performed on the record. If the A option is used, the 8-bit ASCII characters are masked to 7-bits (i.e., MSB=0). A null or all-blank record constitutes an end-of-file condition. (For further information regarding magnetic tape operations, refer to the Reality Programmer's Reference Manual.)

The general forms of the above commands are summarized in Figure A. Figure B presents a number of examples.

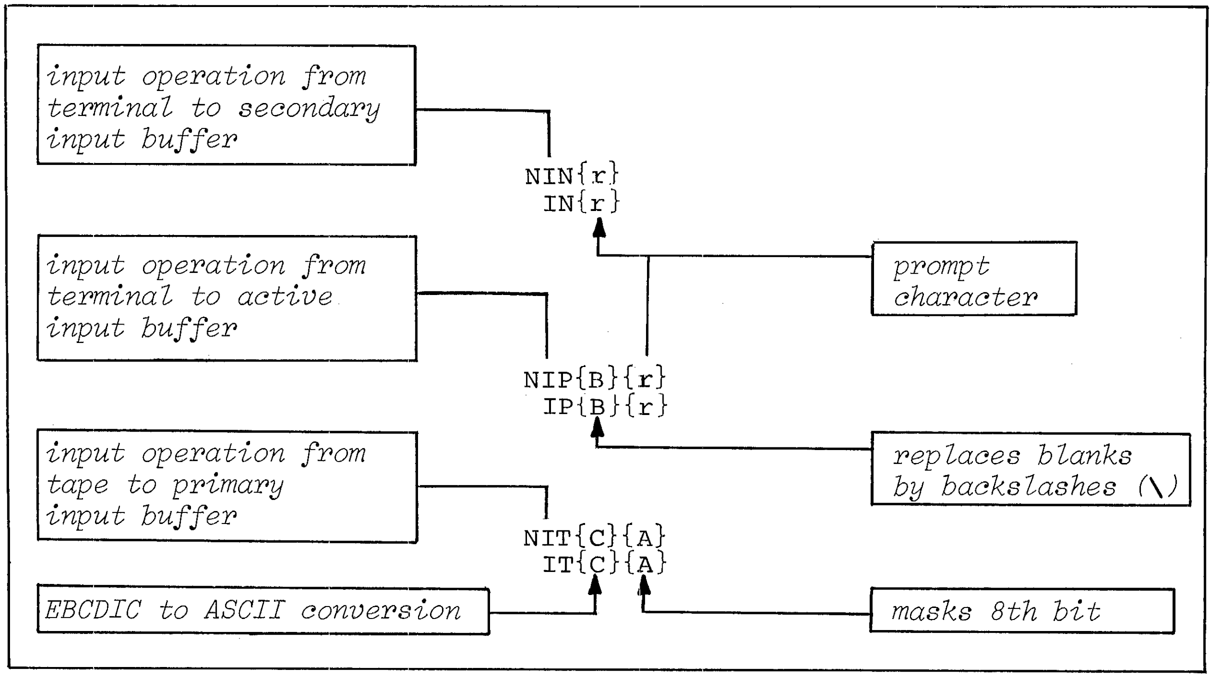

input operation from terminal to secondary input buffer
NIN{r}
IN{r}
input operation from terminal to active input buffer
prompt character
NIP{B}{r}
IP{B}{r}
input operation from tape to primary input buffer
replaces blanks by backslashes (\)
NIT{C}{A}
IT{C}{A}
EBCDIC to ASCII conversion
masks 8th bit

Figure A. General Form of IN, IP, and IT Commands

COMMAND	EXPLANATION
IN	*Selects secondary input buffer and inputs data from terminal. Prompt character is a colon (:).*
IN=	*Selects secondary input buffer and inputs data from terminal. Prompt character is an equal sign (=).*
IP?	*Replaces current parameter in currently active input buffer with data from terminal. Prompt character is a question mark (?).*
IPB?	*Same as above, but embedded blanks in input data are replaced by backslash characters (\).*
ITCA	*Inputs tape record to primary input buffer. EBCDIC to ASCII conversion is performed, with 8-bit ASCII masked to 7-bits.*

Figure B. Sample Usage of IN, IP, and IT Commands

2.12 OUTPUTTING DATA: THE O AND D COMMANDS

> The O command is used to output a specified text string to the terminal. The D command is used to output parameters from either input buffer to the terminal.

The O command has the following general form:

```
O{text}{+}
```

The O command causes the text which immediately follows the O to be output to the terminal. If the last character of the text is a plus sign (+), then a carriage return will not be executed at the end of the text output. This feature is useful when using the O command in conjunction with the IN command. For example, consider the following commands:

```
OPART-NUMBER+
IN=
```

These commands produce the following output on the terminal:

```
PART-NUMBER=
```

The specified prompt character (=) is displayed adjacent to the output text since the O command ended with a plus sign (+). The user then enters the input data right after the prompt character. For example:

```
PART-NUMBER=115020
```

Figure B illustrates further examples of the O command.

The D command is used to output parameters from either input buffer to the terminal. The D command may be used in either of the following general forms:

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

If the form Dp is used, then the p'th parameter of the currently active input buffer is displayed on the terminal. If the form D is used, then the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer is displayed on the terminal. If the form DØ is used [i.e., p = 0 (zero)], the complete currently active input buffer is displayed on the terminal. If the form D(m,n) is used, then the characters starting at the m'th column for n characters (including separators) are displayed. If the form D(m) is used, then the characters starting at the m'th column (blanks are included in the column count) of the primary buffer (up to the first blank character encountered) are displayed.

A plus sign (+) may be appended to the end of the D command, thus specifying the suppression of a carriage return (as for the O command described above.) The D command does not affect the input pointer.

Figure C illustrates the use of the D command.

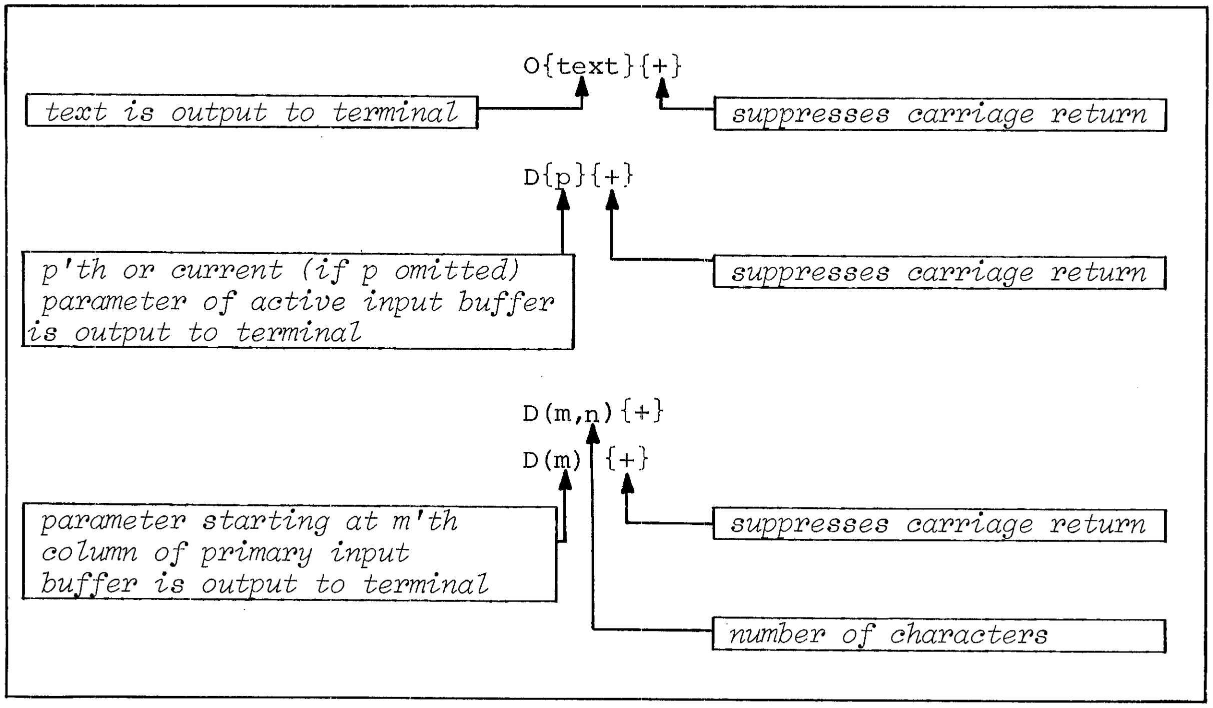

O{text}{+}
text is output to terminal
suppresses carriage return
D{p}{+}
p'th or current (if p omitted) parameter of active input buffer is output to terminal
suppresses carriage return
D(m,n){+}
D(m) {+}
parameter starting at m'th column of primary input buffer is output to terminal
suppresses carriage return
number of characters

Figure A. General Form of O and D Commands

COMMAND	OUTPUT TO TERMINAL
OTHIS IS AN EXAMPLE	THIS IS AN EXAMPLE –
OTHIS IS AN EXAMPLE+	THIS IS AN EXAMPLE_

Figure B. Sample Usage of O Command (– = The Cursor)

PRIMARY INPUT BUFFER	COMMAND*	OUTPUT TO TERMINAL
AA BBB CC DDD ↑	D	BBB
SECONDARY INPUT BUFFER	COMMAND**	OUTPUT TO TERMINAL
AA BBB CC DDD ↑	D4+	DDD
PRIMARY INPUT BUFFER	COMMAND***	OUTPUT TO TERMINAL
ABC XYZ 123 ↑	D(6)	YZ

ACTIVE BUFFER PRIOR TO COMMAND EXECUTION:
 *primary input buffer
 **secondary input buffer
***primary or secondary input buffer

Figure C. Sample Usage of D Command

2.13 TEXT STRINGS AND CLEARING BUFFERS: THE IH,NIH,H,NH,RI AND RO COMMANDS

The IH or NIH and H or NH commands are used to place a specified text string in the currently active input or output buffer, respectively. The RI and RO commands are used to reset the input and output buffers (respectively) to the empty (null) condition.

The IH (input Hollerith) command causes the text (including any blanks) immediately following the IH to replace the current parameter (as specified by the input pointer) in the active input buffer. The input buffer pointer will remain pointing to the beginning of this string.

The NIH command will input a string into the primary input buffer at the location specified by the current position of the input buffer pointer. The string will replace the attribute immediately following the pointer. A space appearing within the string being input will be converted to an attribute mark. Use of the NIH command with only a backslash following (as NIH\) will be used to null an attribute, and all appearances of a space-backslash-space sequence within the string will establish a null attribute. Certain special characters may not appear in an NIH literal string. These are the value mark (M[cs]) and the sub-value mark (L[cs]). If any invalid characters do appear, the resultant values loaded into the buffer will be unpredictable.

The H command causes the text (including blanks) which follows the H to be placed in the active output buffer at the position of the output pointer. When the last parameter has been moved to the secondary output buffer (the stack), a carriage return specification (<) must be placed in the stack. Two less than signs (<<) appearing as the last two characters in an H command will be recognized as a continuation character and a carriage return. This is useful for stacked information containing more than 140 characters.

The NH command uses an attribute mark as the parameter delimiter instead of a blank as in the H command and is otherwise similar.

The RI (Reset Input Buffers) command has three forms. If the form RI is used, then both input buffers are reset to the empty (null) condition. If the form RIp is used, the primary input buffer from the p'th parameter to the end of the buffer (as well as the entire secondary input buffer) is reset to the empty condition. If the form RI(m) is used, then the primary input buffer from the m'th column to the end of the buffer (as well as the entire secondary input buffer) is reset to the empty condition. The input buffer pointer remains at the end of the buffer.

The RO (Reset Output Buffers) command resets both output buffers to the empty (null) condition. The RO command also selects the primary output buffers as though an STOFF command had been executed.

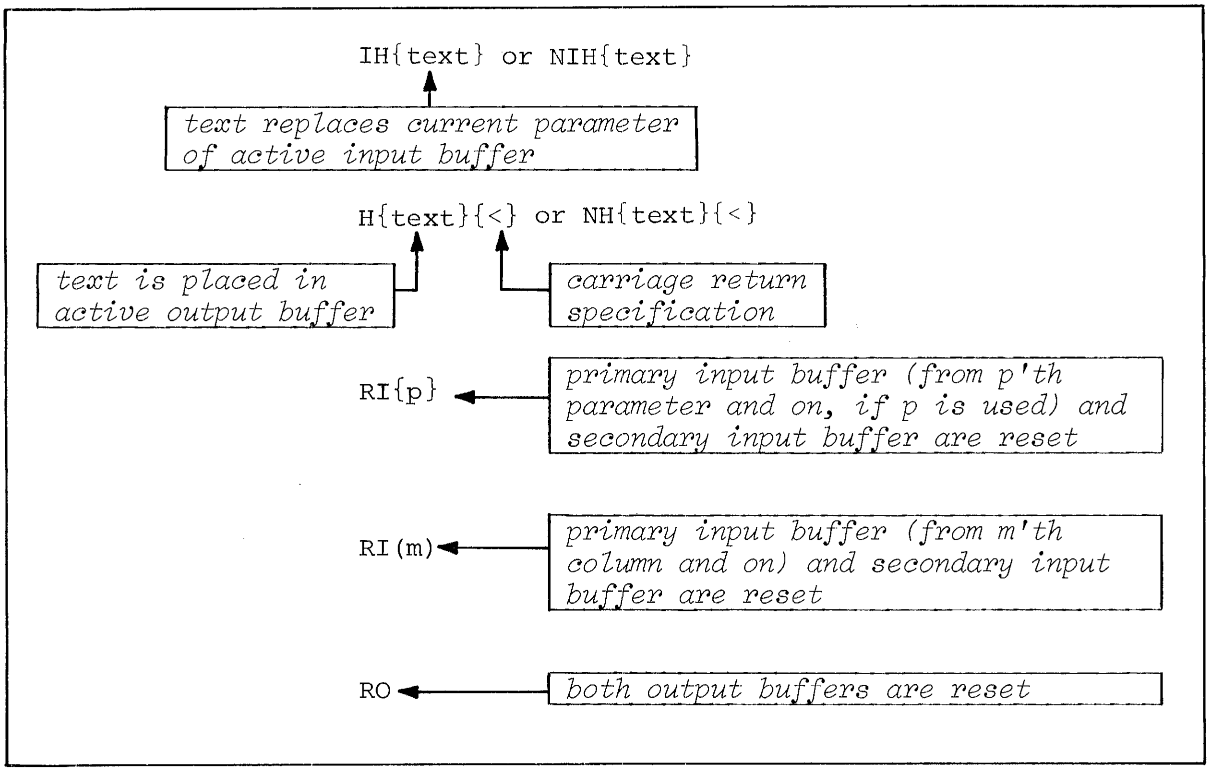

IH{text} or NIH{text}
text replaces current parameter of active input buffer
H{text}{<} or NH{text}{<}
text is placed in active output buffer
carriage return specification
RI{p}
primary input buffer (from p'th parameter and on, if p is used) and secondary input buffer are reset
RI(m)
primary input buffer (from m'th column and on) and secondary input buffer are reset
RO
both output buffers are reset

Figure A. General Form of IH, H, RI, and RO Commands

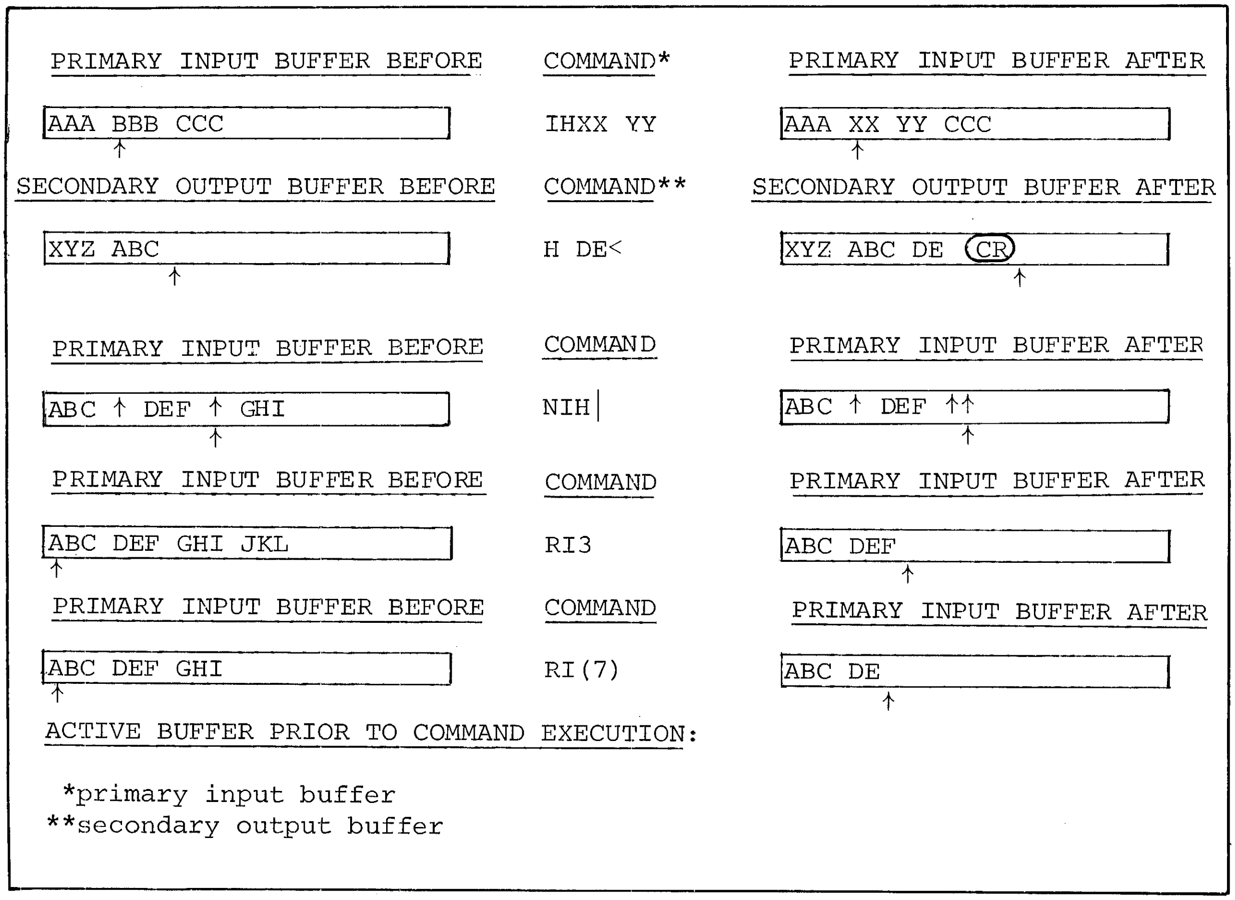

PRIMARY INPUT BUFFER BEFORE
COMMAND*
PRIMARY INPUT BUFFER AFTER
AAA BBB CCC
IHXX YY
AAA XX YY CCC
SECONDARY OUTPUT BUFFER BEFORE
COMMAND**
SECONDARY OUTPUT BUFFER AFTER
XYZ ABC
H DE<
XYZ ABC DE CR
PRIMARY INPUT BUFFER BEFORE
COMMAND
PRIMARY INPUT BUFFER AFTER
ABC ↑ DEF ↑ GHI
NIH|
ABC ↑ DEF ↑↑
PRIMARY INPUT BUFFER BEFORE
COMMAND
PRIMARY INPUT BUFFER AFTER
ABC DEF GHI JKL
RI3
ABC DEF
PRIMARY INPUT BUFFER BEFORE
COMMAND
PRIMARY INPUT BUFFER AFTER
ABC DEF GHI
RI(7)
ABC DE
ACTIVE BUFFER PRIOR TO COMMAND EXECUTION:
*primary input buffer
**secondary output buffer

Figure B. Sample Usage of IH, H, and RI Commands

2.14 TRANSFERRING CONTROL: THE GO,GO F,GO B,M,GOSUB AND RSUB COMMANDS

The GO commands provide unconditional branching within PROC's.

The GO command causes control to unconditionally transfer to the PROC command which has the numeric label n. For example:

```
G 10  or GO 1Ø
```

This command causes control to transfer to the PROC command which begins with the label 10. The user should note that several PROC commands may begin with the same label. If this is the case, the GO command transfers control to the *first* PROC command which begins with the specified label (scanning from the top). This is generally the slowest form of the branching instructions.

The GO F and GO B commands go forward and backward (respectively) to the appropriate M (mark) command. The GO B command will branch to the instruction following the last executed M (mark) command. The GO F command scans forward until it can execute an M command and then continues processing of the command following the M command.

The M (mark) command simply causes the PROC processor to remember or "mark" its location. Only the last executed M command is remembered.

Note that GO F, GO B, and M are not necessary if the PROC is to be compiled (see PQ-COMPILE verb).

A GOSUB command causes execution control to be transferred to the specified label. See Figure A for a summary of the instructions. An RSUB n command returns execution control n lines after the GOSUB. If n is missing, then n=1 is assumed. An RSUB command will be ignored if no previous GOSUB was executed.

GO n	*Go to label n.*
GOSUB n	*Go to subroutine at label n.*
RSUB n	*Return to subroutine call. Begin executing n lines after the call. If n is missing, then n=1.*
GO B	*Go back to previous mark command.*
GO F	*Go forward to next mark command.*
M	*Mark command.*

Figure A. General Command Formats

This PROC will print the numbers 1 through 5 on the terminal.

001	PQ	*PROC identification.*
002	GOSUB 200	*Branch to subroutine 200.*
003	M	*"Mark" location.*
004	IF # A GO F	*Test for null parameter; if missing, go forward to next "Mark".*
005	D+	*Output current parameter.*
006	F	*Move buffer pointer forward.*
007	GO B	*Go back to previous "Mark".*
008	M	*"Mark".*
009	X FINI!!	*Exit PROC and print FINI!!*
010	200 IH 1 2 3 4 5	*Subroutine 200, insert numbers into buffer.*
011	RSUB	*Return to subroutine call (Line 200).*

Figure B. Example of Transferring Control

2.15 TRANSFERRING CONTROL: THE SIMPLE IF COMMAND

The IF command transfers control to another statement within the PROC.

The IF command provides for the conditional execution of a specified PROC command. The IF command takes on three general forms. The simple form is as follows:

```
IF {#}a-cmnd proc-cmnd
```

where a-cmnd is any legal form of the A command (refer to the topic titled MOVING PARAMETERS: THE A AND NA COMMAND) except for the form using the character surround feature (i.e., Ac), and where proc-cmnd is any legal PROC command. If the optional # is not used, the IF command simply tests for the existence of a parameter in the input buffer as specified by the A command. If a parameter exists, the specified PROC command is executed; otherwise, control passes to the next sequential PROC command. For example:

```
IF NA2 GO 15
```

This command tests for the existence of a second parameter in the currently active input buffer. If a parameter exists, control passes to the PROC command beginning with label 15; otherwise, control passes to the next sequential PROC command. If the # option is used, the test is reversed. For example:

```
IF # A2 GO 15
```

This command causes control to transfer to the command with label 15 if a second parameter does *not* exist.

The user should note that when using an A command as a test condition of an If command, parameters are not moved to an output buffer as would be the case if the A command were used along. Rather, the A command is used simply to specify which parameter in the input buffer is to be tested. However, the input pointer *will* be re-positioned as specified by the A command.

A number of examples illustrating the simple form of the IF command are presented in Figure B. For a discussion of the two other forms of the IF command, refer to the topic titled RELATIONAL TESTING: THE RELATIONAL IF COMMAND and topic titled PATTERN TESTING: THE PATTERN MATCHING IF COMMAND.

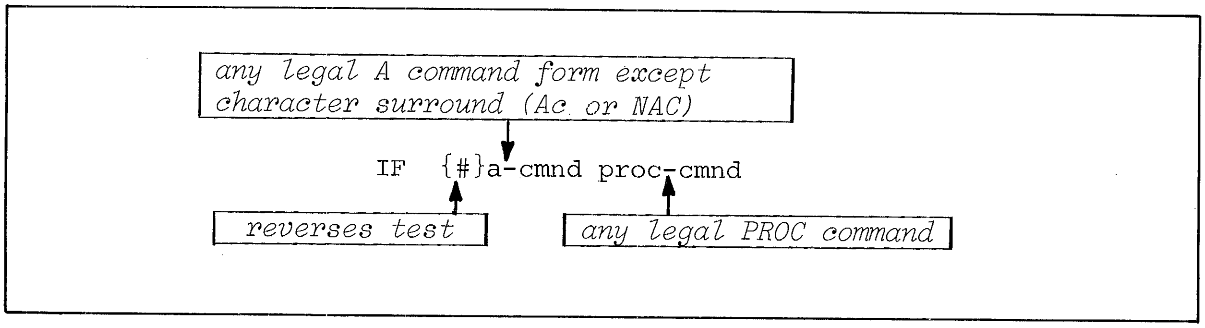

any legal A command form except character surround (Ac or NAC)
IF {#}a-cmnd proc-cmnd
reverses test
any legal PROC command

Figure A. Simple Form of IF Command

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
ABC AAA XYZ
↑
```

COMMAND	EXPLANATION
IF A GO 27	*Control is transferred to the command with label 27.*
IF A3 OHELLO	*Message HELLO is output to terminal; control then continues with next sequential command.*
IF A4 OHELLO	*Message is not output; control continues with next sequential command.*
IF A(11) GO 2	*Control is transferred to the command with label 2.*

Figure B. Sample Usage of Simple IF Command

2.16 RELATIONAL TESTING: THE RELATIONAL IF COMMAND

The relational form of the IF command allows parameters in the input buffers to be tested relationally.

The relational form of the IF command is an extended version of the simple IF form (see topic titled TRANSFERRING CONTROL: THE GO AND SIMPLE IF COMMANDS). The relational form is as follows:

```
If a-cmnd op string proc-cmnd
```

where a-cmnd and proc-cmnd are as defined for the simple IF form, where op is one of the relational operators listed in Figure B, and where string is a literal string of characters which the parameter is to be compared against. For example:

```
IF A = YES GO 5
```

Here the PROC would transfer control to the command with label 5 if the current parameter in the currently active input buffer is the character string YES.

To resolve a 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 are found, the characters are ranked according to their numeric ASCII code equivalents (refer to the LIST OF ASCII CODES in the Appendix B to this manual). The character string contributing the higher numeric ASCII code equivalent is considered to be greater than the other string. For example, AAB is considered 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.

Further examples illustrating the relational IF command are presented in Figure C.

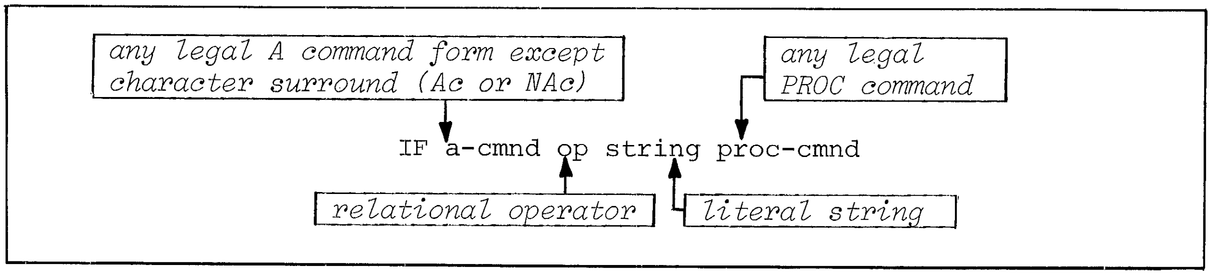

any legal A command form except character surround (Ac or NAc)
any legal PROC command
IF a-cmnd op string proc-cmnd
relational operator
literal string

Figure A. Relational Form of IF Command

OPERATOR SYMBOL	OPERATION
=	*test for equal*
#	*test for not equal*
<	*test if parameter less than literal string*
>	*test if parameter greater than literal string*
[	*test if parameter less than or equal to literal string*
]	*test if parameter is greater than or equal to literal string*

Figure B. Relational Operators

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
 ABC AAA XYZ
 ↑
```

COMMAND	EXPLANATION
IF A = ABC GO 3	*Control is transferred to the command with label 3*
IF A3 > XYX HTEST	*The text string TEST is placed in the currently active output buffer; control then continues with next sequential command.*
IF A2 > XYX HTEST	*Text string TEST in not placed in output buffer; control continues with next sequential command.*
IF A(2,2) = BC GO 7	*Control is transferred to the command with label 7.*

Figure C. Sample Usage of Relational IF Command

2.17 PATTERN TESTING: THE PATTERN MATCHING IF COMMAND

The pattern matching form of the IF command allows parameters in the input buffers to be tested for a specific pattern match.

The pattern matching form of the IF command is an extended version of the simple IF form (see topic titled TRANSFERRING CONTROL: THE GO AND SIMPLE IF COMMANDS). The pattern matching form is as follows:

```
IF a-cmnd op (pattern) proc-cmnd
```

where a-cmnd and proc-cmnd are as defined for the simple IF form, where op is one of the relational operators described for the relational IF form, and where pattern is a pre-defined format string enclosed in parentheses.

A pattern is used to test a parameter for a specified combination of numeric characters, alpha characters, alpha-numeric characters, or literals. The pattern specification in an IF statement consists of any combination of the following:

- An integer number followed by the letter N (which tests for that number of numeric characters).
- An integer number followed by the letter A (which tests for that number of alpha characters).
- An integer number followed by the letter X (which tests for that number of alpha-numeric characters).
- A literal string (which tests for that literal string of characters).

As an example, consider the following command:

```
If A = (3NABC)  G 3
```

This command causes a transfer of control to the command with label 3 when the current parameter of the currently active input buffer consists of three numerals followed by the characters ABC (e.g., 123ABC).

If the integer number used in the pattern is 0, the test is true only if all the characters in the parameter conform to character type. The following command, for example, outputs the message OK if the characters of the current parameter are all alpha characters:

```
IF A = (0A) OOK
```

Further examples of the pattern matching form of the IF command are illustrated in Figure B.

The user should note that for any of the three IF command forms, the PROC statement which is conditionally executed may in turn be another IF command (i.e., IF commands may be nested). The following command, for example, 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 ] 10 IF A [ 19 GO 99
```

The user may wish to visualize nested IF commands as though implied AND operators were placed between them.

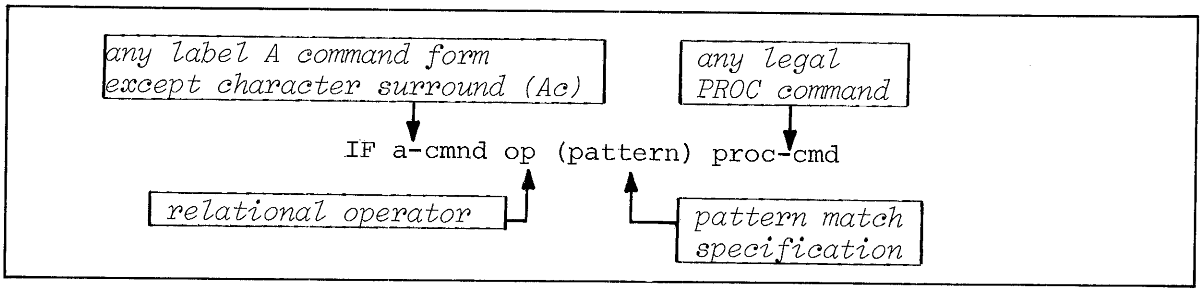

any label A command form except character surround (Ac)
any legal PROC command
IF a-cmnd op (pattern) proc-cmd
relational operator
pattern match specification

Figure A. Pattern Matching Form of IF Command

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters.

```
ABC 10/09/75 XYZ B123C 33
↑
```

COMMAND	EXPLANATION
IF A = (3A) G 7	*Control is transferred to the command with label 7.*
IF A2 = (2N/2N/2N) G 5	*Control is transferred to the command with label 5.*
IF A4 = (0N) G 9	*Control continues with next sequential command.*
IF A5 = (0N) GO 2	*Control is transferred to the command with label 2.*
IF A4 = (1A3NC) OGOOD	*The message GOOD is output to the terminal; control continues with the next sequential command.*
IF A1 = (3X) IF A1 > ABB G 9	*Control is transferred to the command with label 9.*
IF A(5,2) = (2N) G3	*Control is transferred to the command with label 3.*

Figure B. Sample Usage of Pattern Matching IF Command

2.18 COMPUTED BRANCHING: THE MULTI-VALUED IF COMMAND

The IF command can contain a multi-valued object and/or a multi-valued destination.

When using the IF command with the equal(=) or not equal(#) operator, the object and/or the destination may be multi-valued.

When using the equal operator, the multi-values are "OR'ed" together. This means that each multi-value will be tested against the argument; if *any* value is true, the branch is taken.

When using the not equal operator, the multi-values are "AND'ed" together. *All* values must match the argument for the branch to be taken.

If the object is multi-valued, then the destination *may* be a multi-valued GO or GOSUB command. This means that the destination labels may be multi-valued (separated by VM's). This is equivalent to an N-way branch. If there are more object values than destination values, the last destination will be executed for object values greater than the number of destinations.

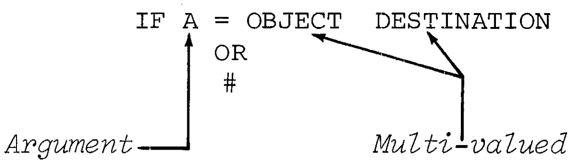


Figure A. General Command Format

If A2 = 1]2]3 GO 200	*If A2 equals a 1 or 2 or 3, then go to label 200.*
If A4 # RED]WHITE]BLUE GO 500	*If A4 is not equal to RED and not equal to WHITE and not equal to BLUE, then go to label 500.*
If A3 = 1]2]3 GOSUB 10]20]30	*If A3 equals 1, then GOSUB label 10. If A3 equals 2, then GOSUB 20. If A3 equals 30, then GOSUB 30.*
If A5 = 11]12]13 GO 10]20	*If A5 equals 11, then GO TO 10. If A5 equals 12 or 13, then GO TO 20.*

Figure B. Example Usage of the Multi-valued IF Command

2.19 FILE COMMANDS: THE F-OPEN AND F-CLEAR COMMANDS

Files may be manipulated with the use of the PROC file commands. Up to nine files may be opened with the F-OPEN command. F-CLEAR clears the file buffer.

File manipulation is performed by the various file commands available from PROC. Before the files can be worked on, they must be opened.

The F-OPEN command (see Figure A) opens file Y and assigns it to file buffer number X. X may be any digit 1 through 9. Y may be a literal file name or may be an indirect reference to a file name. The dictionary of a file may be opened by preceding the file parameter (Y) with the word DICT.

A file need only be opened once. Files remain open throughout the PROC execution, including subroutine calls and PROC-to-PROC linkage. All files are closed upon termination of the PROC. File buffer numbers can be re-used by executing another F-OPEN command. The F-OPEN effectively closes the file previously opened to that buffer number and assigns the buffer to the newly opened file. See examples in Figure B.

The F-CLEAR command clears the file buffer number Y (see Figure A). An F-CLEAR is required if new records are being built, or if the file buffer is being used as a scratch area. It must be used before the first OPEN, READ, WRITE, or DELETE.

NOTE: *The file commands use an attribute mark as the parameter delimiter.*

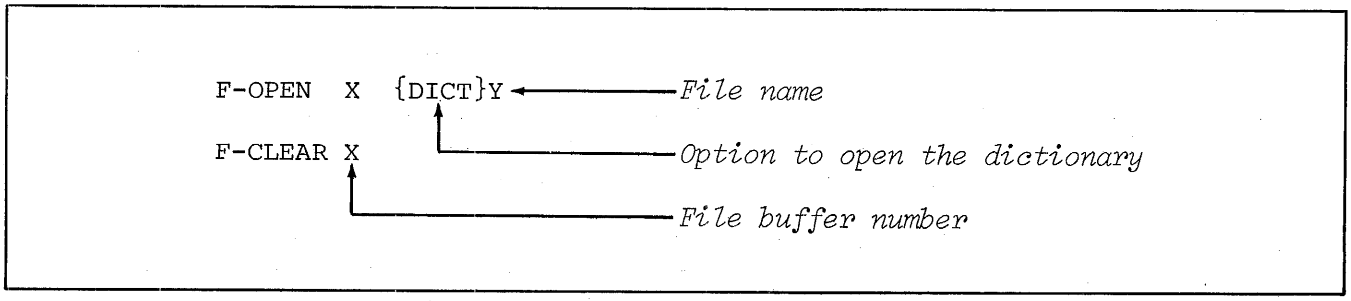


Figure A. General Command Format

F-OPEN 5 INVENTORY	*Opens the INVENTORY file to file buffer 5*
F-CLEAR 3	*Clear file buffer number 3*

Figure B. Examples of Opening Files and Clearing

2.20 FILE COMMANDS: THE F-READ, F-WRITE, AND F-DELETE COMMANDS

The F--READ command loads an item into a file buffer. F-WRITE writes the contents of a file buffer. F-DELETE deletes from the file the item specified in the file buffer.

The F-READ command reads an item into a file buffer. The item is read from the file which was previously opened to the file buffer number specified. The item-ID may be a literal or an indirect reference to an item-ID.

If the item is found by the F-READ command, it is read into the file buffer and the PROC continues execution on the second line following the F-READ command (normal return). If the item is not found, the PROC executes the statement immediately following the F-READ command (error return). See Figure B. F-READ does not lock the group of the file containing the item.

The F-WRITE command writes the data contained in the file buffer number specified to the file currently opened to that file buffer. The item-ID for the item is obtained from the file buffer, attribute zero. No error condition exists for a write.

The F-DELETE deletes the item whose item-ID is found in attribute zero in the file buffer number specified. The item is deleted from the file currently opened to the file buffer. No error condition exists for a DELETE.

NOTE: *These commands utilize an attribute mark as the separator parameter.*

```
F-READ   X  Y

F-WRITE  X
                      Item-ID (literal or indirect reference)
F-DELETE X
                      File buffer number
```

Figure A. General Command Format

```
F-READ  1  1234     Read item from the file opened to file
GO 140               buffer 1 into file buffer 1 using item-ID
GO 200               '1234'.  If found, execute second command
                     (GO 200); if not found, execute first
                     command (GO 140).

F-READ  3  %4       Read from file buffer 3 using the value in
                     the 4th parameter in the primary input
                     buffer.
X READ ERROR             Error return after read
GO 500                   Normal return after read

F-WRITE  4          Write the contents of file buffer 4 to the
                     file currently opened to file buffer 4.

F-DELETE  7         Delete the item, whose item-ID is contained
                     as attribute zero in file buffer 7, from
                     the file currently opened to file buffer
                     number 7.
```

Figure B. Examples of File Commands

2.21 FILE COMMANDS: THE FB COMMAND

The FB command combines an OPEN and a READ to load data into a "fast buffer".

The FB command reads an item into an internal buffer from the file specified. There is only one "fast buffer", so subsequent uses of the FB command destroy previous data in the fast buffer. The FB command is preferable to an F-OPEN and F-READ sequence if only one item from a file is to be processed. The fast buffer is destroyed by a P command.

The FB command, like the F-READ, has two return points. The normal return (successful read) is the second command following the FB command. The error return (unsuccessful read) is the command immediately following the FB command.

Once the item is read into the fast buffer, it can be referenced using the &X form of indirect reference. Note that the command is ampersand (&) X (attribute number).

If the item-ID is omitted from the FB command, then the value in the current location of the primary input buffer is used as the item-ID.

NOTE: *The FB command uses an attribute mark as the parameter delimiter.*

```
FB  ({DICT}  file-name{item-ID})

ERROR RETURN

NORMAL RETURN
```

Figure A. General Form of the FB Command

FB (INVENTORY AB-4158) ERROR RETURN NORMAL RETURN	*Read into the fast buffer the item whose ID is AB-4158 from the inventory file.*
FB (DICT INVENTORY DL/ID) ERROR RETURN NORMAL RETURN	*Read the DL/ID from the dictionary of the inventory file into the fast buffer.*
FB (INVENTORY) ERROR RETURN NORMAL RETURN	*Read from the inventory file the item whose ID is contained at the current pointer position in the primary input buffer.*

Figure B. Using the FB Command

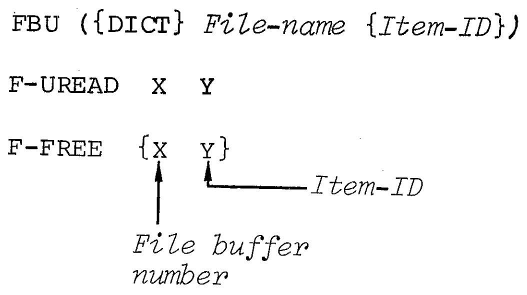
2.22 FILE-LOCKS: THE F-UREAD, FBU, AND F-FREE COMMANDS

The Read for Update commands lock out file items against other users. They use the Reality group lock facility.

The F-UREAD and FBU commands function identically to the F-READ and FB commands except that they lock the group of the referenced file to which the item-ID hashes. This serves to protect against simultaneous item retrieval by one line and item update by another. It also gives a logical lock capability.

The group will be locked by the F-UREAD and FBU even if the item is not found.

The F-FREE command will free all groups locked by the process. The form F-FREE X Y will unlock the group in the file X to which the item Y hashes. It will be unlocked only if the process has it locked.



```
FBU ({DICT} File-name {Item-ID})

F-UREAD  X  Y

F-FREE  {X  Y}
```

Figure A. General Form of the File-Lock Commands

F-UREAD 1 1234	*Locks the group in the file opened to file buffer 1. Locks the group to which '1234' hashes, even if '1234' is not on file, then proceeds as 'F-READ 1 1234'.*
F-FREE 1 1234	*Unlocks the group locked in the above example, but only if locked by this line.*
F-FREE	*Unlocks all group locks locked by this line.*

Figure B. Examples

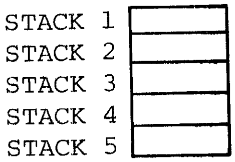
2.23 MATHEMATICAL FUNCTIONS: THE F; COMMAND

> The F; command provides the PROC processor with arithmetic functions.

The F; command performs all functions in fixed point, integer arithmetic. This command operates like the function correlative in dictionaires. Elements in an F; command are separated by semi-colons (;). An element may be an operand or an operator. The command is processed from left to right. Each element encountered is placed on the top of a push-down/pop-up stack. Each operator performs its function on the top of two entries in the stack, deletes them and places the result on the top of the stack.

The result of the function can be moved into any of the PROC buffers.

The stack concept can be visualized as follows:



```
F;element1; element2;...;element n
```

Figure A. General Format of the F; Command

OPERANDS

#n,%n,&n,&F.n	*Indirect references to numeric values to be placed on the top of the stack.*
Cn	*Constant "n" to be placed on the stack.*

OPERATORS

+	*Add top two entries in the stack.*
-	*Subtract STACK2 from STACK1.*
*	*Multiply top two entries in the stack.*
/	*Divide STACK1 by STACK2.*
R	*Divide STACK1 by STACK2, but return remainder to STACK1.*
←	*Exchange STACK1 and STACK2.*
?P	*Store STACK1 at current primary input buffer pointer location.*
?D	*Store STACK1 at buffer location specified by D. D is an indirect buffer reference.*

Figure B. F; Command Elements

```
F;C20;%4;*;?P          F; command

SMITH^400^^8^          Primary input buffer (pointer at parameter 3)

         E1  E2  E3   E4
STACK1   20   8  160  160
STACK2   -   20   -    -

SMITH^400^160^8^  Primary input buffer after execution of the F; command
```

Figure C. Sample of F; Command Usage

2.24 FORMATTED LINE PRINTER OUTPUT: THE L COMMAND

The L command causes the PROC processor to enter a line printer formatting routine.

The L command is followed by one blank and any number of elements separated with commas. Each L command formats a line of output. If more elements are required than will fit on one PROC line, then the L command line should be terminated with a comma. Figure A shows the general format.

Page headings are generated through the use of the HDR element. Any element may be used in a line containing a HDR; however, the elements shown in Figure B following "HDR" may be used *only* following the HDR element. The E and n elements cannot be used in a HDR line.

When a "HDR" line is encountered, any direct or indirect references are resolved and stored in a heading buffer. Therefore, it is not necessary to maintain the indirect reference values. After the first execution of a HDR line, the PROC will maintain a line counter and print a heading each time the counter exceeds the page depth set by the last TERM command.

```
L element1, element2,...,...,element n,
                                       ↑
           causes continuation─────────┘
```

Figure A. General Form of L Command

"text"	*Output any text between the quotes.*
(n)	*Set to column position n.*
%,#,&	*All forms of indirect referencing.*
E	*Eject to top of form.*
n	*Skip n lines before printing.*
C	*Close the print file.*
HDR	*Generates the page heading line.*
P	*Print current page number.*
T	*Print current time and date.*
Z	*Zero the page counter.*

Figure B. L Command Elements

```
001  PQ
002  NIP
003  L HDR,T,(40),"PAGE",P
004  L 1,(10),%1,(15),%2,(20),%3,(25),"PARAMETERS 1 THRU 3"
005  L 1,(10),%4,(15),%5,(20),%6,(25),"PARAMETERS 4 THRU 6"

IF INPUT:  100 200 300 400 500 600   CR
```

will print:

```
10:09:25    1 SEPT 1979          PAGE 1

            100  200  300  PARAMETERS 1 THRU 3

            400  500  600  PARAMETERS 4 THRU 6
```

Figure C. Sample PROC and Output

2.25 TERMINAL FORMATTING: THE T COMMAND

The T command causes the PROC command to enter a terminal formatting routine.

The T command is followed by one blank and any number of elements which are separated by commas (Figure A). If a T command line ends with a comma, then the command is continued on the next line.

If the last element of a T command line is "+", the automatic carriage return-line feed is suppressed.

The %, #, and & elements are used to display the PROC buffers. These elements may be followed by any Reality conversion code enclosed in (:) colons. An example would be %2:D2,:.

NOTE: *The T command assumes an attribute mark as the parameter delimiter.*

```
T (element1,element2,...,element n),
```

causes command continuation

Figure A. The T Command Format

"text"	*Output data between quotes*
(X,Y)	*Set cursor position: X-column, Y-row*
B	*Ring bell*
C	*Clear Screen*
In	*Output character with ASCII integer value n*
Xn	*Output character with ASCII hex value n*
Sn	*Output n spaces*
U	*Move cursor up one line*
%n	*Output nth parameter of primary input buffer*
#n	*Output nth parameter of current output buffer*
&n	*Output nth parameter of internal fast buffer*
T	*Tag*
D	*Delay*
L	*Loop*

Figure B. T Command Elements

```
PQ
T C
M
T (10,10,),"DATE AND TIME",S5,+
HTIME
T (10,10),S30,+
GO B
```

Prints the date and time continuously at the 10th column and 10th row on the screen.

Figure C. Sample PROC with T Command Usage

2.26 THE PLUS (+), MINUS (-), C, AND TR COMMANDS

The Plus and Minus commands are used to add or subtract (respectively) a specified decimal number to/from the current parameter of the currently active input buffer. The C command is used to place comments within the body of the PROC. The TR (TRace) command is used to print PROC commands as they are executed.

The Plus (+) command has the following general form:

```
+n
```

This command causes the decimal number n to be added to the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

The Minus (-) command has the following general form:

```
-n
```

This command causes the decimal number n to be subtracted from the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

The Plus or Minus commands will have no effect if the input pointer is currently at the end of the buffer. Also, the user must take care that the updated value of the parameter is the same length as the original value of the parameter, since no automatic check for this is made. Sample usage of the Plus and Minus commands are shown in Figure B.

The C command is used to place comments within the body of the PROC. The general form of this command is as follows:

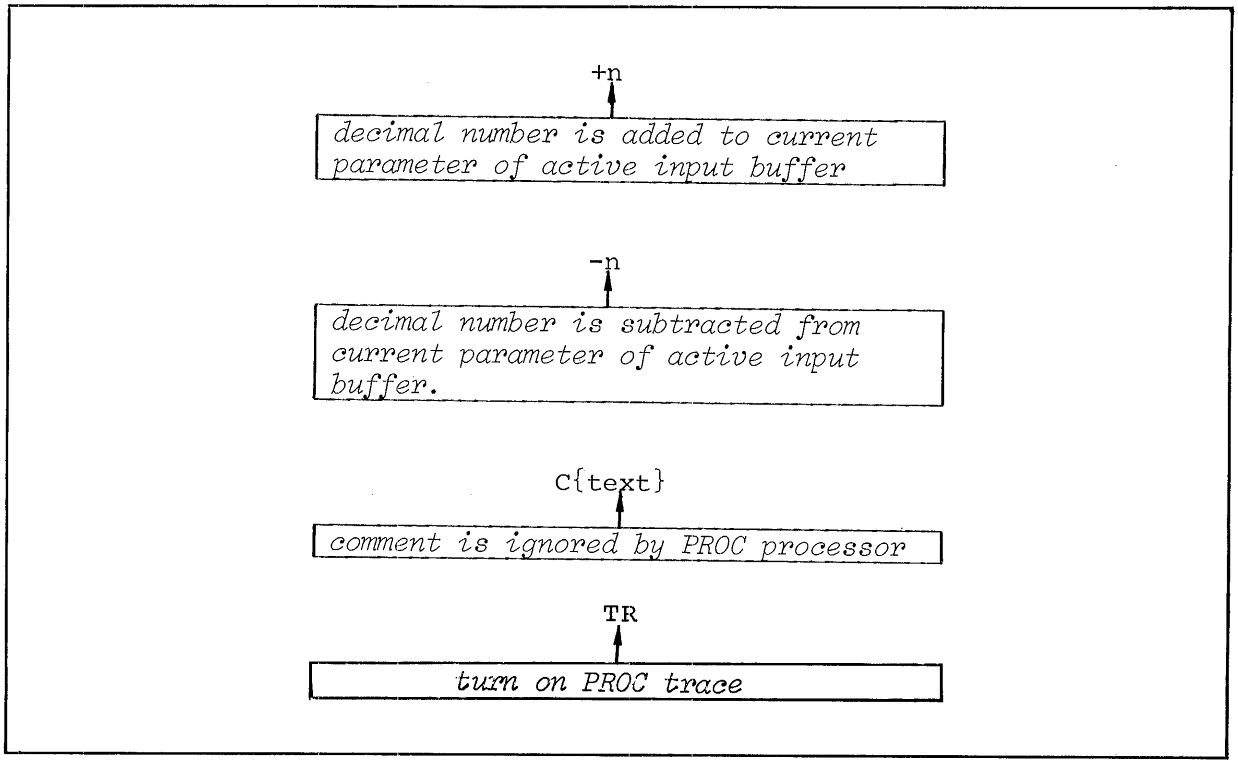
```
C{text}
```

All text following the C will be ignored by the PROC processor. For example:

```
C THIS IS A COMMENT
```

The C command may be used freely throughout the PROC for purposes of clarity and documentation.

The TR (Trace) command is a debugging tool designed to assist the programmer with a debugging problem. Once executed, all PROC commands print on the terminal prior to executing. The "IF" commands will print a second time if the evaluation was true. The Trace command remains in effect until return to TCL.



+n

decimal number is added to current parameter of active input buffer

-n

decimal number is subtracted from current parameter of active input buffer.

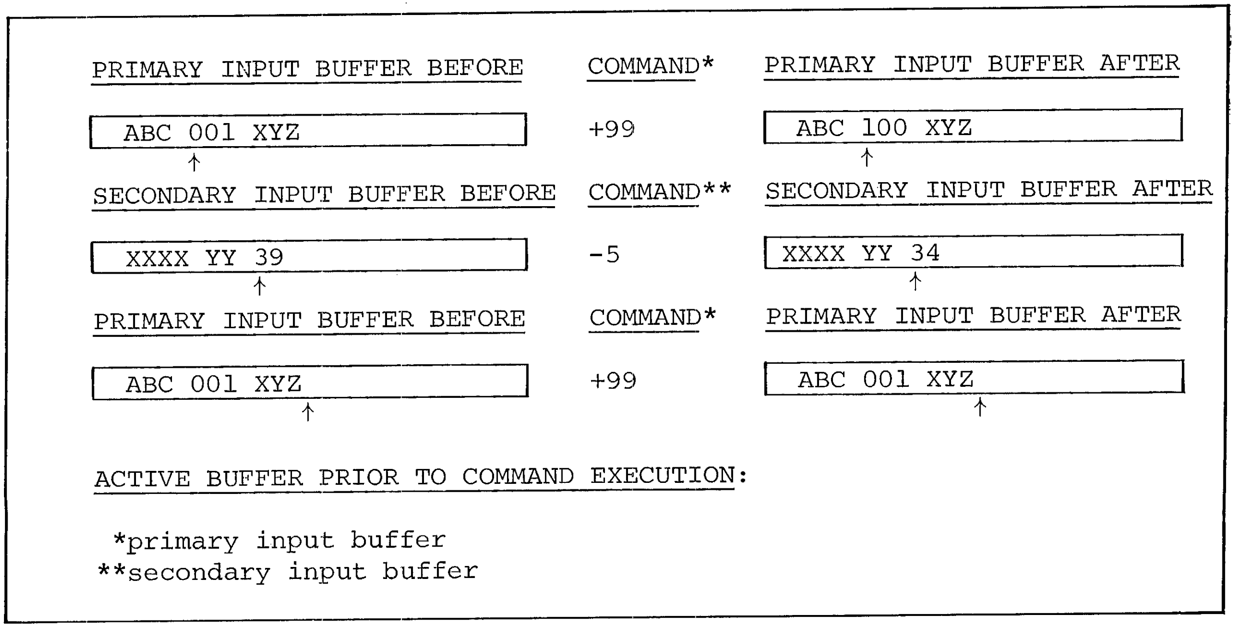
C{text}

comment is ignored by PROC processor

TR

turn on PROC trace

Figure A. General Form of Plus (+), Minus (-), and C Commands



PRIMARY INPUT BUFFER BEFORE	COMMAND*	PRIMARY INPUT BUFFER AFTER
ABC 001 XYZ ↑	+99	ABC 100 XYZ ↑
SECONDARY INPUT BUFFER BEFORE	**COMMAND****	**SECONDARY INPUT BUFFER AFTER**
XXXX YY 39 ↑	-5	XXXX YY 34 ↑
PRIMARY INPUT BUFFER BEFORE	**COMMAND***	**PRIMARY INPUT BUFFER AFTER**
ABC 001 XYZ ↑	+99	ABC 001 XYZ ↑

ACTIVE BUFFER PRIOR TO COMMAND EXECUTION:

*primary input buffer
**secondary input buffer

Figure B. Sample Usage of Plus (+) and Minus (-) Commands

2.27 CALLING PROCESSORS, PROGRAM LOCKS

The PROC processor has a Process command which may take several forms.

The P (Process) command is used by the PROC processor to execute the command which has been built in the primary output buffer. Functionally, temporary control is passed to TCL and upon completion returned to the PROC.

The Process command with a single P (see Figure A) simply passes control to TCL. Variations to the P command allow additional processing to occur. The PH command "hushes" or suppresses system messages during execution. The PL command locks the PROC execution. This means that if more than one user is executing this PROC, the first user to reach this command "roadblocks" the other users until execution is complete. The PHL combines the PH and PL forms. These locks are shared with the BASIC and BATCH locks of the same number (0 to 63).

The double P (PP) version of the Process command also causes transfer to TCL; however, prior to processing, the contents of both output buffers are displayed and the user is prompted with a (?) question mark. If execution is *not* desired, a response of N is required and the PROC is ended. If execution is not desired but the PROC should be continued, respond with an S. This will cause the output buffers to be cleared. The PP command is usually used as a debugging tool.

WARNING: *A PROC may fail if it is unsituated from its location in its file while it is being executed. For this and other reasons, it is good practice to put PROC's into dictionary level files dedicated to PROC's.*

```
{P} P          Process the output buffer
{P} PH         Process and suppress messages
{P} PLn        Process and lock execution
{P} PHLn       Process, suppress, and lock
 ↑
 |   n = Resource lock number (0 to 63)
 └─ Option to display output buffers
```

Figure A. General Command Format

```
     END-OF-MONTH
001  PQ
002  H LIST INV LPTR          Load the output buffer.
003  P                        Process the output buffer.
004  H SELECT MASTER WITH     Load the primary output buffer.
005  H QTY > "500"
006  STON                     Select the "stack".
007  H RUN BP UPDATE          Load the stack.
008  PP                       Print output buffers prior to processing.
```

Figure B. Sample Usage of the Process Command

```
:END-OF-MONTH  CR                   NOTE: No indication of the first
                                    process command.

SELECT MASTER WITH QTY > "500"      Primary output buffer.

RUN BP UPDATE←                      Secondary output buffer (stack).

?  CR                               Carriage return to continue.

1200 ITEMS SELECTED                 Result of SELECT; would not have
                                    printed with the PPH form.

:                                   Back to TCL.
```

Figure C. Execution of the PROC in Figure B

2.28 PROC TERMINATION AND USER EXITS: THE X AND U COMMAND

The X command causes an immediate return to TCL. The U command allows an exit from PROC to an assembly coded subroutine.

The X command is used to leave a PROC and return to TCL. When the X command is executed, the return to TCL is immediate and no processing of the output buffers takes place. Any text following the X command will be output to the users terminal.

The U command is used to exit from the PROC to a user written assembly coded subroutine. The interface to PROC's is described in the appendix of this manual. Strict adherence to this interface will ensure compatability to future software releases.

```
X text                    Exit from PROC, no processing; output optional
                           message to terminal.
```

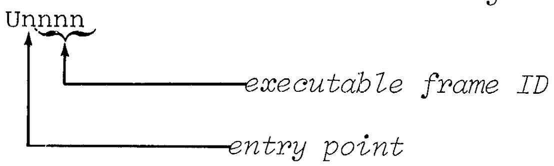


Figure A. General Command Format

```
001  PQ

002  IH 000

002  10 + 1

003  IF A = 100 X FINI!!          Exit from PROC and print FINI!!

004  UD1A6                        Execute user subroutine.  Entry point
                                  '0' of frame 1A6₁₆

005  GO 10
```

Figure B. Sample Usage of X and U Commands

2.29 TRANSFERS AND SUBROUTINE CALLS: THE ( ), [ ], AND RTN COMMANDS

The Link commands, denoted as ( ) or [ ], allow other PROC's to be evoked.

A Link command in one PROC causes control to transfer to the first command of another PROC, which may reside in any dictionary or data file. This allows the storage of PROC's (except for the LOGON PROC) outside of the M/DICT. Also, large PROC's can be broken into smaller PROC's to minimize processing time.

If the item-ID (Figure A) is omitted, the current parameter (as specified by the input pointer) of the currently active input buffer is retrieved and used as the item-ID. The optional DICT specifies the dictionary portion of the file.

The user should note that the PROC buffers remain unchanged when a linkage occurs. Also, the first line of the linked-to item is always skipped, since it is assumed that this line contains the PQ code.

The parentheses ( ) form of the Link command causes a one-way transfer.

The bracket [ ] form of the Link command allows the external subroutine capability for PROC's. Using this form, transfer is made to the indicated PROC, but a return to the "calling" PROC is made if a RTN command is executed. One PROC may call another PROC, which may call another PROC, etc., indefinitely.

The RTN command is ignored if no previous Link command was executed. An optional value may be placed after the RTN command, which indicates that the return should be made n lines after the Link command.

```
({DICT} file-name {item-name}

[{DICT} file-name {item-name}]

RTN{X}
```

Figure A. General Link Command Formats and Return

```
ITEM-LISTFILES

001  PQ

002  (DICT PROCLIB)              Item-name is missing; transfer to item-
                                  name whose value is in the current
                                  position of the primary input buffer.
```

Figure B. Permanent Transfer

```
ITEM - FILE-SAVE

001  PQ

002  [DICT MD VERIFY-SYSTEM]             The three PROC's must have a
                                         RTN command.
003  [DICT SYSPROG-PL]

004  [DICT SYSPROG-PL FILE-STATS]

005  HOFF

006  O FILE SAVE COMPLETE

007  P
```

Figure C. PROC's as Subroutines

2.30 SPECIAL PROC VERBS: PQ-COMPILE AND PQ-SELECT

The PQ-COMPILE verb compiles PROC source programs into smaller and faster executing "object" programs. The PQ-SELECT verb loads the designated select register with item-ID's.

The PQ-COMPILE verb (Figure A) compiles the PROC into "object" form. BRANCHES and GOSUB commands, in particular, will be speeded up. The PROC compiler will *not* allow the original PROC to be overlayed. Prior to compilation, the user will be prompted for the destination file. The compiled code is readable, but many commands take an altered form. Also, all comments are deleted from the "object" or compiled version of the PROC.

WARNING: *Changing a compiled PROC with other than another PQ-COMPILE may cause the PROC to abort or fail when run.*

See Figure B for a sample compilation.

The PQ-SELECT verb is used to place item-ID's into a select register (see Figure A). Note that PQ-SELECT is a system verb and must be placed in an output buffer and passed to the TCL processors for execution. The PQ-SELECT verb is designed to function following a SELECT, GET-LIST, SSELECT or FORM-LIST verb. The items selected are then placed in the designated select register. The item-ID's in the select register may be referenced using the (!n) form of the indirect buffer reference described elsewhere in this manual (see Figure D for an example).

```
PQ-COMPILE file-name item-name
               ↑          ↑
               │          └────proc name
               │
               └─source file name

PQ-SELECT n

                 A digit (1 through 5)
                 representing the desired
                 select register.
```

Figure A. General Verb Formats

```
:PQ-COMPILE  PROC-SOURCE  LISTU  CR
DESTINATION:  PROC-LIBRARY  CR
```

Figure B. Sample PROC Compilation

9020	COMPILATION ABORTED
9021	FORMAT ERROR IN LABEL
9022	DUPLICATE LABEL
9025	FORMAT ERROR IN 'GO' STATEMENT

Figure C. PROC Compiler Messages

```
NHSELECT SYSTEM-OBJECT       Select items from file.
STON                         Turn on stack.
NH PQ-SELECT 1               Save item-ID's.
10 P                         Process output buffers.
NHMVERIFY SYSTEM-OBJECT      Move verb and file name to buffer.
NH !1                        Move item-ID to buffer.
IF # #3 GO 10                If not complete, branch.
X DONE!!                     Exit and print DONE!!
```

Figure D. Sample Use of PQ-SELECT

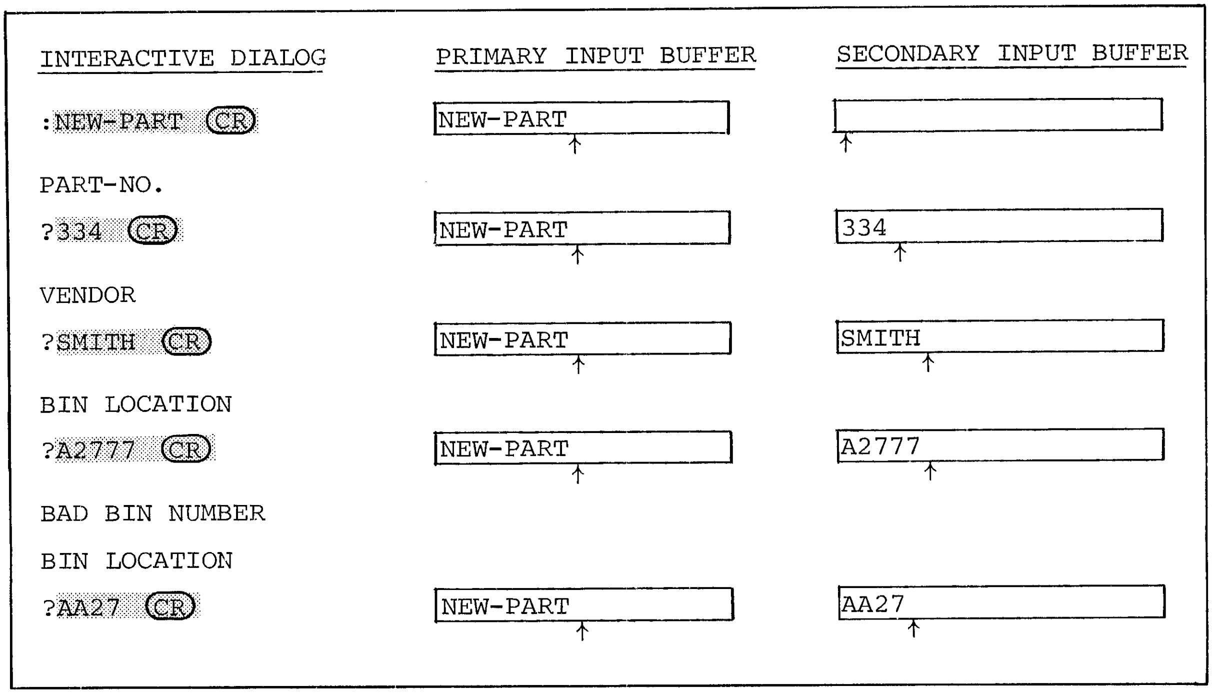
2.31 SAMPLE PROC: FILE UPDATE VIA BATCH

This topic presents a sample PROC which updates a file via the BATCH processor.

The left-hand column of Figure A shows the sample dialog used to evoke the PROC named NEW-PART. Note that the PROC prompts the user for the required data. Figure A also shows the respective content of both input buffers as the data are input.

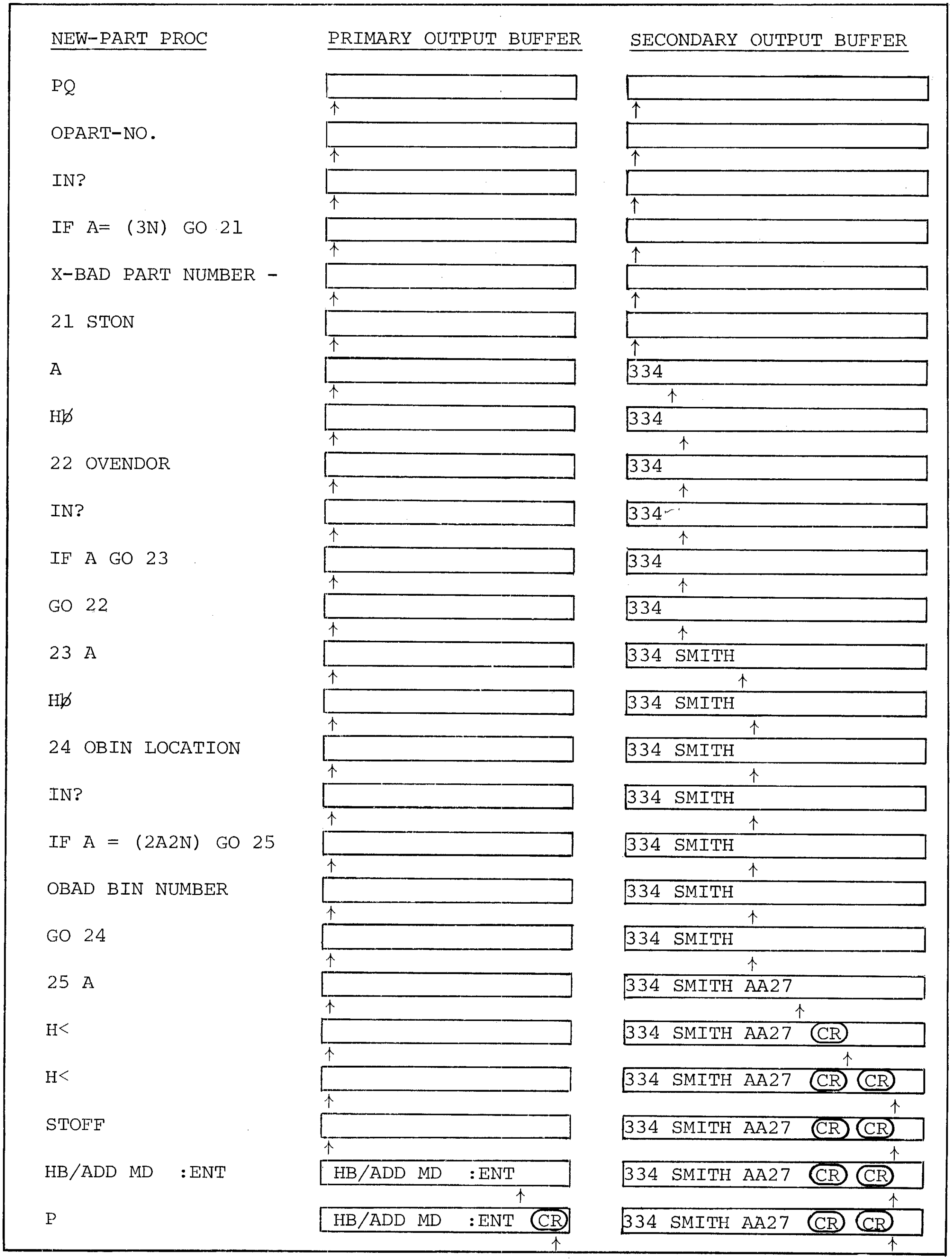
The left-hand column of Figure B shows the actual PROC listing. (The PROC is stored as item NEW-PART in the user's M/DICT.) Figure B also shows the content of both output buffers as they are affected by each PROC statement (and by the input data shown in Figure A).

Note that the TCL command to evoke the BATCH processor is being built in the primary output buffer, while the input lines used to feed the BATCH processor are being built in the secondary output buffer. The second carriage return in the secondary output buffer specifies a second input line for BATCH (i.e., a null input line). For further information regarding BATCH, refer to the section in this manual titled THE BATCH PROCESSOR.



INTERACTIVE DIALOG	PRIMARY INPUT BUFFER	SECONDARY INPUT BUFFER
:NEW-PART (CR)	NEW-PART	
PART-NO.		
?334 (CR)	NEW-PART	334
VENDOR		
?SMITH (CR)	NEW-PART	SMITH
BIN LOCATION		
?A2777 (CR)	NEW-PART	A2777
BAD BIN NUMBER		
BIN LOCATION		
?AA27 (CR)	NEW-PART	AA27

Figure A. Interactive Dialog and Input Buffers



NEW-PART PROC	PRIMARY OUTPUT BUFFER	SECONDARY OUTPUT BUFFER
PQ		
OPART-NO.		
IN?		
IF A= (3N) GO 21		
X-BAD PART NUMBER -		
21 STON		
A		334
HØ		334
22 OVENDOR		334
IN?		334
IF A GO 23		334
GO 22		334
23 A		334 SMITH
HØ		334 SMITH
24 OBIN LOCATION		334 SMITH
IN?		334 SMITH
IF A = (2A2N) GO 25		334 SMITH
OBAD BIN NUMBER		334 SMITH
GO 24		334 SMITH
25 A		334 SMITH AA27
H<		334 SMITH AA27 CR
H<		334 SMITH AA27 CR CR
STOFF		334 SMITH AA27 CR CR
HB/ADD MD :ENT	HB/ADD MD :ENT	334 SMITH AA27 CR CR
P	HB/ADD MD :ENT CR	334 SMITH AA27 CR CR

Figure B. PROC Listing and Output Buffers

2.32 SAMPLE PROC: FILE UPDATE VIA EDITOR

This topic presents a sample PROC which changes a specified attribute value via the EDITOR.

Figure A shows a sample EDITOR operation which changes attribute 3 of item 11115 of file ACCOUNT to the value ABC. Figure B shows a PROC named CHANGE which will perform the exact same operation. Note that the PROC has been written in such a manner that it updates any specified attribute in any specified item in any specified file. The format used to evoke this PROC is as follows:

```
CHANGE file item attribute-no  new-value
```

If, for example, the user wishes to perform the same operation shown in Figure A, the PROC must be evoked as follows:

```
:CHANGE ACCOUNT 11115 3 ABC (CR)
```

The user should note that the normal messages output by the EDITOR (e.g., TOP, '11115' FILED, etc.) are output when the PROC in Figure B is executed. These messages may be suppressed, however, by using the Z option when calling the EDITOR, and by preceding each EDITOR command by a period (.); for further information regarding these features, refer to the EDITOR Reference Manual.

```
:EDIT ACCOUNT 11115 (CR)
TOP
.G3 (CR)
003 100 AVOCADO
.R (CR)
003 ABC (CR)
.FI (CR)
'11115' FILED.
```

Figure A. Sample EDITOR Operation

```
item 'CHANGE' in M/DICT

       001 PQ
       002 HEDIT
       003 A2
       004 A3
       005 STON
       006 HG
       007 A4
       008 H<
       009 HR<
       010 A5
       011 H<
       012 HFI<
       013 P
```

Figure B. Generalized PROC Stored As Item 'CHANGE' Which Will Perform Identical Operation

2.33 SAMPLE PROC: USING SSELECT AND COPY VERBS

This topic presents a sample PROC which uses the SSELECT and COPY verbs.

Figure A shows a sample operation at the TCL level using the SSELECT verb and then the COPY verb. This identical operation is performed by the PROC named TEST shown in Figure B. Upon execution of the TEST PROC, the output buffers contain the data shown in Figure C. Note that the SSELECT sentence is contained in the primary output buffer, while the secondary output buffer contains both input elements of the COPY operation, each terminated by a carriage return.

For further information regarding the SSELECT verb, refer to the ENGLISH Reference Manual. For further information regarding the COPY verb, refer to the Reality Programmer's Reference Manual.

```
:SSELECT INVENTORY WITH QOH > "900" BY-DSND QOH (CR)
19 ITEMS SELECTED
:COPY INVENTORY (CR)
TO: (HOLD-FILE) (CR)
19 ITEMS COPIED
```

Figure A. SSELECT and COPY Operation at TCL Level

```
item 'TEST' in M/DICT

001  PQ
002  HSSELECT INVENTORY WITH QOH > "900" BY-DSND QOH
003  STON
004  HCOPY INVENTORY<
005  H(HOLD-FILE)<
006  P
```

Figure B. PROC Stored as Item 'TEST' Which Performs Identical SSELECT and COPY Operations

```
PRIMARY OUTPUT BUFFER

| SSELECT INVENTORY WITH QOH > "900" BY-DSND QOH (CR)  |

SECONDARY OUTPUT BUFFER

| COPY INVENTORY  CR  (HOLD-FILE) (CR)                 |
```

Figure C. Output Buffers Upon Execution of TEST PROC

2.34 SAMPLE PROC: USING VARIABLE TESTING, GO AND D COMMANDS

This topic presents a sample PROC which uses variable testing, GO and D commands.

Figure A shows a sample tape positioning PROC. It differs from previous examples in that it uses the arithmetic command. It has practical value in that the user does not have to enter T-FWD at the TCL level for every file that is positioned over.

```
SPACE
PQ
C--THIS PROC WILL T-FWD A TAPE OF (NN) FILES
05 0
OSPECIFY NUMBER OF TAPE FILES TO T-FWD
OMUST BE TWO DIGITS+
S(1)
IP=
IF A = (2N) GO 10
GO 05
10 IF A = 00 GO 30
HT-ATT
P
HT-REW
P
20 D
HT-FWD
P
S(1)
-1
IF A # 00 GO 20
30 X...YOU ARE SPACED OUT...
```

Figure A. Sample Tape Positioning PROC

3 THE BATCH PROCESSOR

3.1 AN INTRODUCTION TO BATCH

The BATCH processor provides a facility for inputting, updating, and deleting items (or attributes) within Reality files.

BATCH operates via a predefined "BATCH-string" and an input line, used to update one or more items in multiple files simultaneously. The BATCH-string is stored as an item in a file and provides the dictionary function for the subsequent update. In other words, the BATCH processor ignores the attribute defining items defined for the designated files (i.e., which are used by the ENGLISH language processor), and instead relies on the BATCH-string to define the updating algorithm.

BATCH is evoked by issuing one of the following commands at the TCL level:

```
B/ADD file-name item-id
B/DEL file-name item-id
```

The file-name and item-id define the location of the specified BATCH-string. B/ADD in general defines an updating function (but may be used to delete items). B/DEL provides a reversing update function, in that using B/DEL on the identical BATCH-string and input line will in general negate or reverse the effects of a previous B/ADD operation. The exact nature of a B/DEL operation as opposed to a B/ADD operation will become clearer as the user reads the remaining topics within this section.

Once a valid B/ADD or B/DEL command has been entered, and the BATCH processor has gained control, the user is prompted for input lines at the terminal via the prompt character ">". Each input line entered by the user is processed separately by BATCH and generates a separate file update. BATCH continues to prompt for more input lines until the user exists by entering a null line (i.e., a carriage return only) following the BATCH prompt character.

Many users store BATCH-strings in the Master Dictionary (M/DICT), but this is not required. In fact, the recommended procedure is to define a separate file (or files) used exclusively to store BATCH-strings. These files should be single-level (i.e., Dictionaries) to save an additional file access in retrieving the BATCH-string. Also, the most common usage of the BATCH processor is from a PROC, where the B/ADD or B/DEL command has been constructed in the PROC's primary output buffer and the input lines to BATCH have been constructed in the PROC's secondary output buffer. For this reason, one must examine both the BATCH-string and its associated PROC to fully comprehend the resulting processing. (For a specific example illustrating the usage of BATCH in conjunction with a PROC, refer to the topic in this manual titled SAMPLE PROC: FILE UPDATE VIA BATCH.)

Figure A summarizes the general forms of the B/ADD and B/DEL commands. Figure B presents a number of examples.

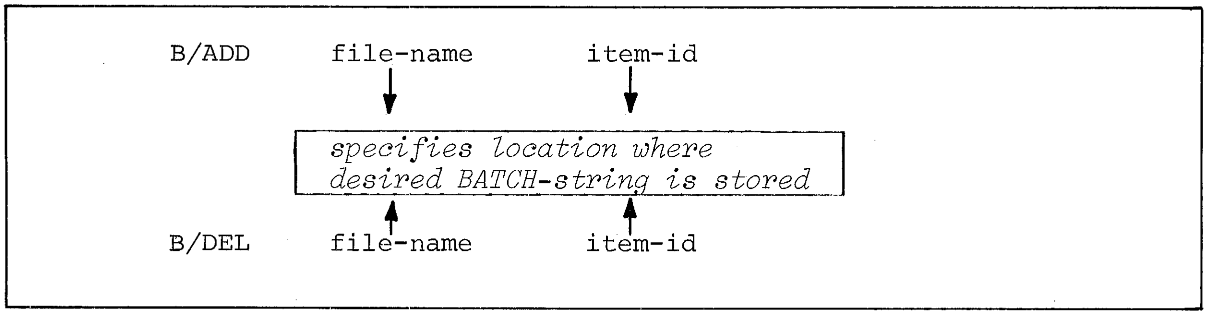


Figure A. General Form of BATCH Commands

EXAMPLE	EXPLANATION
:B/ADD BATCHFILE CHANGE CR >	*Evokes the BATCH processor using the predefined BATCH-string stored as item CHANGE in file BATCHFILE. The operation performed is a B/ADD operation further defined by the input line entered by the user immediately after the ">" prompt character.*
:B/DEL BATCH-STRINGS UPDATE CR >	*Evokes the BATCH processor using the predefined BATCH-string stored as item UPDATE in file BATCH-STRINGS. The operation performed is a B/DEL operation defined by the input line entered by the user immediately after the ">" prompt character.*

Figure B. Sample Usage of BATCH Commands

3.2 BATCH-STRINGS

> A BATCH-string is a definition of the updating algorithm to be performed. BATCH-strings are stored as Reality file items.

A BATCH-string consists of a set of "elements", one "element" per line (attribute). In general, a BATCH-string consists of the following elements for each file item to be updated: a file defining element, followed by one attribute defining element for each attribute, followed by a Z element. This sequence is repeated for each file item to be updated by the BATCH-string. (The Z elements separate the file item update sequences.)

Each element in the BATCH-string begins with a mnemonic identifying the element type (the sole exception being the file defining element which has no mnemonic and appears as the first element in the BATCH-string and as the first element following each F element). The mnemonic is optionally followed by modifiers and sub-elements delimited by commas. These modifiers and sub-elements are defined in subsequent topics which discuss each element type in detail.

The entire update resulting from a BATCH-string is accumulated and processes in a single step. Any errors encountered during the processing of the BATCH-string aborts the process and thus none of the updates occur. Additionally, multiple updates to the same file item cannot be performed by the same BATCH-string; the last file item update will override any previously generated updates by the BATCH-string to the same file item.

There is a one-to-one correspondence between atrribute defining elements in the BATCH-string and the defined attribute values themselves. In other words, the Attribute Mark Count (AMC) is not explicitly specified in the element, but is implied by the sequence of attribute defining elements. This principle is illustrated by the example shown in Figures A through D.

Figure A shows item 123 in file XYZ before the update takes place. The BATCH-string is stored as item UPDATE in file BATCH, as illustrated in Figure B. Figure C shows the B/ADD command and associated input line entered by the user. Finally, Figure D shows item 123 after the update has been performed. The user should note that the 123 in the input line (see Figure C) "feeds" the BATCH-string and provides the item-id to accompany file defining element XYZ,I (see Figure B), thus specifying that item 123 of file XYZ is to be updated. The two N elements in the BATCH-string (see Figure B) direct the BATCH processor to ignore (nop) the first two attribute values (ABC and DEF). The JKLM in the input line (see Figure C) "feeds" attribute defining element A,Y21 (see Figure B) and directs the BATCH processor to replace the old attribute value (GHI) with the new value (JKLM) from the input line, thus producing the updated item shown in Figure C. (Note that the BATCH-string in Figure B does not include a Z element since an update is being defined for one file item only.)

```
item 123 in file XYZ

001   ABC
002   DEF
003   GHI
```

Figure A. Item 123 Before Update

```
item UPDATE in file BATCH

001   XYZ,I◄──────────────────  file defining element
002   N◄──────────────────────  attribute defining element
003   N◄──────────────────────  attribute defining element
004   A,Y21◄──────────────────  attribute defining element
```

Figure B. BATCH-String Stored As Item UPDATE in File BATCH

```
:B/ADD BATCH UPDATE (CR)◄─────────  BATCH command
>123 JKLM (CR) ◄──────────────────  input line
>(CR) ◄───────────────────────────  terminates BATCH
```

Figure C. BATCH Command and Associated Input Line

```
item 123 in file XYZ
001   ABC
002   DEF
003   JKLM
```

Figure D. Item 123 After Update

3.3 INPUT LINES

BATCH-string elements reference fields in input lines. The user enters input lines in response to the ">" prompt character.

Once the BATCH processor has been evoked via a B/ADD or B/DEL command, the user is prompted for input lines at the terminal. Each input line entered by the user is used to generate a separate file item update. The BATCH processor continued to prompt for input lines until the user enters a null input line (carriage return only) at which point control returns to TCL.

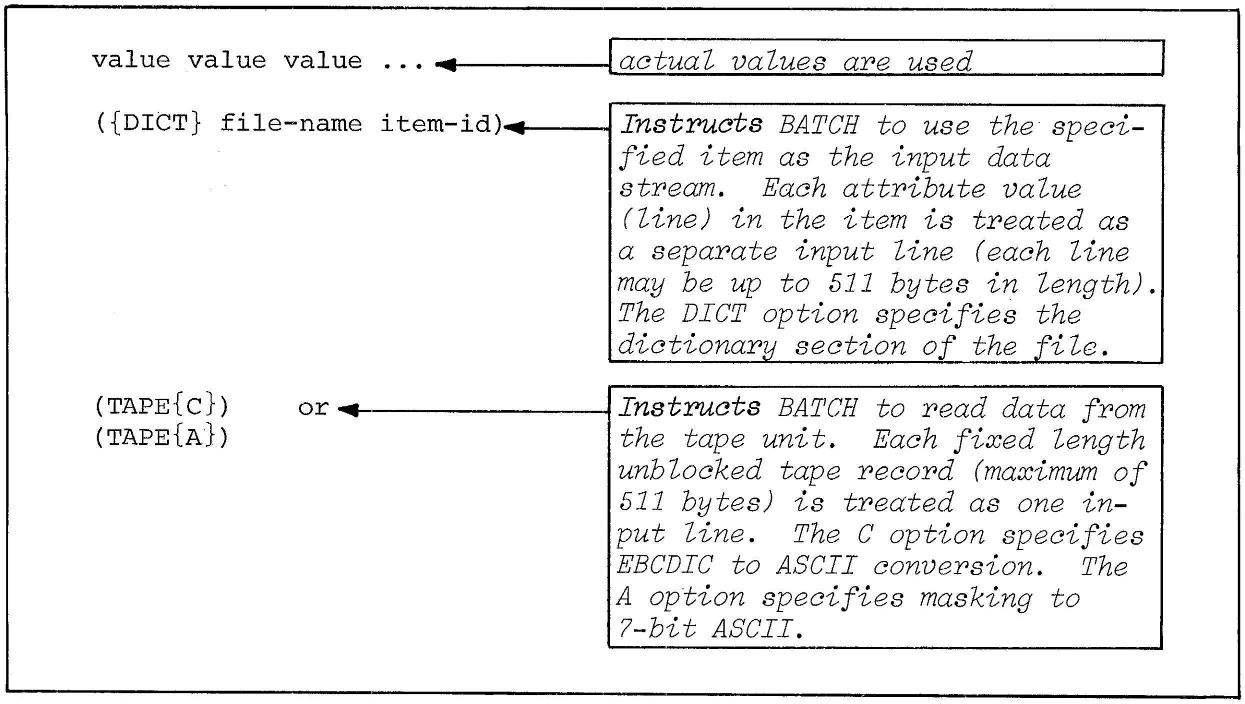
Values from an input line may be referenced by a BATCH-string element in either a free-field or fixed-field format. BATCH uses an input line pointer to keep track of data in the input line. Each file defining or attribute defining BATCH-string element uses one value from the input line; at the conclusion of this usage the input line pointer is advanced to the character immediately following the previously used value. Special BATCH-string elements are available which permit movement of this pointer so that the same value from the input-line may be used repetitively (refer to the topic titled ADDITIONAL BATCH-STRING ELEMENTS AND SUB-ELEMENTS).

An input line value referenced by a BATCH-string element in the free-field format is considered as starting with the first non-blank character at or after the current position of the input pointer, and continuing up to (but not including) the next blank. Values with imbedded blanks may be created in the input line by surrounding the entire value with single quotes (e.g., 'ABC DEF'), or by replacing the imbedded blank with a backslash (e.g., ABC\DEF). A backslash surrounded by blanks represents a null (missing) value.

Fixed-field input line values may be referenced by BATCH-string elements by specifying a starting column number and a field width. Fixed-field values always contain the number of characters specified and may contain embedded blanks. All leading and trailing blanks are deleted. Two or more contiguous blanks are considered to only one blank.

Rather than using explicit input line values, the first input line entered may optionally be one of special input line formats listed in Figure A. These special input line formats specify that the actual input values are to be obtained from a prestored file item, or from the attached magnetic tape unit.

Sample input lines are depicted in Figure B.

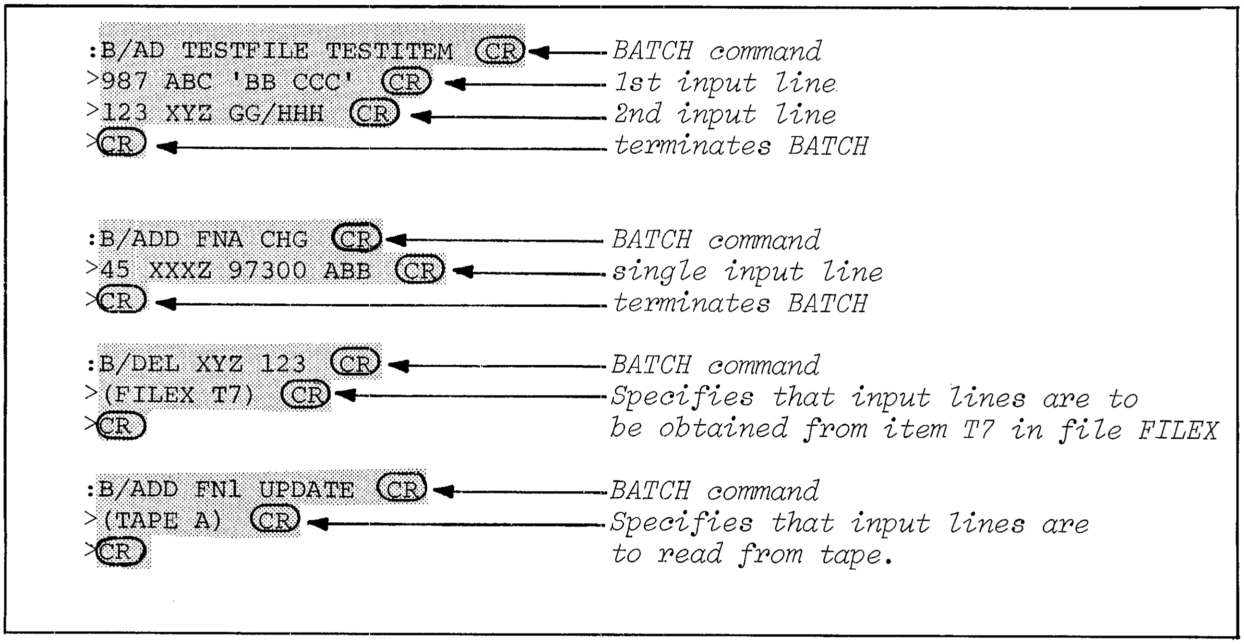


value value value ... ← *actual values are used*

({DICT} file-name item-id) ← *Instructs BATCH to use the specified item as the input data stream. Each attribute value (line) in the item is treated as a separate input line (each line may be up to 511 bytes in length). The DICT option specifies the dictionary section of the file.*

(TAPE{C}) or
(TAPE{A}) ← *Instructs BATCH to read data from the tape unit. Each fixed length unblocked tape record (maximum of 511 bytes) is treated as one input line. The C option specifies EBCDIC to ASCII conversion. The A option specifies masking to 7-bit ASCII.*

Figure A. General Form of Input Line



```
:B/AD TESTFILE TESTITEM (CR) ←——— BATCH command
>987 ABC 'BB CCC' (CR) ←————————— 1st input line
>123 XYZ GG/HHH (CR) ←——————————— 2nd input line
>(CR) ←—————————————————————————— terminates BATCH


:B/ADD FNA CHG (CR)←————————————— BATCH command
>45 XXXZ 97300 ABB (CR) ←———————— single input line
>(CR) ←—————————————————————————— terminates BATCH

:B/DEL XYZ 123 (CR) ←———————————— BATCH command
>(FILEX T7) (CR)←———————————————— Specifies that input lines are to
>(CR)                             be obtained from item T7 in file FILEX

:B/ADD FN1 UPDATE (CR) ←————————— BATCH command
>(TAPE A) (CR) ←————————————————— Specifies that input lines are
>(CR)                             to read from tape.
```

Figure B. Sample Input Lines

3.4 BATCH-STRING FILE DEFINING ELEMENTS

> The BATCH-string file defining element generates a specific item-id in a file to be updated.

The first element in a BATCH-string is a file defining element. A BATCH-string may contain additional file-defining elements immediately following a Z element. The first file specified is called the primary file; other files are called secondary files. The general form of the file defining element is as follows:

```
{DICT} file,code{(loc)}{,C(loc)}...{,conv}...{,bridge}...{,other}...
```

The "file" specification names the file containing the time to be updated. If the DICT option is used, then a dictionary item is updated.

The "code" specification may be any one of the following:

```
I  -  Update existing item or create new one.
N  -  Overwrite existing item.
V  -  Verify that item exists; abort otherwise.
X  -  Delete existing item.
A  -  Verify item does not exist; abort otherwise
```

The "(loc)" option immediately following the "code" specifies the location (in the input line) of the value used as the item-id. (If the "(loc)" option is not used, then free-field format is assumed, with the field used as the item-id beginning at the next nonblank input line character and continuing to the next blank.) The "(loc)" option may be any one of the following:

```
(m,n)  -  Use n characters starting at column m.
(m)    -  Use characters from column m up to first blank encountered.
(,n)   -  Use n characters starting at current pointer position.
p      -  Use p'th field of input line.
```

The "C(loc)" option specifies that another value is to be concatenated onto the end of the value specified by "(loc)", thus forming the desired item-id. Multiple "C(loc)" specifications are permitted. The "C(loc)" option may be any one of the following:

```
C(m,n)  -  Concatenate n characters starting at column n.
C(m)    -  Concatenate characters from column m up to first blank
           encountered.
C(,n)   -  Concatenate n characters starting at current pointer
           position.
Cp      -  Concatenate p'th field.
```

The "conv" option specifies input data conversions. Any conversion code defined in the CONVERSIONS section of the Reality ENGLISH Reference Manual may be used here (except for the Concatenate Code).

The "bridge" option may be either BC(n) or BV(n), and may be used for secondary files only. This option specifies that item-id is created (BC) or verified (BV) from the attribute value with Attribute-Mark-Count (AMC) "n", in the primary file item (in new item image). Multiple "bridge" specifications may be repeated to specify concatenated item-id's, or may be combined with I, J, K, or C sub-elements as needed. (Note: If the attribute is multi-valued, it is the first element of a concatenated "bridge" specification; D2 attributes should not be used with a "bridge" specification).

The "other" options may be any of the additional sub-elements described in the topic titled ADDITIONAL BATCH-STRING ELEMENTS AND SUB-ELEMENTS.

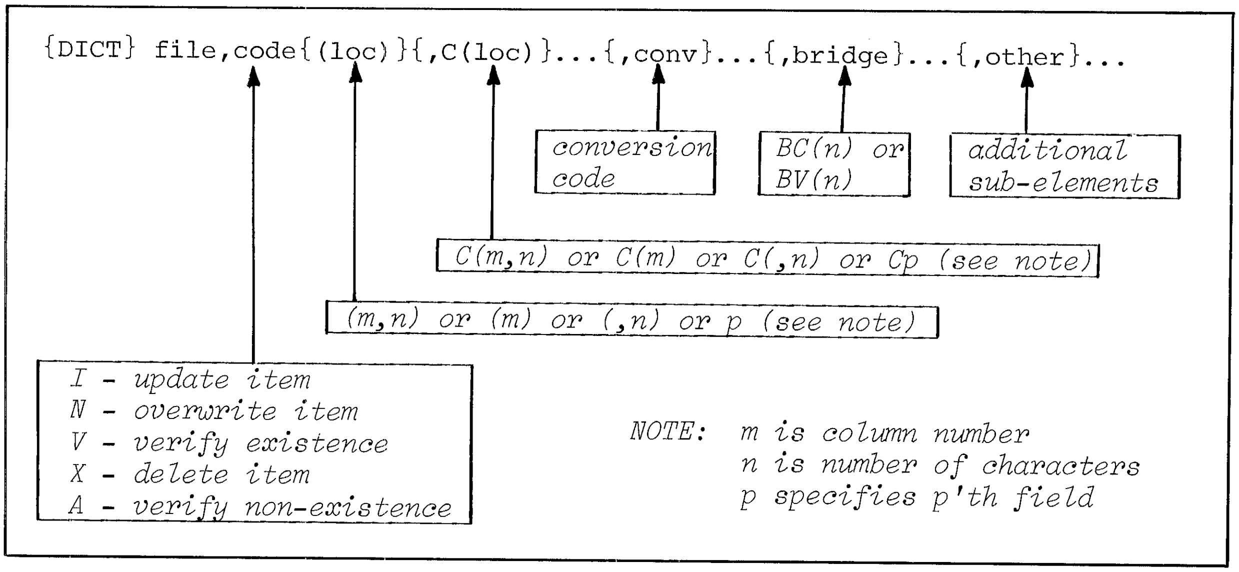


Figure A. General Form of BATCH-String File Defining Element

The examples below assume the following input line:

1234 ABC XYZ 987

FILE DEFINING ELEMENT	EXPLANATION
FNA,I	*Item with item-id 1234 in file FNA is updated.*
INVENTORY,X(6,2)	*Item with item-id AB in file INVENTORY is deleted.*
FILEX,I(3,2),C2	*Item with item-id 34ABC in file FILEX is updated.*
FNB,N,C(6,1),C(13,3)	*Item with item-id 1234A987 in file FNB is overwritten.*
FNB,N,C(6,1),C(13,3),MT	*Same as above, but Time Conversion is specified (see ENGLISH Reference Manual).*

Figure B. Sample Usage of BATCH-String File Defining Elements

3.5 BATCH-STRING ATTRIBUTE DEFINING ELEMENTS

The BATCH-string attribute defining elements define the operation performed on the corresponding attribute value.

Following the file defining element in a BATCH-string there must be one and only one) attribute defining element for each attribute to be updated. The attribute defining element and the actual attribute in the file have a one-to-one positional relationship. Attributes not updated beyond the last updated attribute need not be represented in the BATCH-string. There are five types of attribute defining elements: A, D, N, T, and X. The A type has the following general form:

```
A{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
```

The "(loc)" and "C(loc)" options define the value used from the input string; these specifications (as well as the "conv" and "other" options) are identical to the forms described for the BATCH-string file defining elements (refer to that topic). The "code" specification, which defines the operation performed, may be any one of the following:

Y11 - Store *unique* multiple values (i.e., won't store if value already present).

Y12 - Store *non-unique* multiple values.

Y21 - Replace a single value.

Y22 - Abort BATCH-string if *any* value already present; insert value if sttribute is null.

Y23 - Ignore if *any* value already present; insert value if attribute is null.

Y31 - Add value to attribute (subtract for B/DEL).

Y32 - Subtract value from attribute (add for B/DEL).

Y33 - Add 1 to attribute (subtract for B/DEL).

Y41 - Store multiple *unique* values in ascending ASCII collating sequence; not used with associative attributes.

Y42 - Same as Y41, but redundant values are also stored.

Additionally, a 4 may be suffixed to the "code" to place a "negative balance not permitted" restriction on the result (e.g., Y324).

If the attribute to be modified is an associative attribute (refer to the ENGLISH Reference Manual), then the D type is used. The D type takes on the following general forms:

```
D1;x{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
D2;x{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
```

The primary associative attribute is identified by D1;x. Subsequent secondary associative attributes are identified by D2;x. Three sets of associative attributes may be processed by BATCH (x is 1, 2, or 3, thus identifying the set). The remaining specifications of the D type attribute defining element are the same as defined for the A type above.

The N, T, and X type attribute defining elements are as follows:

N - Ignore corresponding attribute

nN - Ignore next n attributes

T - Add 1 to corresponding attribute (for both B/ADD and B/DEL).

X - Delete corresponding attribute

A complete BATCH example illustrating the use of the attribute defining elements is presented in the topic titled A COMPLETE BATCH EXAMPLE.

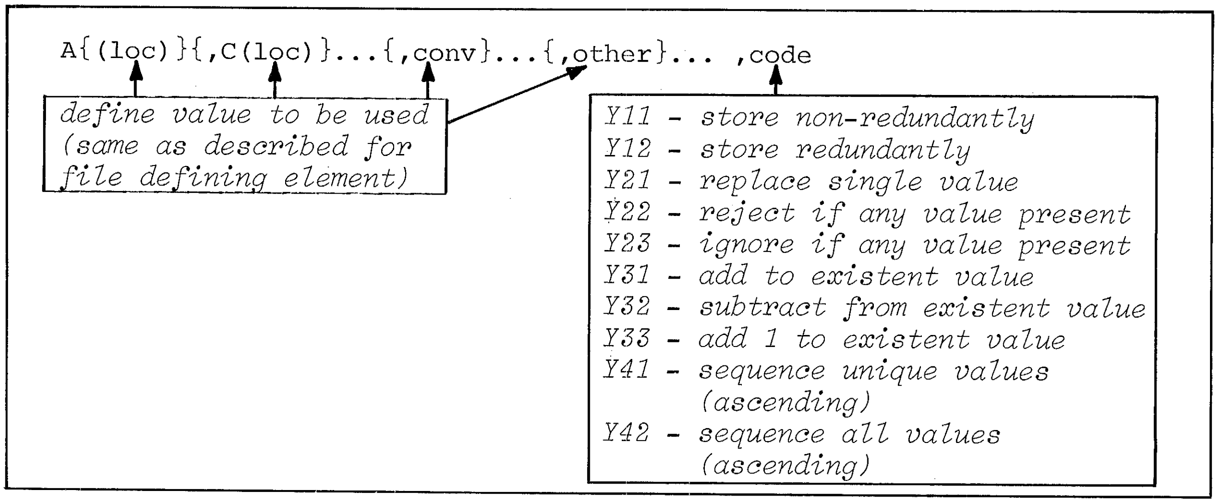

A{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
define value to be used (same as described for file defining element)
Y11 - store non-redundantly
Y12 - store redundantly
Y21 - replace single value
Y22 - reject if any value present
Y23 - ignore if any value present
Y31 - add to existent value
Y32 - subtract from existent value
Y33 - add 1 to existent value
Y41 - sequence unique values (ascending)
Y42 - sequence all values (ascending)

Figure A. General Form of BATCH-String Attribute Defining Element (A Type)

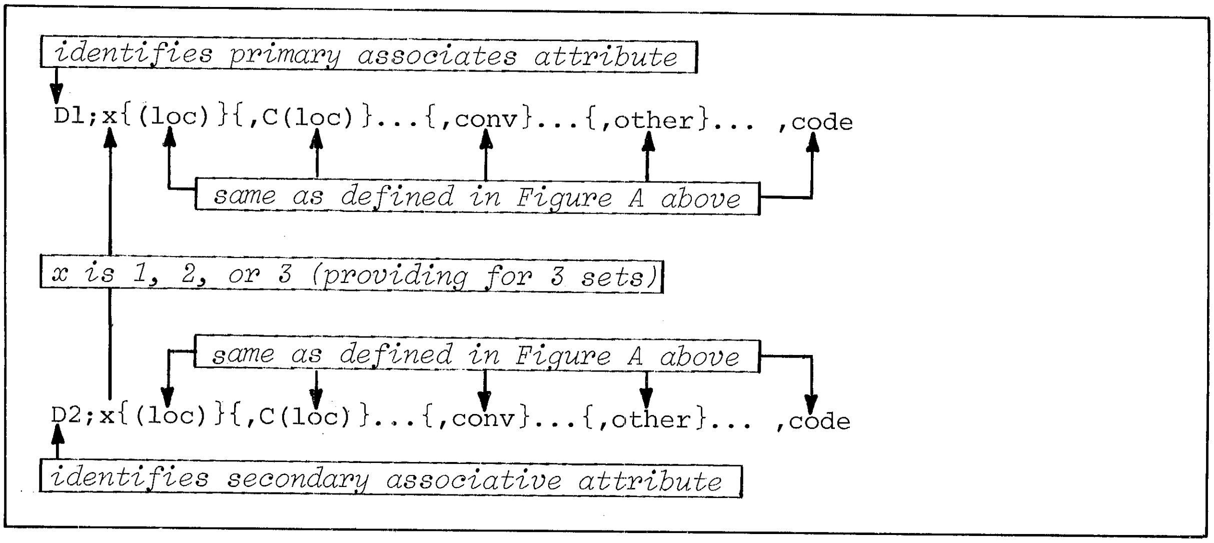

identifies primary associates attribute
D1;x{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
same as defined in Figure A above
x is 1, 2, or 3 (providing for 3 sets)
same as defined in Figure A above
D2;x{(loc)}{,C(loc)}...{,conv}...{,other}... ,code
identifies secondary associative attribute

Figure B. General Form of BATCH-String Attribute Defining Element (D Type)

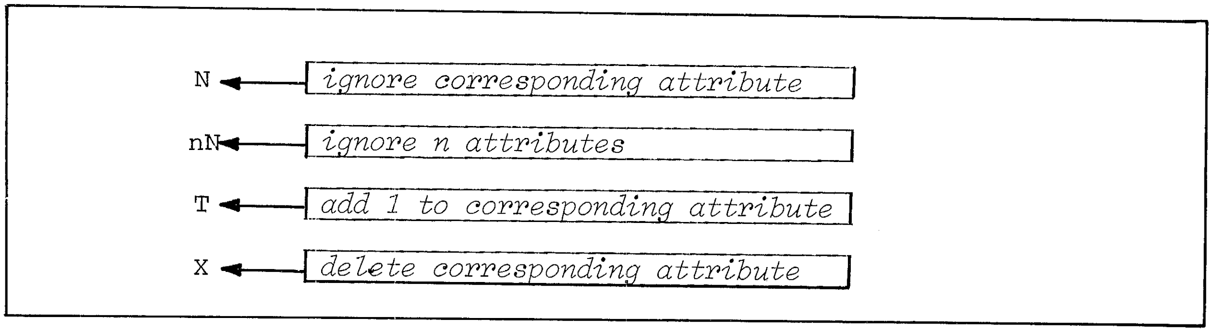

N
ignore corresponding attribute
nN
ignore n attributes
T
add 1 to corresponding attribute
X
delete corresponding attribute

Figure C. General Forms of BATCH-String Attribute Defining Elements (N, T, and X Types)

3.6 ADDITIONAL BATCH-STRING ELEMENTS AND SUB-ELEMENTS

In addition to file defining and attribute defining elements, four additional BATCH-string elements are provided. Additional sub-elements which may be used in both file defining and attribute defining BATCH-string elements are also provided.

Additional Elements

The following elements may be used in a BATCH-string.

F - Move the pointer forward to the next field in input line.
B - Move pointer backward one field in input line.
S(m) - Set pointer to m'th column in input line.
Z - Terminate a file item update section (must be immediately followed by another file item update section).

These elements are exemplified in the topic titled A COMPLETE BATCH EXAMPLE.

Additional Sub-Elements

The additional sub-elements described below may be used (unless otherwise specified) in both file defining and attribute defining BATCH-string elements.

The F sub-element specifies arithmetic operations on the input line data. Its general form is: {,Fop{(loc}}...

The "op" specification specifies the operation to be performed, as listed in Figure C. The "loc" specification specifies the *second* operand of the arithmetic operation; the *first* operand of the operation is the value which has previously been extracted (generated) by the BATCH-string element. The "loc" specification takes on the same general form as described for the attribute definition element (refer to that topic). For example, assuming an input line of "123 456" and an attribute defining element of "A2,F-(2,2),Y21" the resultant value stored would be "433".

The U sub-element exists to a user-defined subroutine. It's general form is: {,Uxxxx}

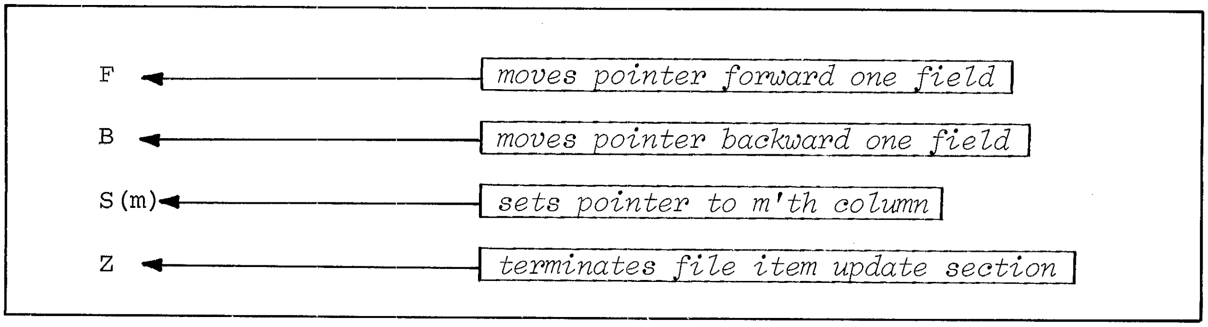
where xxx is the hexadecimal mode-id of the subroutine.

The SH and SD sub-elements store the system time and system date, repectively. The general forms are as follows: {,SH} {,SD}

No input data is necessary for either of these sub-elements if they immediately follow the A, D1, or D2.

The J sub-element has the following general form: {,J(n)}

This sub-element should only occur for secondary file attributes. It is ignored if the update is a B/ADD or if the update is a B/DEL with some specific data input to the field used by this element. If no data are input (meaning delete the entire value in the attribute), then the n'th primary file attribute is referenced, and the value that was deleted there is also deleted from the secondary file attribute. Only the first value used in the delete processing is used. Consequently, multi-valued primary attributes are in general processed incorrectly. The sub-element must be specified if the primary item itself is to be deleted. An example illustrating the use of the J sub-element is presented in the topic titled: SAMPLE USAGE OF B/DEL COMMAND WITH J SUB-ELEMENT.



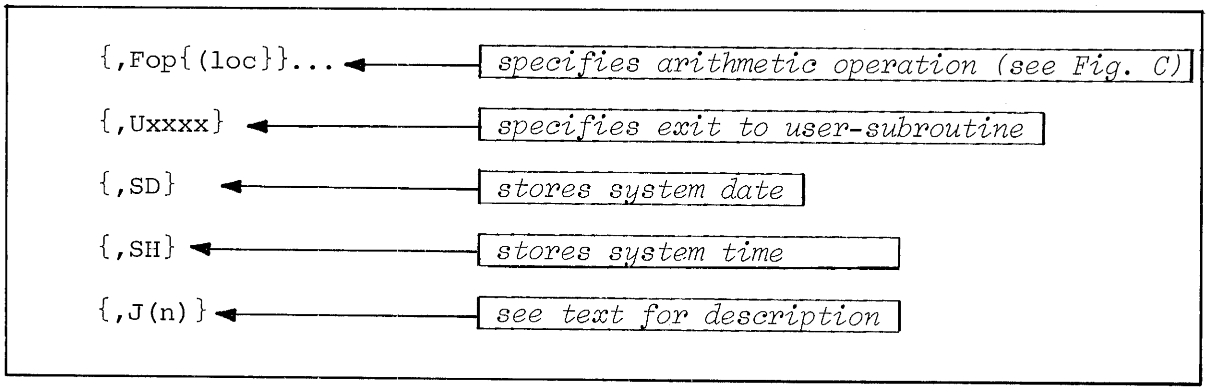
F — *moves pointer forward one field*

B — *moves pointer backward one field*

S(m) — *sets pointer to m'th column*

Z — *terminates file item update section*

Figure A. General Form of F, B, S(m), and Z BATCH-String Elements



{,Fop{(loc}}... — *specifies arithmetic operation (see Fig. C)*

{,Uxxxx} — *specifies exit to user-subroutine*

{,SD} — *stores system date*

{,SH} — *stores system time*

{,J(n)} — *see text for description*

Figure B. General Form of F, U, SD, SH, and J BATCH-String Sub-Elements

SYMBOL	OPERATION
+	*Addition*
-	*Subtraction*
/	*Division*
*	*Multiplication (if the n option is used, it must be a positive number and will be used as a scaling factor, i.e., the product will be divided by 10^n)*

Figure C. F Sub-Element "op" Specification Symbols

3.7 BATCH-LOCKS AND THE CONTINUATION CHARACTER

> A BATCH-lock is a "gate" which restricts concurrent access to a BATCH-string. The continuation character feature overcomes the 140-character restriction which normally applies to a BATCH input line.

BATCH-Locks

A BATCH-lock is a BATCH-string element which denies or permits access to the associated BATCH-string. If a potential user is denied access, he is put in a wait loop until access is permitted. The purpose of a BATCH-lock is to protect the current user until he has concluded execution of a BATCH-string (i.e., protects against concurrent access).

A BATCH-string is locked by a code upon entering, and unlocked automatically upon exiting. The format for the BATCH-lock is:

```
     L (code)
```

where "code is a number from 1 through 8 (inclusive).

All BATCH-strings using the same code are locked when any one of them is in use. Typically all BATCH-strings of an application system (e.g., payroll, inventory, etc.) share the same lock code, by user convention. A lock code should be the first line (element) of a BATCH-string, but may precede any file-definition element. Figure C illustrates a sample BATCH-string with lock code 5.

In the event that the execution of a BATCH-string is interrupted by an abnormal abort or by the BREAK key, an automatic unlock will not occur. A TCL verb to unlock all codes is provided. Its general form is:

```
     B/UNLOCK  {d}
```

where the "d" option is the decimal equivalent of an ASCII character to be used as the user's "lock-character". The lock character is displayed whenever the user's lock code is set into the lock state. For example, consider the following command issued at the TCL level:

```
     :B/UNLOCK 96 (CR)
```

This example displays a pound sign (#) whenever the lock code was set via execution of a BATCH-string. Note that the lock character option need only be established once.

Continuation Character

The continuation character feature is useful when using BATCH in conjunction with a PROC. The PROC output buffers are infinitely long; however, BATCH only accepts a maximum of 140 characters as an input line. To overcome this restriction, the user may place a second carriage return specification (<) in the PROC stack (refer to the topic describing the H command in the PROC section of this manual). A carriage return specification appearing as the next-to-last character in the stack is interpreted as a continuation character. For example:

```
1234 ABC DEF  <<
GHI JKL <<
```

The next-to-last "<" symbol is the continuation character and the last "<" symbol is the carriage return. The two lines in the example are processed as one.

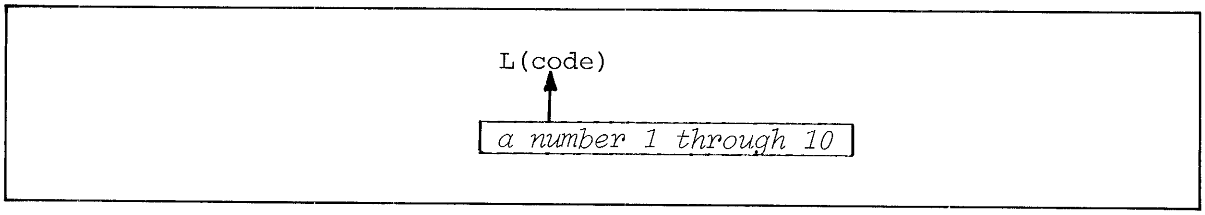


Figure A. General Form of BATCH-Lock Element

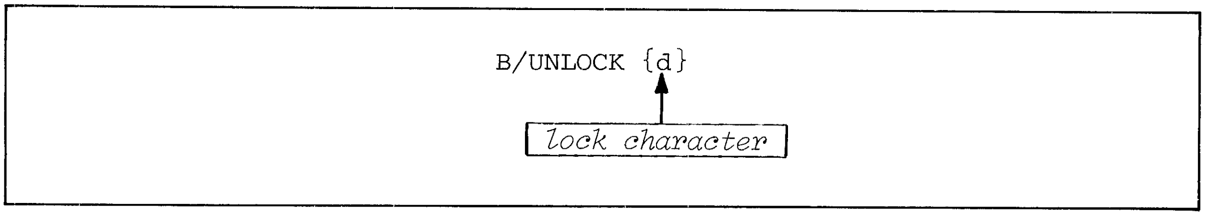


Figure B. General Form of B/UNLOCK TCL Verb

```
001  L(5)
002  ACCOUNT,I
003  A,Y21
004  11N
005  A,Y21
```

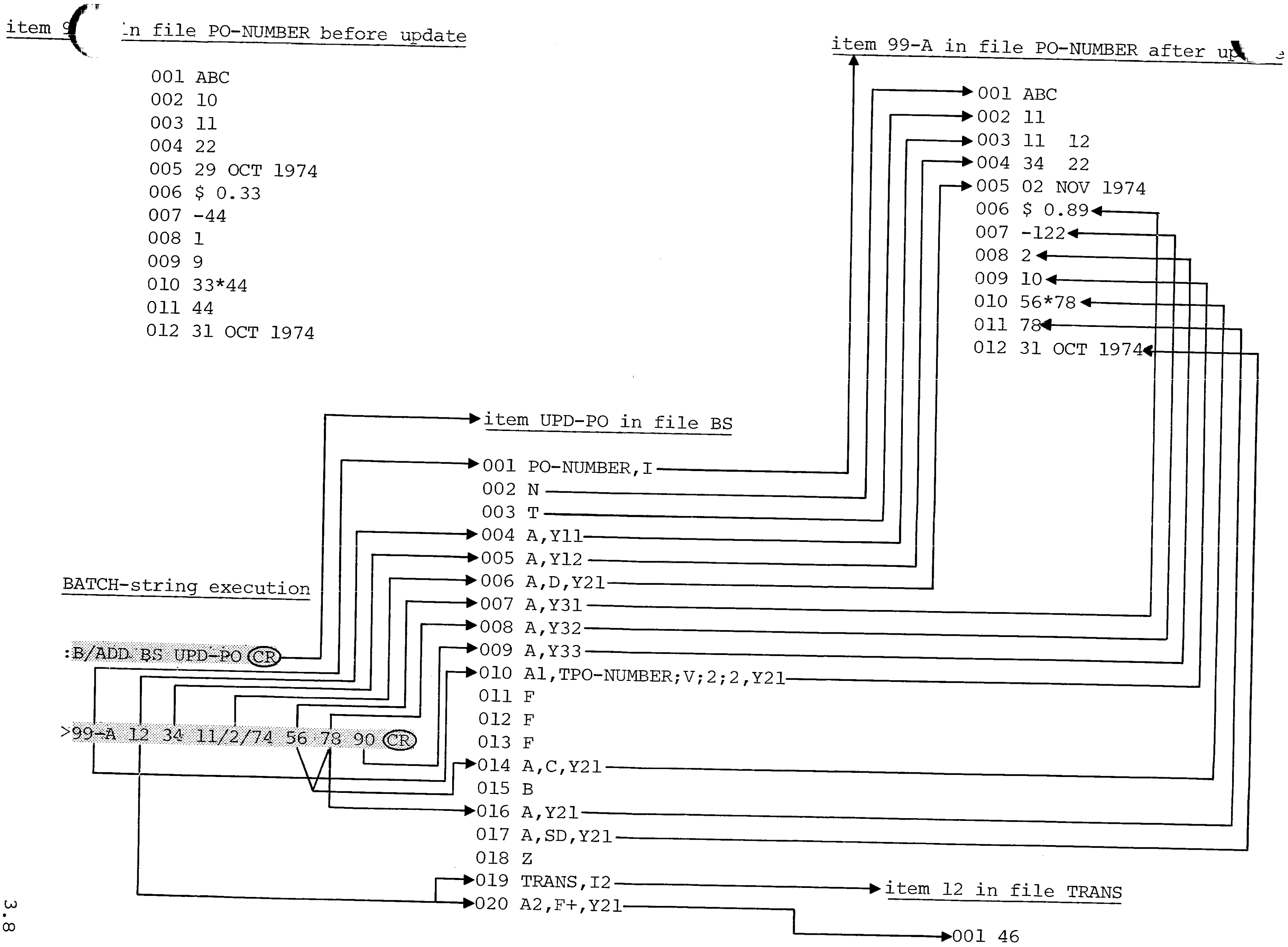
Figure C. Sample BATCH-String Using BATCH-Lock Element

3.8 A COMPLETE BATCH EXAMPLE

This topic presents a graphic example of a complete BATCH update operation.

The example in Figure A illustrates the update of item 99-A in the PO-NUMBER file. The update is accomplished via the BATCH-string stored as item UPD-PO in the BS file. The arrows illustrate the interrelationship among the input line, the BATCH-string elements, and the updated item. The reader should follow each arrow carefully, taking notice of how each BATCH-string element interacts with its specified input line field, thus producing the associated updates to item 99-A.

In addition, the reader should note than an item with item-id 12 in the TRANS FILE (a secondary file) is also created by this sample operation.



item 99-A in file PO-NUMBER before update

```
001 ABC
002 10
003 11
004 22
005 29 OCT 1974
006 $ 0.33
007 -44
008 1
009 9
010 33*44
011 44
012 31 OCT 1974
```

item 99-A in file PO-NUMBER after update

```
001 ABC
002 11
003 11  12
004 34  22
005 02 NOV 1974
006 $ 0.89
007 -122
008 2
009 10
010 56*78
011 78
012 31 OCT 1974
```

item UPD-PO in file BS

```
001 PO-NUMBER,I
002 N
003 T
004 A,Y11
005 A,Y12
006 A,D,Y21
007 A,Y31
008 A,Y32
009 A,Y33
010 A1,TPO-NUMBER;V;2;2,Y21
011 F
012 F
013 F
014 A,C,Y21
015 B
016 A,Y21
017 A,SD,Y21
018 Z
019 TRANS,I2
020 A2,F+,Y21
```

BATCH-string execution

```
:B/ADD BS UPD-PO (CR)

>99-A 12 34 11/2/74 56 78 90 (CR)
```

item 12 in file TRANS

```
001 46
```

Figure A. Sample BATCH-String Update

3.9 SAMPLE USAGE OF B/DEL COMMAND WITH J SUB-ELEMENT

The B/DEL command allows the user to delete specific values from attributes in an item.

In general, the B/DEL command is used to delete specified values from one or more single of multi-valued attributes, or to delete implied values (not specified) in single-valued attributes. In addition, such implied values may also be accessed by the "secondary file" section(s) of the BATCH-string. Thus it is possible to delete a value from an attribute in the primary file without knowing what the value on file is, and to use this same value as implied inputs to attribute defining elements in secondary files.

As a specific example, suppose that attribute 5 of the file ACCOUNT contains a value that is to be zeroed at the end of a month, and that this value is to be added in to attribute 6 of the file HISTORY. A representative BATCH-string is illustrated in Figure A.

If the value to be deleted is known (e.g., 99), then the input for this BATCH-string is as shown in Figure B (assuming an item-id of '123'). This input causes the value 99 to be subtracted from attribute 5 of the ACCOUNT file (see line 003), and causes the same value to be added to attribute 6 of the HISTORY file (see line 007). Note the reversal of the Y31 and Y32 specifications. In this case the "J(5)" sub-element in line 007 is not used.

If, however, the value to be deleted is not known (or if the user does not wish to enter the value, then the input is as shown in Figure C, or optionally as shown in Figure D. In this case the element in line 003 causes the deletion of the value in attribute 5 of ACCOUNT (as before); the "J(5)" sub-element is used to "link" back to the primary file (ACCOUNT) and to add in the value to the HISTORY file.

Note that the same effect could have been achieved (at significantly greater processing cost) by a suitable BATCH-string employing Translate Conversions to pick up the values to be deleted (refer to the ENGLISH Reference Manual). However, if the primary attribute is multi-valued and one of the values is deleted, the "J" sub-element must be used to delete the corresponding value in a secondary file.

If the primary item itself is deleted (using the ",X" option in the file defining element), the "J" sub-elements can add or delete values into secondary file attributes.

```
item CHANGE in file BATCH-FILE

     001 ACCOUNT,I
     002 4N
     003 A,Y31
     004 Z
     005 HISTORY,I1
     006 5N
     007 A2,J(5),Y32
```

Figure A. BATCH-String Stored as Item CHANGE In File BATCH-FILE

```
:B/DEL BATCH-FILE CHANGE (CR)
>123 99 (CR)
>(CR)
```

Figure B. Sample Input When Value Known

```
:B/DEL BATCH-FILE CHANGE (CR)
>123 (CR)
>(CR)
```

Figure C. Sample Input When Value Not Known

```
:B/DEL BATCH-FILE CHANGE (CR)
>123\ (CR)
>(CR)
```

Figure D. Optional Input When Value Not Known (Produces Same Results as Figure C)

3.10 SAMPLE USAGE OF BATCH WITH SELECT AND SSELECT VERB

This topic presents two examples of illustrating the use of BATCH in conjunction with the SELECT and SSELECT verbs. (For further information regarding these verbs, refer to the ENGLISH Reference Manual.)

When evoking the BATCH processor immediately after performing a SELECT or SSELECT operation, at least one "item-id substitution code" (i.e., an asterisk (*) character) must be in the BATCH input line in place of the normally explicit item-id. More than one asterisk is permitted, with each automatically being replaced by the item-id currently being processed. Only one BATCH input line is allowed after a SELECT or SSELECT operation.

Figure A illustrates a SELECT operation on the PAYROLL file. The BATCH-string stored as item DELETE in the dictionary of the PAYROLL file is then executed. The asterisk entered as the input line essentially causes each selected item-id to in turn be "picked up" and processed by the BATCH-string (i.e., all 5 selected items are deleted).

Figure B illustrates a SELECT operation on the PAYROLL file. The BATCH-string stored as item DELETE in the dictionary of the PAYROLL file is then executed. The asterisk entered as the input line essentially causes each selected item-id to in turn be "picked up" and processed by the BATCH-string (i.e., all 5 selected items are deleted).

Figure B illustrates an SSELECT operation on the ACCOUNT file. The BATCH-string stored as item UPDATE-ACCOUNT in the BATCHS file is then executed.

The user should note that the following error message will be displayed if at least one asterisk is not used in input line:

```
DATA INPUT LINE TO BATCH AFTER A SELECT MUST CONTAIN
AT LEAST ONE ITEM-ID SUBSTITUTION CODE (ASTERISK*)
```

```
:SELECT PAYROLL WITH CODE "T" CR

5 ITEMS SELECTED.

:B/ADD DICT PAYROLL DELETE CR

>* CR

'578A' UPDATED
'592A' UPDATED
'522A' UPDATED
'599A' UPDATED
'621A' UPDATED
```

Figure A. Sample Usage of BATCH With SELECT Verb

```
:SSELECT ACCOUNT WITH SEWER-ASMT CR

3 ITEMS SELECTED.

:B/ADD BATCHES UPDATE-ACCOUNT CR

>* 2.00 * 3.00 * 4.00 CR

'23070' UPDATED
'25025' UPDATED
'25050' UPDATED
```

Figure B. Sample Usage of BATCH With SSELECT Verb.

APPENDIX A

PROC COMMAND SUMMARY

This appendix presents the general form for each PROC command. The commands are listed in alphabetical order. For further information regarding a particular command, refer to the topic reference.

COMMAND FORMAT	TOPIC
A{c}{({m}{,n})x	2.8
A{c}{p}x	2.8
B	2.7
BO	2.7
C{text}	2.26
D(m){+}	2.12
D{p}{+}	2.12
({DICT} file-name {item-id})	2.29
F	2.7
F;E1;E2;En	2.23
F-CLEAR n{file}	2.19
F-DELETE X	2.20
F-FREE{XY}	2.22
F-OPEN n{file}	2.19
F-READ{XY}	2.20
F-UREAD{XY}	2.22
F-WRITE{XY}	2.20
FB(file)	2.21
FBU(file)	2.22
G n	2.14
GO n	2.14
GOSUB n	2.14
H{text}{<}	2.13
IF {#}a-cmnd proc-cmd	2.15
IF a-cmnd op (pattern) proc-cmnd	2.17
IF a-cmnd op string proc-cmnd	2.16
IH{text}	2.13
IN{r}	2.11
IP{B}{r}	2.11
IT{C}{A}	2.11
L E1,E2,En	2.24
MV operand operand	2.9
MVA operand operand	2.10
MVD operand operand	2.10
+n	2.26
-n	2.26
NA{C}{p}x	2.8
NB	2.7
NBO	2.7
NF	2.7
NH{text}{<}	2.13

COMMAND FORMAT	TOPIC
NIH{text}	2.13
NIN{r}	2.11
NIP{B}{r}	2.11
NIT{C}{A}	2.11
NS(m)	2.7
O{text}{+}	2.12
P{mode-id}	2.27
PP	2.27
PQ-COMPILE	2.30
PQ-SELECT	2.30
RI(m)	2.13
RI{p}	2.13
RO	2.13
RSUB n	2.14
RTN{X}	2.29
S(m)	2.7
SP	2.7
ST OFF	2.7
ST ON	2.7
T E1,E2,En	2.25
TR	2.26
Umode-id	2.28
X{text}	2.28

APPENDIX B

LIST OF ASCII CODES

This appendix presents a list of ASCII codes used in the Reality system.

DECIMAL	HEX	CHARACTER		SPECIAL USE IN REALITY
0	0	NUL		Null prompt character
1	1	SOH	^A	Cursor home on CRT Terminal
2	2	STX	^B	
3	3	ETX	^C	
4	4	EOT	^D	
5	5	ENQ	^E	
6	6	ACK	^F	Cursor forward on CRT Terminal
7	7	BEL	^G	Bell on CRT Terminal
8	8	BS	^H	
9	9	HT	^I	
10	A	LF	^J	Cursor down on CRT Terminal
11	B	VT	^K	Vertical address on CRT Terminal
12	C	FF	^L	Screen erase on CRT Terminal
13	D	CR	^M	Carriage return on CRT Terminal
14	E	SO	^N	
15	F	SI	^O	
16	10	DLE	^P	Horizontal address on CRT Terminal
17	11	DC1	^Q	
18	12	DC2	^R	
19	13	DC3	^S	
20	14	DC4	^T	
21	15	NAK	^U	Cursor back on CRT Terminal
22	16	SYN	^V	
23	17	ETB	^W	
24	18	CAN	^X	
25	19	EM	^Y	
26	1A	SUB	^Z	Cursor up on CRT Terminal
27	1B	ESC		
28	1C	FS		
29	1D	GS		
30	1E	RS		
31	1F	US		
32	20	SPACE		
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	30	0		

DECIMAL	HEX	CHARACTER	SPECIAL USE IN REALITY
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	\	
93	5D	]	
94	5E	¬	
95	5F	_	
123	7B	{	
124	7C	¦	
125	7D	}	
126	7E	~	
127	7F	DEL	
251	FB	SB	Start Buffer
252	FC	SVM	Secondary Value Mark
253	FD	VM	Value Mark
254	FE	AM	Attribute Mark
255	FF	SM	Segment Mark

APPENDIX C

ASSEMBLY INTERFACE FOR PROC USER EXITS

User exits from PROC may be used to reach special purpose assembly code.

A user program can gain control during execution of a PROC by using the UXXXX command where XXXX is the hex mode-id of the user program. The assembly program can perform special processing, and then return control to the PROC processor. This material reproduces what is contained in the ASSY Language Manual.

1 INPUT INTERFACE

PQFLG	B	Set, indicating that a PROC is being executed.	
BASE	D	Base of the M/DICT.	
MODULO	T	Modulo of the M/DICT.	
SEPAR	T	Separation of the M/DICT.	
PQBEG	S	Points one prior to the first PROC statement. This will be within the file in which the PROC resides.	
PQEND	S	Points to the terminal AM of the PROC.	
PQCUR	S	Points to the AM following the UXXXX statement.	
*R9	R	Points to the PROC control block. R9 may be altered within the code, with the consideration that elements defined relative to R9 (see PQ-CUR-IB) will not be available.	
*PQ-REG	S	Points to the PROC control block.	
IR	R	Same as PQCUR.	
PBUFBEG	S	Points to a buffer containing the primary and secondary input buffers. Format: (SB)...primary input...(SM)(SB)...secondary input...(SM). LOGON sets this to one frame. Additional frames are added as needed by subroutine PQNEXTOVF. Additional frames are released at LOGOFF.	
*ISBEG	S	Points one before the 1st character of the primary output buffer. This is the processor's "IS" work area. This buffer is and must always be terminated with a (SM).	
*STKBEG	S	Points one before the 1st character of the secondary output buffer (STACK). This is initially 2 linked frames, though more frames may be linked to it automatically by the subroutine PQNEXTOVF. Additional frames are released at LOGOFF. This buffer is and must be terminated by a (SM).	
*SFLG	B	Set if 'STACK ON'. Zero otherwise.	
IB	R	Input buffer pointer. Points to the current location in whichever is active: the primary or secondary input buffer.	
*PQ-CUR-IB	B	Set if IB points into the secondary input buffer. Zero otherwise. This bit is defined relative to register 9, which must be set to the PROC control block in order to reference this bit.	
SC2	C	Contains a blank.	
		SFLG OFF	SFLG ON
IS	R	The last byte of the primary output buffer.	The last byte of the secondary output buffer.
UPD	R	The last byte of the secondary output buffer.	The last byte of the primary output buffer.

An * notes a change from previous conventions. Note in particular that SFLG replaces SBIT, that attribute marks may separate parameters, and that a (SM) terminates each PROC buffer.

OUTPUT INTERFACE

IR	R	Points to the (AM) preceding the next PROC statement to be executed; may be altered to change the location of PROC execution.
IS	R	May be altered to reflect a changed output buffer, but the formats described above must be maintained. For example, SFLG must be kept accurate.
UPD	R	
IB	R	May be altered to reflect a changed output buffer, but the formats described above must be maintained. For example, PQ-CUR-BUF must be maintained.
	R	
SFLG	B	Set if "STACK ON" condition, otherwise zero.
PQ-CUR-IB	B	Set if secondary input buffer is active. Zero otherwise.
PROC BUFFERS		Must be maintained in the formats described in the input section. Note that a (SM) must terminate each buffer.
ABIT-ZBIT	B	Zero.
BASE	D	As at entry.
MOD	T	As at entry.
SEPAR	T	As at entry.
SC0	C	As at entry.
SC1	C	As at entry.
SC2	C	As at entry.
PQBEG	S	As at entry.
PQEND	S	As at entry.
PBUFBEG	S	As at entry.
ISBEG	S	As at entry.
TKBEG	S	As at entry.
PQFLG	B	Set if PROC execution is to continue.

The normal method of returning control to PROC is to execute an external branch to 2,PROC-I. If it is necessary to abort PROC control and exit to WRAPUP, set PQFLG off, and execute an external branch to the appropriate WRAPUP entry point.

1 PQNEXTOVF

A subroutine called PQNEXTOVF is provided so that user written code may add parameters to the PROC buffers without concern about exceeding the buffers. This is applicable to the primary and secondary input buffers, and the secondary output buffer. Additional frames will be added to these buffers as needed. This subroutine handles a forward link zero trap on registers 15 or 8. It is called automatically when XMODE contains the mode-id 'PQNEXTOVF'. Note that R15 and R8 are not destroyed by this routine.

2 INPUT INTERFACE

R15 or R8	R	Points to the end of the frame on which the forward link zero trap occurs.
R9	R	Set to PROC control block.

3 OUTPUT INTERFACE

R15 or R8	R	As at entry.

The output convention for ATTOVE is applicable. Note in particular that OVRFLW is 0 if no frame was available.

PQ-INSERT is a subroutine which will insert single or multiple fields into any of the PROC buffers.

INPUT INTERFACE:

PBUF Points to first byte of receiving field. Sending field(s) must be terminated by an attribute mark. Multiple field inserts can be accomplished by separating the sending fields by blanks.

IB Points to one byte before sending field

SB2 0 - Causes blanks in sending field to be replaced by attribute marks in the receiving field.
1 - Blanks are left as blanks

R9 Points to PROC control frame (MOV PQ-REG,R9)

OUTPUT INTERFACE:

PBUF and IB Point to first byte of receiving field if SB2 = 1
Point one byte before receiving field if SB2 = 0

INTERNAL USAGE:

BMS
R14
R15