*NASA TM-80144*

# NASA Technical Memorandum 80144

NASA-TM-80144 19800012559

## ISIS USERS MANUAL

CAROLYN GRANTHAM

MARCH 1980

**NASA**

National Aeronautics and
Space Administration

**Langley Research Center**
Hampton, Virginia 23665

## ABSTRACT

The Interactive Software Invocation System (ISIS) is an interactive data management system. ISIS is being developed to act as a buffer between the user and host computing system. ISIS provides the user with a powerful system for developing software systems in an interactive environment. ISIS protects the user from the idiosyncracies of the host computing system by providing such a complete range of capabilities that the user should have no need for direct access to the host computing system. These capabilities include a data editor, a file manager, and a tool invoker, all under the control of a PASCAL-like Interactive Programming Language (IPL).

N90-21043

## TABLE OF CONTENTS

# INTRODUCTION

This document covers the latest version of the Interactive Software

Invocation System (ISIS) (12/20/79) discussing the syntax and operation of

the Interactive Program Language, IPL.  IPL is based on the higher order

language PASCAL and anyone wishing to use ISIS should have a working knowledge

of PASCAL or have access to a PASCAL Manual (see ref. 1).  PASCAL was chosen

as the base language because of its simplicity and its wide range of capabilities.

IPL and PASCAL differences are discussed on page 12.  IPL contains most of

the arithmetic operations, functions, and control statements of a traditional

programming language.  This language has been extended to include statements

for data and text editing, file management, and tool invocation.  The editor

manipulates pages of text or data.  The IPL statements allow insertion,

deletion, replacement, and modification of lines of text or data.  The file

manager allows the user to save, access, and purge pages within a 5-level

hierarchical file system.  The tool invoker allows the user to communicate

with the host computer system.

ISIS is being developed by Dr. W. Joseph Berman on contract for Langley

Research Center (LaRC).  ISIS was originally developed under the CDC NOS-BE

operating system at the University of Virginia and is currently running under

the CDC NOS 1.3 operating system at LaRC.  The transportability of ISIS is

being tested by efforts to rehost it to an IBM 370 system and to a PDP-11

machine.  Anyone having an application for ISIS is welcome to use it with

the understanding that ISIS is not  a production system, but a developing system.

On the LaRC system, ISIS is stored on a direct access permanent file under

the user number 961300N.  The control statements to access and execute ISIS are:


```
/attach,isis/un=961300n. (CR)          - Retrieve the ISIS file.
/isis (CR)                             - Executes ISIS

  ISIS MONITOR    V 1.00      80/02/13. 15.02.18.
  15.02.20?

          .                            - Acceptable ISIS statements.
          .
          .

  15.02.47?stop (CR)                   - Terminate ISIS session.

ISIS TERMINATED.  (ADDRESS:     5)
- LOAD FL 043210     STACK FL 024570
/
```

INTERACTIVE PROGRAMMING LANGUAGE (IPL) SYNTAX

The IPL descriptions provided in subsequent sections have the following
format: The IPL statement syntax at the top of the page. The words appearing
in caps must be typed as shown, whereas, the words in small letters can be
replaced with appropriate information. A discussion of the statement, what
it does, and how to use it is followed by examples illustrating how the
statement can be used. See example page 18.

.The statements are discussed in groups according to capabilities. The
first group, Interactive Programming Statements, contains statements of a
traditional programming language. The second group, File Management Statements,
allows the user to save, replace, access, and purge information which is
contained within a 5-level hierarchical file system. The third group,
Text Editing Statements, contains statements which allows editing (Insertion,
Deletion, Replacing, and Modification) of lines of text or data which is
contained in the file system. The fourth group, Tool Invocation Statements ,
allows the user to communicate with the host computer system. The fifth,
and last group, Interrogation Statements, allows the user to make inquiries
of ISIS, relating to statements in any of the above groups.

Syntax Conventions

These conventions will be used in the IPL statements or in the discussions of the statements.

| | |
|---|---|
| ? | ISIS prompt - informs the user that ISIS is ready |
| > | ISIS indication that the current statement is not yet complete and more input is required |
| id | Identifier |
| id(s) | One or more identifiers separated by commas |
| ℓn | Line number |
| nℓ | A specific number of lines |
| [ ] | Optional information for command |
| {\|} | Command choices are enclosed in brackets separated by vertical slashes (\|) |
| col | Column number |
| inc | Line increment |
| ... | ISIS acknowledgement to the BREAK key |
| . | Separators in DATA BASE library page name |
| , | List separator |

9

BREAK Key

The BREAK key has two functions. It enables the user to discard a partially typed line and reenter it. This is in addition to the normal BKSP key. It also allows the user to terminate a command. The line to be reentered may be a command statement (statement verb) or a line in the ACTIVE page the user is in the process of editing. A good example would be if he made a typographical error in the text being inserted. The user hits the BREAK key, ISIS acknowledges this by printing 3 dots (...), reprompts with the line number for reentry of the line, and then the user retypes the information.

To terminate commands using the BREAK key, the user simply depresses the BREAK key after he receives the prompt for the next line number. No other information may appear on the line preceeding the BREAK key.

EXAMPLES:

```
INSERT 4/2
    4.      =THIS IS AN EXAMPLE
    6.      =PROGRAM ILLISTRATING(BR).*..
    6.      =PROGRAM ILLUSTRATING THE
    8.      =USE OF THE BREAK KEY
   10.      =FOR DISCARDING ALINE
   12.      =CURRENTLY BEINF(BR).*..
   12.      =CURRENTLY BEING TYPED
   14.      =(BR) *
INSERT TERMINATED.
 16.40.41?
            LIST
    4.      =THIS IS AN EXAMPLE
    6.      =PROGRAM ILLUSTRATING THE
    8.      =USE OF THE BREAK KEY
   10.      =FOR DISCARDING ALINE
   12.      =CURRENTLY BEING TYPED
 16.40.51?
```

- Mistake here in spelling. User depressed the BREAK key. ISIS responded with 3 dots and prompted the user for reentry of the line.
- Another mistake in spelling -
- User depressed the BREAK key to terminate the command.

- List the ACTIVE page to check correctness.

* - BREAK key ( (BR) ) is typed but does not echo back to terminal.

10

## ERRORS

Error indications in ISIS are similar to those in PASCAL. Syntax errors
are denoted by an error message line printed directly below the statement
containing faulty code. Like PASCAL, this message consists of an up arrow (∧)
under the statement column where the ISIS parser became confused. A short
message describing the problem follows rather than the error code numbers as
in PASCAL. These messages are intended to be self-explanatory. Several typical
examples of errors are shown below. A complete list of all possible error messages
is located in Appendix D.

```
13.57.51?VAR V: ARRAY[1..3] OF REAL;
13.58.57?VAR R,H: REAL;
13.59.23?    R=2; H=1;
13.59.52?   REPEAT
14.00.13?      V(R,H)=.333*PI*R*R*H;
XXXXXXXX        ↑ ':=' EXPECTED
```
- Parentheses were used for arrays instead of square brackets ([])

```
15.03.56?A1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
XXXXXXXX    ↑ UNRECOGNIZED STATEMENT
15.04.47?
```
- A1 frame has not been declared.

```
         FRAME A1,A2
15.05.21?A1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
XXXXXXXX  ↑ '=' OR ':' EXPECTED.
15.05.51?SHOW FRAMES
** NONE **
15.06.46?
         VAR X,Y,Z:REAL   X=10; Y=5;
XXXXXXXX                   ↑ ';' EXPECTED
15.07.49?SHOW VARS
'X        ': REAL;
'Y        ': REAL;
'Z        ': REAL;
15.08.21?PRINT Y,'=Y',X,'=X';
                   0=Y                      0=X
15.09.00?
```
- The TYPE (:STRING) is missing on the FRAME declaration.

- The statement terminator (;) is missing after the VAR declaration.

```
         VAR I,J,K:INT; I=3; J=2;
15.10.31?                X=2.5; Y=5;
15.11.18? Z=X*J; PRINT Z,'=Z'
5.00000000000000E+000=Z
15.12.02?   K=Y*I; PRINT K,'=K';
XXXXXXXX        ↑ INCOMPATIBLE TYPES
15.13.00?K=I*J;PRINT K,'=K'
      6=K
```
- Incompatible types (INT=REAL*INT)

## Programming Statements

### IPL/PASCAL Differences

IPL contains the PASCAL variable types, REAL, INTEGER, and BOOLEAN in abbreviated forms: REAL, INT, BOOL. ISIS has two data types not in PASCAL: STRING and KEY. ISIS deviates from PASCAL in not allowing CHARS and ALFA's but instead includes a type called STRING. This STRING type is similar to the PASCAL ALFA except that there is no set length on the STRING. A STRING contains alphanumeric information enclosed by quotation marks and may be assigned to a variable.

The other data type not in PASCAL is the KEY type. KEY is defined as a line number that is assigned to each line of code in the work frame. The KEY type allows the user to define a variable which may be used in the range of the edit commands of ISIS. KEY values are between 0.0000 and 999.9999.

In IPL a semicolon (;) placed at the end of a statement is optional if it is not followed by another statement on that line. If there is more than one statement on a line, then the semicolon (;) must be used to separate the two statements.

In IPL, each simple statement must be completed on a single line, unless explicitly continued to the next line by having a $ as the last nonblank character on the line. The maximum number of characters permitted on a line is limited to 133 characters.

Another IPL/PASCAL difference is in the assignment statement. IPL allows either = or := for assignments, whereas, PASCAL requires a :=.

Another difference of IPL is it does not use BEGIN and END's to surround compound statements. All that is required is an END to terminate a compound statement.

12

Comments are similar to those of PASCAL in that they begin with a (*;

however, unlike PASCAL, IPL comments are automatically ended with the end of

the line. This means the user doesn't have to close comments. It also means

that comments may not be followed by code nore are they automatically continued

on the next line.

Records cannot have CASE variants. IPL does not presently allow underscores

to be used as part of the variable name. IPL does not allow packing, does not

have a CASE statement, and does not allow sub range types.

# System Variables Used in Programming Statements

There are several ISIS system variables which have been made available to the user. These variables are contained in a record (ref. 1, p. 42) named SYSTEM. The SYSTEM record field identifiers are as follows:

| Identifier | Type | | Description |
|---|---|---|---|
| .VERBOSE | - BOOL | - | If TRUE, SET, and CLEAR statements print acknowledgement. If FALSE, no such acknowledgement is printed. |
| .DELTA | - KEY | - | The default range increment (inc) on the Text Editing Statement, INSERT (default value is 1) |
| .ALARM | - INT | - | Twenty-four hour clock alarm - when ALARM (clock time) is 0, then a message is printed to the user on the CRT- ***ALARM***. This may be useful to a person with a very tight schedule and a poor memory. |
| .CLOCK | - INT | - | Is the number of elapsed milliseconds in the current terminal session (read only variable) |
| .TIME | - STRING | - | Current day time (read only variable) |
| .DATE | - STRING | - | Current date (read only variable) |
| .F | - KEY | - | The first line number in a frame. |
| .L | - KEY | - | The last line number in a frame. |
| .C | - KEY | - | The current line number in a frame. |
| .K | - KEY | - | Current line number in a FOREACH loop only, otherwise it is zero. |
| .COUNT | - INT | - | Number of items in the range of the last editing command (other than FOREACH). |
| .USERNUM | - STRING | - | Current seven character user number. |

To obtain the information in this record, the user may inquire with SHOW ID SYSTEM, or PRINT SYSTEM. The SHOW ID SYSTEM will print out the field identifier names and types. The PRINT SYSTEM will print out the current values of these field identifiers.

14

EXAMPLE:

```
    11.54.29?.ALARM=1156                      - Set .ALARM for 11:56
    11.54.53?
            PRINT .CLOCK                       - Print SYSTEM.CLOCK
        133
    11.55.03?
            PRINT .TIME                        - Print SYSTEM.TIME
    11.55.14.
    11.55.15?
            PRINT .DATE                        - Print SYSTEM.DATE
    79/08/29.
    11.55.27?
            SHOW ID SYSTEM                     - Display SYSTEM record
VARIABLE
'SYSTEM      ': RECORD
                VERBOSE: BOOL;
                DELTA: KEY;
                ALARM: INT;
                K: READONLY (KEY);
                F: READONLY (KEY);
                L: READONLY (KEY);
                CLOCK: READONLY (INT);
                DATE: READONLY (STRING);
                TIME: READONLY (STRING);
            END;
    11.55.42?
            PRINT SYSTEM                       - Print SYSTEM record
    VERBOSE =       TRUE                         variables and values
    DELTA =    1.
    ALARM =          1156
    K =     0.
    F = '999.9999
    L =     0.
    CLOCK =          279
    DATE =  79/08/29.
    TIME =  11.56.06.
    ***** ALARM *****                          -Alarm message is
    11.56.07?                                    typed when the alarm
                                                 went off at 11:56
```

15

## Comments

[line of code]    [   (*comment   ]


    User comments may be added to program code.  A left parenthesis followed by an asterisk, (*, indicates the beginning of the comment.  The end of the comment is denoted by the end of the line.  This means that comments cannot be embedded in IPL statements.    Comments are not automatically continued on the next line.

## Programming Statements

These have been divided into two groups: the Declarative statements which describe program variables, and the Action statement which are the executable statements. It should be noted here that the IPL compiler collects all declarations first, allocates space for them and then puts them into a symbol table. This is done without regard to the program logic or the order in which they appear. An illustration of this is shown in the example below:

```
IF  X > Y    THEN    VAR Z:STRING;
             ELSE    VAR Z:REAL;

END;
```

You would expect only one of the declarations to be declared based on the program logic, but in actuality both declarations will be collected for allocating space and since a variable may be declared only once, the IPL compiler will consider this an error. IPL, being an interactive language, allows the user to make declarations at any time or anywhere in the program.

ABBREV    abbrev-id(s)  :  statement - verb

      The ABBREV statement allows the user to abbreviate ISIS statement verbs.
More than one abbreviation may be given to a single statement verb.  See
SHOW STATEMENTS for list of verbs that may abbreviated.  ABBREVS are cleared
by the ERASE statement.  ABBREV can only be used in the ACTIVE frame at present.

EXAMPLE:

```
ABBREV P,PR,W:PRINT
08.32.52?
          SHOW ABBREVS
'PR         ': PRINT
'P          ': PRINT
'W          ': PRINT
08.33.17?
          P .DATE
79/08/10.
08.33.39?
          PR .TIME
08.33.56.
08.33.57?
          W .CLOCK
        218
08.34.17?
          ERASE PR,W

08.34.31?SHOW ABBREVS
'P          ': PRINT
```

- Set P, PR, and W as abbreviation for PRINT

- Show abbreviations

- Use abbreviations in place of PRINT

Programming Statements


TYPE  type-id(s)  {=|:}  type-specification


     The TYPE statement is similar to the TYPE section of a PASCAL program.  It allows the user to specify a new TYPE for subsequent use in variable declaration statements.  The type specification consists of combining any of the system-provided types (INT,BOOL,REAL,STRING,KEY,ARRAY,RECORD) into RECORDS and ARRAYS, etc. to obtain a user-defined type.  Subsequent type specifications may involve previous user-defined types in addition to system-provided types.  Types are disposed of by the ERASE statement.  As each TYPE statement is processed, the types are immediately entered into tables.  No code is generated by this statement. This means that pages which contain TYPE statements are EXEC'ed repeatedly or those which have compilation errors and are re-executed will fail on the second execution because their types have been previously declared.  This may be overcome by preceding all TYPE statements with an ERASE statement for that type.


EXAMPLE:


```
SHOW TYPES
** NONE **
08.35.56?TYPE PERSONS:REAL;                              - Declare types
08.36.11?TYPE MESS : ARRAY[1..5] OF BOOL;
08.36.40?TYPE REC1 : RECORD NUM:INT; FLAG:BOOL; NAM:STRING; END;
08.37.24?TYPE RECM : ARRAY[1..3] OF REC1;
08.38.06?
          SHOW TYPES
'MESS        ': ARRAY [1..5] OF BOOL;                    - Display TYPE symbol table
'PERSONS     ': REAL;
'RECM        ': ARRAY [1..3] OF
                    RECORD
                        NUM: INT;
                        FLAG: BOOL;
                        NAM: STRING;
                    END;
'REC1        ': RECORD
                        NUM: INT;
                        FLAG: BOOL;
                        NAM: STRING;
                    END;
08.38.17?
```

19

VAR  var-id(s):  type-specification


The VAR statement is similar to the VAR section of a PASCAL program.  It
allows the user to assign a prespecified type (REAL, INT,BOOL,STRING,KEY
and user defined types) to program variables.  All program variables must be
declared in this manner.  If the user fails to declare all variables being
used, the ISIS system interrogates the user for the type of the undeclared
variable instead of aborting the command. Variables are eliminated using
the ERASE statement.

Declared variables are assigned default values by the ISIS system.
Integers and real numbers are set equal to zero, Booleans are set FALSE
and strings are empty  (zero length). As each VAR statement is processed,
the variables are immediately entered into tables.  No code is generated
by this statement.  This means that pages which contain VAR statements are
EXEC'd repeatedly as those which have compilation errors and are re-executed
will fail on the second execution because the variables have been previously
declared.  This may be overcome by preceding all VAR statements with an ERASE
statement for that variable.


EXAMPLE:

```
 14.08.08?SHOW VARS
** NONE **
 10.34.17?VAR X,Y,Z:REAL;                              - Declare variables
 10.34.41?VAR I,J:INT
 10.34.54?VAR B,C: BOOL;
 10.35.04?VAR HAM: STRING;
 10.35.16?TYPE RECM: ARRAY[1..3] OF BOOL;              - Define types
 10.35.42?VAR INPT: RECM;
 10.36.21?TYPE LOCK: RECORD UN:BOOL; SHUT:STRING; END;
 10.36.58?VAR BTHE: LOCK;

 10.37.11?SHOW VARS                                    - Display variables in
'BTHE       ': RECORD                                    symbol table to show
               UN: BOOL;                                 the new declared
               SHUT: STRING;                             Variables have been
             END;                                        included in the symbol
'B          ': BOOL;                                     table
'C          ': BOOL;
'HAM        ': STRING;
'INPT       ': ARRAY [1..3] OF BOOL;
'I          ': INT;
'J          ': INT;
'X          ': REAL;
'Y          ': REAL;
'Z          ': REAL;


 09.48.45?Z=A+Y;
```

| VAR A | : | REAL |
|-------|---|------|

↑ System prompt   ↑ User response

- Equation to be calculated contains
  the undefined variable, A.  The system
  interrogates users for type.  The user
  responds with type and execution continues.

ERASE     {abbrev-id(s)|type-id(s)|var-id(s)|frame-id(s)}

 

The ERASE statement removes the specified types, variable-ids or frame name from the identifier tables.  More than one id may be erased at one time with ids separated by a comma.  Caution should be exercised when using ERASE.  Erasure of a TYPE will not affect already defined variables of that type, but it will prevent the user from defining new variables of that type.  Also, note that erasure of the ACTIVE frame is <u>not</u> allowed.

EXAMPLE:

```
09.24.44?SHOW TYPES                    - Display existing types
'PERSONS    ': REAL;
'RECM       ': ARRAY [1..3] OF BOOL;
09.25.21?
          SHOW VARS                    - Display existing variables
'A         ': REAL;
'B         ': REAL;
'C         ': REAL;
'PROMPT    ': STRING;
'RESPONSE  ': STRING;
'X         ': INT;
'Y         ': REAL;
'Z         ': REAL;
09.25.43?
          ERASE PERSONS,A,B,C          - Erase types and variables from tables
09.26.15?
          SHOW TYPES
'RECM       ': ARRAY [1..3] OF BOOL;
09.26.30?
          SHOW VARS                    - Display types and variables again to show
'PROMPT    ': STRING;                    that the erased variables and types were
'RESPONSE  ': STRING;                    removed from the identifier table
'X         ': INT;
'Y         ': REAL;
'Z         ': REAL;
```

Programming Statements

var-id $\{=|:=\}$    arithmetic expression

    The assignment statement consists of a program variable, an = sign and an expression. The resultant value of the expression is assigned to the variable on the left hand side of the equal sign.

    Types must be compatible as there is not implicit conversion in IPL.

EXAMPLES:

    A = X + Y

    B := Z * (A-1)

Programming Statements

EXITIF   conditional

The EXITIF statement allows the user to exit from the middle of a loop statement (IF, WHILE, REPEAT, LOOP, AND FOR) when a specified condition becomes true.  The EXITIF may appear anywhere in the loop.  A single loop may contain any number of EXITIFs.

EXAMPLE:

```
.1.27.4/YLEC)
   1.     =VAR R,T,X:REAL;
   3.     =R=10.
   5.     =T=.1745329;
   7.     =WHILE T < 3.14 DO
   9.     =       X=R*COS(T);
  11.     =       PRINTLN T*57.295780,'=T(DEG)',X,'=X';
  13.     =       T=T+.1745329;
  15.     =     EXITIF X <= 0;
  17.     =END
.1.36.2-?
        EXEC
```

– A WHILE loop
  containing an
  EXITIF statement
  (residing in ACTIVE
  page)

– Execute ACTIVE page

```
9.9999866411620E+000=T(DEG)   9.8480775738804E+000=X
1.9999997282324E+001=T(DEG)   9.3969263802333E+000=X
2.9999095923486E+001=T(DEG)   8.6602544158358E+000=X
3.9999494564648E+001=T(DEG)   7.6604450791050E+000=X
4.9999093205813E+001=T(DEG)   6.4278770520596E+000=X
5.9999961840972E+001=T(DEG)   5.0000013094009E+000=X
6.9999990483134E+001=T(DEG)   3.4202030908371E+000=X
7.9999998912929E+001=T(DEG)   1.7364837619970E+000=X
8.9999987770438E+001=T(DEG)   2.2679489403726E-006=X
9.9999986411620E+001=T(DEG)  -1.7364792950096E+000=X
```

← The output shows
  the WHILE loop was
  exited via the
  EXITIF (x becomes
  < 0)

23

IF    condition    THEN    $\left\{\begin{array}{l}\text{statement(s)} \quad [\text{ELSE statement(s)}] \\ \text{[EXITIF condition]}\end{array}\right\}$    END

The IF statement allows for conditional execution of          statements.
The condition must evaluate to a BOOL    value.  If the condition is TRUE,
the statements following the THEN are executed and those following the ELSE
(if present) are skipped.  If the condition is FALSE, the statements following
the THEN are skipped and those following the ELSE (if present) are executed.

It should be noted that ISIS deviates from PASCAL by not requiring
BEGIN . . . END's around the THEN and ELSE sections of the IF statement when
multiple statements are contained in them.
EXITIF in either the THEN or ELSE clause transfers to the END of the entire
IF statement.

EXAMPLE:

```
15.08.59?LOOP
15.09.10>    IF B<A THEN PRINTLN B; B=B+1;
15.09.39>           ELSE IF B=A THEN PRINTLN 'ONE MORE STEP';
15.10.06>                       ELSE PRINTLN ' READY TO STOP';
15.10.29>                       END
15.10.36>               B=B+1
15.10.46>       · END
15.10.53>    EXITIF B>16
15.11.05>END
 1.00000000000000E+001
 1.10000000000000E+001
 1.20000000000000E+001
 1.30000000000000E+001
 1.40000000000000E+001
ONE MORE STEP
 READY TO STOP
```

Programming Statements

FOR    var-id = initial-value    {TO|DOWNTO}    final-value  DO  { statement(s) }END
                                                                 { [EXITIF condition] }

The FOR statement is another form of loop statement which allows the user to perform a sequence of statements repeatedly while the variable-id takes on a progression of values between an initial and final value. This progression may go either upward or downward in value. The initial-value and final-value may be INT variables, literals, or expressions.

EXAMPLE:

```
          VAR A : REAL
10.46.11?VAR Z : ARRAY[1..10] OF REAL;
10.46.36?VAR L,I : INT;
10.46.52?  I=-5;
10.47.15?   FOR L=I+15 DOWNTO 1 DO
10.47.55>          A=L*L;
10.48.19>          Z[L]=3.14*A;
10.48.47>          PRINTLN L,Z[L];
10.49.11>   END;
        10  3.140000000000E+002
         9  2.543400000000E+002
         8  2.009600000000E+002
         7  1.538600000000E+002
         6  1.130400000000E+002
         5  7.850000000000E+001
         4  5.024000000000E+001
         3  2.826000000000E+001
         2  1.256000000000E+001
         1  3.140000000000E+000
```

— For statement with downward progression (10 to 1)

25

LOOP    [statement(s)]    EXITIF    condition    [statement(s)]    END

The LOOP statement is a generalization of the WHILE and REPEAT statements.
A set of statements are executed repetitively until the condition of the EXITIF
becomes true.  This EXITIF becomes part of the loop statement.  EXITIF may
appear anywhere in the loop and when the condition becomes true, the loop is
exited at that point in the code.  If the EXITIF is left out, the LOOP will
be executed infinitely.  If this occurs, the user can abort the command by
depressing the BREAK key.

EXAMPLE:

```
10.53.40?VAR X:REAL;   VAR S:STRING
10.54.15?SHOW VARS
'A         ': REAL;
'I         ': INT;
'L         ': INT;
'S         ': STRING;
'X         ': REAL;
'Z         ': ARRAY [1..10] OF REAL;
10.54.24?VAR B:BOOL;

10.59.19?  X=45.45; I=99;  B=TRUE;
10.59.40?
          LOOP
10.59.59?     ASK S,'PLEASE TYPE INPUT EXP *'
11.00.35>     EXITIF S='0'
11.00.54>     XEQ CAT('PRINTLN ',S)
11.01.19>END
PLEASE TYPE INPUT EXP * 6+6*2
        18
PLEASE TYPE INPUT EXP * X+2
   4.745000000000E+001
PLEASE TYPE INPUT EXP * I
       99
PLEASE TYPE INPUT EXP * B
      TRUE
PLEASE TYPE INPUT EXP * 5
        5
PLEASE TYPE INPUT EXP * 0
```

Loop of statements is executed until
the user types an input of S = 0.

$$\text{WHILE} \quad \text{condition DO} \left\{ \begin{array}{c} \text{statement(s)} \\ \text{[EXITIF condition]} \end{array} \right\} \text{END}$$

The WHILE statement is a type of LOOP statement. The statements contained in the loop will be executed WHILE a certain conditions exists. The WHILE statement evaluates a condition, which must reduce to a BOOL result. If the condition is FALSE, the statements are skipped. If the condition is TRUE, the statements are executed and the condition is then re-evaluated. If the condition is again TRUE, the statements are re-executed and the condition is then re-evaluated. This process continues until the condition is FALSE. When this happens, the statements are skipped and execution continues with the next statement.

EXAMPLE:

```
12.21.18?
12.21.50?VAR S:STRING
12.22.16?VAR I:INT
12.22.26?  S='CHECK WHILE'
12.22.43?  I=1
12.22.49?  WHILE I<10 DO PRINTLN SUB(S,I,1):2*I
12.23.29>                I=I+1
12.23.38>  END
C
  H
    E
      C
        K

          W
            H
              I
```

REPEAT $\left\{ \begin{array}{l} \text{statement(s)} \\ \text{[EXITIF condition]} \end{array} \right\}$ UNTIL  condition

    The REPEAT statement is a type of LOOP statement.  The statements contained in the loop will be repeated UNTIL a condition occurs.  The REPEAT statement is similar to the WHILE statement.  The differences are that the REPEAT statement first executes the statements it controls and then evaluates and checks the condition, and that the statements are re-executed as long as the condition is FALSE, i.e., the statements will always be executed at least once.

EXAMPLE:

```
15.20.43?VAR V:ARRAY[1..6,1..6] OF REAL;
15.21.19?VAR I,J:INT;
        VAR R,H,PI: REAL
15.21.33?  R=2; H=1; PI=3.14; I=1; J=1;

15.22.24?REPEAT
15.26.46?    V[I,J]=.333*PI*R*R*H;
15.27.12>    J=J+1;
15.27.21>    H=H+.5;
15.27.31>    PRINTLN H,V[I,J];
15.27.49>    UNTIL J=6;
 1.5000000000000E+000   6.2737200000000E+000
 2.0000000000000E+000   8.3649600000000E+000
 2.5000000000000E+000   1.0456200000000E+001
 3.0000000000000E+000   1.2547440000000E+001
 3.5000000000000E+000                        0
```

Programming Statements

[frame-id/]FOREACH string-var DO { statement(s) / [EXITIF condition] } END;

FOREACH allows the user to execute a set of statements for each line contained in the active frame or specified frame.

EXAMPLE:

```
1.    =ERASE S; VAR S:STRING;
2.    =SHOW PAGES HALNELL.PCODE.CONTROL. . :KEEP;
3.    =SHOWN/FOREACH S DO
4.    =XEQ CAT('MYJUNK/USE ',S);
5.    =MYJUNK/COUNT;          (*   ANY EDITTING COMMAND
6.    =END;
 13.46.08?EXEC
7 ITEMS INSERTED. LAST ITEM INSERTED IN SHOWN      :   7.
HALNELL.PCODE.CONTROL.CARD.EXEC USED AS MYJUNK
12 ITEMS IN SPECIFIED RANGE.
 H=12, I=0
HALNELL.PCODE.CONTROL.CARD.RUN USED AS MYJUNK
21 ITEMS IN SPECIFIED RANGE.
 H=21, I=0
HALNELL.PCODE.CONTROL.CARD.RERUN USED AS MYJUNK
20 ITEMS IN SPECIFIED RANGE.
 H=20, I=0
HALNELL.PCODE.CONTROL.CARD.REEXEC USED AS MYJUNK
12 ITEMS IN SPECIFIED RANGE.
 H=12, I=0
HALNELL.PCODE.CONTROL.CARD.EXECOLD USED AS MYJUNK
11 ITEMS IN SPECIFIED RANGE.
 H=11, I=0
HALNELL.PCODE.CONTROL.WILL.RUN USED AS MYJUNK
23 ITEMS IN SPECIFIED RANGE.
 H=23, I=0
HALNELL.PCODE.CONTROL.WILL.EXEC USED AS MYJUNK
12 ITEMS IN SPECIFIED RANGE.
 H=12, I=0
 13.47.04?
```

XEQ    string-expression


The XEQ statement allows the user to specify that the contents of a string-expression are to be interpreted as a command to the system.


EXAMPLE:

```
12.16.39?
12.18.01?VAR S:STRING
12.18.21?VAR ABC:REAL
12.18.30?   S='1+2+3'
12.18.46?   ABC=10
12.18.56?
        XEQ CAT('PRINTLN ',S)
        6
12.19.22?
        XEQ 'PRINTLN S'
1+2+3
12.19.47?
        S='ABC'
12.20.26?
        XEQ CAT('PRINTLN ',S)
  1.0000000000000E+001
```

- PRINTLN is concatenated (CAT) with contents of S (1+2+3) and then executed (XEQ). PRINTLN 1+2+3 evaluates expression and prints the value (6).

- PRINTLN S is executed, printing value of S, which is 1+2+3

- Redefine S

- PRINTLN is concatenated with contents of S (ABC) then executed (XEQ). PRINTLN ABC evaluates ABC and prints the value of the variable (10).

```
S='FOR I=1 TO 5 DO PRINTLN I END';
  13.31.43?XEQ S
        1
        2
        3
        4
        5
```

- User can   assign   a number of statements to a string variable and then execute these statements by executing that string variable (XEQ S)

SET TAG   tag-id

CLEAR TAG


     SET TAG allows the user to assign an identifier to a line of code.
This gives the user a way to keep track of the program modifications
made during an editing session.  After setting a tag, all lines of code
modified or added will bear the tag-id.  This tag-id could be the date
the modifications are made, or the name of the person making the modifi-
cations.  All lines modified will continue to contain this tag-id until
the user gets rid of it.  The tag may be changed at any time by executing another
SET TAG command and from there on the new tag-id is appended to the
modified code.  The CLEAR TAG command clears the tag-id and no more tags
will be placed on modified lines of code.  The modified frames <u>with</u>
the <u>tag-ids</u> may be viewed by listing the code  with the tag option on

(LIST {range} :T).  The tag-id is limited to seven characters.  There is
also a limit of 63 different tags in any one frame.


EXAMPLE:

```
 13.27.15?USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS WORK
 13.27.43?LIST 1.3/1.5                              - List existing text
   1.3   =    WDBLANKS = '            ';
   1.4   =
   1.5   =    IDLEN = 10;
 13.28.09?
        SET TAG CXG
WORK     NOW TAGGED AS: CXG                         - SET TAG
 13.28.43?INSERT 1.41//.01                          - INSERT new code
   1.41  =INSERT 3 LINES AT 1.4
   1.42  = WHICH WILL BE TAGGED
   1.43  =  WITH CXG!!!
   1.44  =

INSERT TERMINATED.
 13.30.49?LIST 1.3/1.5:T                            - List with T option.
        1.3   =    WDBLANKS = '         ';    Modifications have been
        1.4   =                               tagged.
CXG     1.41  =INSERT 3 LINES AT 1.4
CXG     1.42  = WHICH WILL BE TAGGED
CXG     1.43  =  WITH CXG!!!
        1.5   =    IDLEN = 10;
```

SET TAG    continued


```
13.32.00?
          SET TAG RWW
WORK      NOW TAGGED AS: RWW                         - Change the TAG
 13.32.45?
          CHANGE 'CXG' TO 'RWW TODAY' IN 1.43  - Modify code
   1.43  =  WITH RWW TODAY!!!
 13.33.41?
          LIST 1.3/1.5:T                             - List with TAG option.
          1.3   =    WDBLANKS = '           ';          Modifications have been
          1.4   =                                       tagged.  Previous tags
CXG       1.41  =INSERT 3 LINES AT 1.4                  are retained.
CXG       1.42  =   WHICH WILL BE TAGGED
RWW       1.43  =   WITH RWW TODAY!!!
          1.5   =     IDLEN = 10;
 13.33.59?
          CLEAR TAG
WORK       NO LONGER TAGGED.
```

Programming Statements

SET TRACE        var-id(s)

CLEAR TRACE    var-id(s)


The SET TRACE statement is used to trace variables.  Each time the
variable's value is assigned, the variable-id and the new value are printed.
A TRACE applies to entire program.  Once a variable is traced, it will be
traced wherever it is used.  Records and array variables can be traced but
no values are printed.


The CLEAR TRACE command releases a TRACE on a variable.


EXAMPLE:


```
11.31.52?VAR I,J:INT
11.32.10?VAR X:REAL
11.32.21?VAR B:BOOL;
11.32.28?VAR S:STRING
11.22.36?    J=1;       X=0;
11.22.56?
        SET TRACE J                      - Set trace on J
'J       ' NOW TRACED.
 11.23.15?SET TRACE X                    - Set trace on X
'X       ' NOW TRACED.
 11.23.26?
        LOOP
11.26.55>    J=J+1;
11.27.05>    IF J < 5 THEN X=X+2;
11.27.23>            ELSE X=X-1;
11.27.38>    END
11.27.44>    PRINTLN '***':2*J;
11.28.07>    EXITIF J=6;
11.28.36>END
'J       ' =       2
'X       ' =  2.0000000000000E+000
 ***
'J       ' =       3
'X       ' =  4.0000000000000E+000
   ***
'J       ' =       4
'X       ' =  6.0000000000000E+000
     ***
'J       ' =       5
'X       ' =  5.0000000000000E+000
       ***
'J       ' =       6
'X       ' =  4.0000000000000E+000
         ***
 11.28.48?
        CLEAR TRACE J                    - Clear Trace on J
'J       ' NO LONGER TRACED.
 11.29.57?X=45.5;  J=99;
'X       ' =  4.5500000000000E+001
```

- TRACE output is printed each
  time the value of the variable
  is changed.

- Change values being traced
- Trace output

33

ASK response, prompt


The ASK statement allows the user to interrupt program processing and accept input from the terminal. The ASK statement has two parameters. The first is the name of the variable which receives the users input. It can be any simple variable type (STRING,BOOL,REAL,INT,KEY). The second parameter is an expression which is typed to the user as a "prompt" for input. This expression may be an actual string expression enclosed in quotes or a string variable or function which has been previously defined. ISIS does not supply a separator between the printed output (prompt) and the users typed input (response); therefore, the user should supply his own separation character(s) within the prompt definition if he desires to be able to discriminate between the prompt and his response. If, in typing the input, (response) to ASK, the user makes a typographical error, he may hit the BREAK key in which case ISIS will disregard what has been typed, indicate this action by typing 3 dots (...) and then reprompt the user for correct input.

**EXAMPLE:**

```
14.31.19?VAR SANS: STRING
14.31.33?VAR IANS:INT
14.31.44?VAR RANS:REAL
14.31.52?VAR BANS:BOOL
14.32.03?VAR PROMPT: STRING
14.32.14?
        PROMPT='INPUT='          - Define users PROMPT variable.
14.32.30?
        ASK SANS,PROMPT
INPUT=THIS IS A STRING INPUT!
14.32.59?
        ASK IANS,PROMPT
INPUT=1254
14.33.25?
        ASK RANS,'INPUT REAL NO.'     - Interrupt program, PRINT prompt and
INPUT REAL NO.1.456732E+2               wait for user to type response.
14.34.15?                                (STRING, INT, REAL & BOOL)
      ASK BANS,'INPUT BOOL VALUE'
INPUT BOOL VALUETRUE
14.34.57?
        PRINT SANS,'   ',IANS,'   ',RANS,'   ',BANS
THIS IS A STRING INPUT!      1254      1.4567320000000E+002        TRUE

14.38.05?ASK SANS,'INPUT ?'
INPUT ? NOTICE BLANK AFTER THE '?'   - A blank is added by NOS system when you
                                       have odd  number of characters. This
                                       is due to the way the CYBER prints
                                       characters.
14.41.33?
        ASK SANS,'TYPE NOW -'
TYPE NOW -AM TYPING ERRRRR...        - User made error, then hit the BREAK
TYPE NOW -WAS TYPING ERROR             key. ISIS prints ... and reprompts
                                       user for input.
```

34

ASK continued

```
   12.42.56?VAR A : REAL; VAR Y : REAL; VAR Z : REAL;
   12.43.29?X=16;  Y=5.;  Z=2.;
VAR X          : REAL
   12.43.55?


           ASK SANS,'STRING INPUT'
VAR SANS          : STRING
STRING INPUTX+Y-3/Z
   12.44.50?PRINT SANS
X+Y-3/Z
   12.44.58?VAR RANS : REAL;
   12.47.18?
           ASK RANS,'REAL INPUT?'
REAL INPUT? X+Y-3/Z
   12.47.59?PRINT RANS
    1.95000000000E+001
   12.48.14?
           ASK RANS,'INPUT?'
INPUT?RANS+2
```

— Examples of expressions as input.

```
   08.27.13?VAR RAY:ARRAY[1..3] OF REAL
   08.28.54?VAR IRAY:ARRAY[1..2] OF INT
   08.29.19?VAR SRAY:ARRAY[1..4] OF STRING
   08.29.42?
           ASK IRAY[2],'INPUT 2ND ELEMENT *'
```

— Array elements can be used in ASK statement

```
INPUT 2ND ELEMENT *.635
```

└─ASK prompt    └─User response

```
   08.30.88?
           PRINTLN IRAY
[   1] =          0
[   2] =        635
```

PRINT      expression [:FORMAT1[:FORMAT2]]
PRINTLN    expression [:FORMAT1[:FORMAT2]]


The PRINT or PRINTLN statement evaluates each expression and prints
its value.  After printing the output, PRINT leaves the cursor at its current
position, whereas PRINTLN advances the cursor to the beginning of the next
line.  The optional FORMAT is similar to that of PASCAL in that each
expression may have its own format.  The format may specify total field
width, scaling factors or base conversions.  Default formats are as follows:

   REAL numbers - an E format type with field width of 22

   INT numbers      - a field width of 10 with all digits right
                      justified

   STRING - the field width is equal to the length of the complete
            string and string is left justified

   BOOL    - a field width of 10 and right justified

Variables that have been declared but not defined are assigned default values
by the ISIS system.  Integers and real variables are set equal to zero,
Booleans are set to FALSE and strings are of <u>zero</u> <u>length</u>.  Discussion of
optional formats is supplied below. All formats are ignored for ARRAY and RECORD outputs.

Format options — <u>:FORMAT1</u> - is the total field width.  The expression is
                 printed in the E format for real variables (PRINT  x:10,
                 y:15).  It can be used with integer, real, and string
                 variables.  Where FORMAT1 is smaller than the number of
                 characters in a string the field width is ignored and the
                 complete string is printed.  If FORMAT is greater than the
                 string length, blanks are added until the string length
                 specified by the format is printed.

                 <u>:FORMAT1:FORMAT2</u> — sets up total field width (FORMAT1)
                 (as described above) and defines the number of significant
                 digits to the right of the decimal point.  This means
                 the variable is printed in a fixed format.  <u>This format
                 applies only to real expressions.</u>


EXAMPLE:  (see next page)
    09.50.10?VAR Z:ARRAY[1..3] OF INT
    09.50.40?
            Z[2]=126
    09.51.07?PRINT Z                      — Print an array of numbers.
    [  1] =          0
    [  2] =        126
    [  3] =          0              36

**PRINT continued**

```
09.35.18?TYPE REC=RECORD NUM:INT; FLAG:BOOL; NAM:STRING; END;
09.36.18?VAR X:INT
09.36.32?VAR Y,R:REAL;
09.36.42?VAR S:STRING
09.36.52?VAR B:BOOL
09.37.01?VAR VREC:REC
09.37.10?
        PRINTLN S,'=S',X,'=X',B,'=B',R,'=R'
=S         0=X      FALSE=B                          0=R
09.38.12?
        PRINTLN VREC
NUM =             0
FLAG =        FALSE
NAM =

09.38.36?X=15;   Y=250.452;
09.39.16?PRINT SQRT(X*2-5), X/2*10-4
 5.00000000000000E+000   7.10000000000000E+001
09.39.53?PRINT SQRT(X*2-5):20:4
             5.0000
09.40.27?
        PRINT X,X:20
     15                   15
09.40.59?            PRINT Y
 2.50452000000000E+002
09.41.40?            PRINT Y:20
 2.50452000000E+002
09.41.55?            PRINT Y:20:10
    250.4520000000
09.42.18?            PRINT Y:20:5
        250.45200
```

- Uninitialized variables printed to show their default values
- Print undefined record variables

(Please note default strings are of zero length (NULL))

- Print expressions

- Print INT (with and without format)

- Print REALS (with and without format)

```
12.13.27?
         I=1
VAR I          : INT
12.14.14?S='CK UNTIL OUT'
12.15.05?REPEAT
12.15.39>         PRINT I
12.15.51>         PRINTLN SUB(S,I,1):2*I
12.16.18>         I=I+1
12.16.27>UNTIL I=10
     1 C
     2   K
     3
     4       U
     5         N
     6           T
     7             I
     8               L
     9
```

- Prints I (default format)
- Prints a subset (I) of length 1 from the string S with a format of (2 * I), which increases as the value of I increase

Declarative statements

CLEAR RUN


The CLEAR RUN statement is used to clear the contents of the RUN (input) file.


EXAMPLE:


```
10.29.55?SHOW RUN
42 LINES IN RUN.

10.30.49? CLEAR RUN
RUN CLEARED.

10.31.12?SHOW RUN
0 LINES IN RUN.
10.31.33?
```

Shown below are the operators and functions available to the user in programming desk top type calculations.

## RESULT TYPE —

### STRING:

Functions: CAT(X,Y...A): Concatenation of as many strings as you like.

SUB(X,Y,Z): Substring of X (string) starting at character number Y (Integer) for Z (Integer) number of characters. Y and Z can be variables, constants, or expressions.

### INT

Operators: +,-,DIV,MOD

Functions: ABS(x):     Absolute value of <u>integer</u> x

LEN(x):     Length of <u>string</u> x

ORD(x):     Ordinal number of the first character of the <u>string</u> x (ORD('C') = 3) [Implementation dependent]

ROUND(x):    <u>Rounded</u> value of <u>real</u> x

SQR(x):     Square of <u>integer</u> x

TRUNC(x):    Truncated value of <u>real</u> x

LOC(S1,S2):  Is S2 a substring of S1?

      If S2 is an undefined string variable (zero length string) then  LOC(S1,S2) = -1

      If S2 is not a substring of S1 then LOC(S1,S2) = 0

      If S2 is a substring of S1 then LOC(S1,S2) = the character position (index) within S1 at which the 1st occurrence of S2 begins

### REAL

Operators: +,-,/,*

Functions: ABS(x):     Absolute value of <u>real</u> x

ARCTAN(x):  Arc-tangent of x radians

COS(x):     Cosine of x radians

EXP(x):     e  raised to the power x

LN(x):      Natural logarithm of x

SIN(x):     Sine of x radians

SQR(x):     Square of <u>real</u> x

SQRT(x):    Square root of <u>real</u> or <u>integer</u> x

### BOOL

Operators: <,>,<=,>=,<>,=,AND,OR,NOT

Functions: ODD(x):  x  must be integer.  The result is BOOL

      If  x  is odd then ODD(x) = TRUE

      IF  x  is even then ODD(x) = FALSE

### KEY

Operators: +,-

# Library Statements

The ISIS library is a 5-level hierarchical file structure.  The library is where the user save, accesses, and purges pages of information.  These pages of information might be programs, data, control cards, or combinations of these.  Each page of information is assigned a pagename and is written in the form:

             library.shelf.book.chapter.page

These levels allow the user to easily describe and identify the information contained on a page.  Each level of the page name is separated by a dot (.).  Library pages are transferred by page name to frames for editing.  The page name will remain associated with a frame until the user changes the name (SET NAME) or transfers another page into the frame.  Further discussion on the association between the library page and the frame is discussed on page 49.  All five levels of the page name must be specified if the frame being used for editing has not previously been assigned a page name.  Otherwise, the page name may be specified by typing only those levels of the name that change.  However, when the user changes one level of a page name, then all <u>lower</u> levels, if not being changed, must be replaced by the dot separator and a blank space.  For example, if the page name is ISIS.CKCASES.SOURCE.PG1.SUB and the user wishes to change it to ISIS.CKCASE.BINARY.PG1.SUB then he may abbreviate as follows:  BINARY. . (all lower levels must be specified or abbreviated with dot and blank).  For the users convenience, default names have been provided for the first four levels of all new frames.  The default names are of the forms

             ISISLIB.S.B.C.(user supplied page level)

Where the user only has to assign the page level part of the name.  Before trying to store anything in a library, the library file must be created on the host computer.  This is done using one of the interface programs, ISISGEN.

The interface between the ISIS library environment and the NOS operating system is handled by 3 utility programs, ISISGEN, ISISPUT, and ISISGET.  ISISGEN sets up an NOS file in the format required by ISIS for its library system.  ISISPUT handles the transfer of information from the NOS system to the ISIS library.  ISISGET handles the transfer of ISIS library information to the NOS file system.  Detailed writeups of these utility programs appear on the next page.

NOTE:  The user should try and keep the number of shelves and books less then 40 in order to allow ISIS to run fast.  Also, each level of the page name is limited to seven characters.

LIBRARY STATEMENTS

ISIS/NOS Interface

## ISISGEN

ISISGEN is a program which allows the user to create a library for use by
ISIS. A library must be created before any pages may be stored. To do this
an NOS direct access file must first be created. This file is where the users
library is saved on the NOS system. Then ISISGEN will convert the file to the
format required by the ISIS library. The control cards necessary to do this
are shown below.

```
ATTACH,ISISGEN/UN=961300N.
DEFINE,LIBRARY=isislib/M=W.
ISISGEN.
RETURN,LIBRARY,ISISGEN.
```

## ISISPUT

ISISPUT is a program which allows the user to take an NOS local
file (Nfn) and put it into the ISIS library system. The NOS file is rewound
before it is stored. The full ISIS page name must be typed. This means that
any NOS file can now be edited by ISIS. The control cards for using ISISPUT
are as follows:

```
ATTACH,ISISPUT/UN=961300N.
ISISPUT,Nfn.   library.shelf.book.chapter.page  (see writeup on page 40)
```

NOTE:   ISISPUT uses the smallest possible increment (.001) for the line number
        assignment. This means the user must REKEY the page before he may make
        any editing INSERTS.

## ISISGET

ISISGET is a program which allows the user to get an ISIS library page and
write it into an NOS local file. The NOS file is NOT rewound
after it is retrieved. The full ISIS page name must be typed. Control cards
necessary to use ISISGET are shown below.

```
ATTACH,ISISGET/UN=961300N.
ISISGET,Nfn.   library.shelf.book.chapter.page  (see writeup on page 40)
```

[frame-id/] SET NAME [library].[shelf].[book].[chapter].[page]


The SET NAME statement assigns a library page name to the contents of the ACTIVE frame or specified frame.  This allows the user to assign a new name or change the library page name associated with that frame.  The page name contains 5 levels for identification purposes.  If the frame has not been previously assigned a page name (SET NAME or USE), then each level of the name must be specified (SET NAME library. shelf . book . chapter . page).  Otherwise, the pagename  may be abbreviated as discussed in the library statement writeup on the previous page.  When the contents of a frame are saved, they will be saved under the library name assigned to it with this statement.

When the user changes the pagename associated with a frame and wishes to avoid retyping lower level parts of the name which do not change, he must type the DOT separator followed by a blank to replace the next lower level name.  See 2nd, 3rd, and 5th examples.


EXAMPLE:


```
10.04.22?SET NAME ALIB.MUST.ISIS.EDITOR.SPECS
ALIB.MUST.ISIS.EDITOR.SPECS   IS THE NAME OF WORK
```
- **The initial SET NAME statement must include all levels of names (NO DEFAULT)**

```
10.04.50?SET NAME DATBASE.
ALIB.MUST.ISIS.DATBASE.SPECS   IS THE NAME OF WORK
```
- **Rename ACTIVE frame changing the 2nd level (chapter) of name only**

```
10.05.25?SET NAME MISC.NEWPOOP. .
ALIB.MISC.NEWPOOP.DATBASE.SPECS   IS THE NAME OF WORK
```
- **Rename ACTIVE frame changing the SHELF and BOOK level of name**

```
10.06.02?SET NAME WILL
ALIB.MISC.NEWPOOP.DATBASE.WILL   IS THE NAME OF WORK
```
- **Rename ACTIVE frame changing only lower level of name**

```
10.06.35?SET NAME COPY. . .
ALIB.COPY.NEWPOOP.DATBASE.WILL   IS THE NAME OF WORK
```
- **Rename ACTIVE frame changing SHELF level of name**

```
14.42.03?FRAME W1,W2:STRING;
```
- Declare frames

```
14.42.56?W1/SET NAME ALIB.NEWVER.CKCASES.IMPL.MOVE
```
- Assign pagename to W1 frame

```
ALIB.NEWVER.CKCASES.IMPL.MOVE   IS THE NAME OF W1
14.44.23?
          W1/SET NAME EXPL.
```
- Rename W1 frame changing chapter level of name.

```
ALIB.NEWVER.CKCASES.EXPL.MOVE   IS THE NAME OF W1
```

[frame-id/]    USE   [library].[shelf].[book].[chapter].[page]


The USE statement is used _to_ read the contents of a specified page into the
ACTIVE or specified frame.  The library page name associated with that frame
becomes what was specified in the USE statement and the contents become what
was extracted from the page in the library.  To specify a frame other than the
ACTIVE frame the user must precede the statement with the frame-id and a slash.


**EXAMPLE:**

```
14.12.56?

        SHOW NAME                                          - ACTIVE frame name
ALIB.NEWVER.CKCASES.TEST.PROGRAM   IS NAME OF WORK

14.15.17?USE TEST.                                         - Put new page into
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS WORK                  ACTIVE frame



14.35.52?FRAME A1,A2: STRING;                              - Declare frame-ids


14.38.59?A1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM          - Put page into A1
                                                              working frame
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS A1
14.39.42?
        A1/USE SHOWS                                      - Put new page into
                                                              A1 frame
ALIB.NEWVER.CKCASES.TEST.SHOWS USED AS A1
```

[frame-id/]  SAVE  [*]


The SAVE statement is used to create a new page in the library or to
replace an already existing page.  SAVE places the contents of the ACTIVE frame
into the library under the same library name now associated with the ACTIVE
frame.  The SAVE statement followed by an  *  will replace a page in the
library of the same name as the ACTIVE frame.  At the present time, the user
may replace a page (SAVE*) even if the page does not already exist.  The user
should be cautious in saving or replacing a page to make sure the library page
name is correct.  The page name may be changed using the SET NAME statement
before saving or replacing it.


EXAMPLE:


```
 16.25.30?SET NAME ALIB.SHELF.BK.CH.CG
ALIB.SHELF.BK.CH.CG  IS NAME OF WORK .
 16.26.10?SAVE
ALIB.SHELF.BK.CH.CG SAVED.
 16.26.27?SET NAME XXX
ALIB.SHELF.BK.CH.XXX  IS NAME OF WORK .
 16.26.52?SAVE
ALIB.SHELF.BK.CH.XXX SAVED.
```

                – Save the ACTIVE frame by storing
                   it in the library under this name.

                – Save the contents of the ACTIVE
                 frame under this page name in the
                 library


```
 16.27.17?SAVE*
ALIB.SHELF.BK.CH.XXX SAVED.
```

                – Replace the contents on the
                 already existing page of the
                 library

```
 14.46.04?FRAME W1:STRING;
```
                   – Declare frame

```
 14.46.17?W1/USE ALIB.NEWVER.CKCASES.EXPL.COPY
```
– Put a page into W1 frame

```
ALIB.NEWVER.CKCASES.EXPL.COPY USED AS W1
 14.48.21?
         W1/SET NAME EXPIMP.
```
                  – Reset the page name of W1 frame

```
ALIB.NEWVER.CKCASES.EXPIMP.COPY  IS THE NAME OF W1
 14.49.35?
         W1/SAVE
```
                – SAVE contents of W1 frame

```
ALIB.NEWVER.CKCASES.EXPIMP.COPY SAVED.
```

Library Statements

[frame-id/]  PURGE  [library].[shelf].[book].[chapter].[page]


The specified page is eliminated from the library.  If the specified page is the only page in its chapter, the chapter is eliminated from the library. If this chapter is the only chapter in its book, then that book is eliminated from the library.  Finally, if the book is the only book in its shelf, then that shelf is eliminated from the library.  If the page name is incompletely specified, its library, shelf, book, chapter, page parts will be taken from the name currently associated with the ACTIVE frame (or the specified frame).


**EXAMPLE:**

```
11.21.18?SET NAME ALIB.GRANT.BOOK.CHOP.PG2         — Set the ACTIVE frame name
ALIB.GRANT.BOOK.CHOP.PG2  IS NAME OF WORK
11.22.32?
        SHOW PAGES ALIB. . . .                      — Display the library ALIB
ALIB   .GRANT  .BOOK    .CHOP   .PGALL
        .        .       .       .CCA
        .        .       .       .PG1
        .        .       .       .PG2
        .SHELF  .BK     .CH     .CG
        .        .       .       .XXX
        .        .BOOK   .CHAP   .PAGE
11.24.24?
        PURGE  PG2                                   — Purge   PG2
ALIB.GRANT.BOOK.CHOP.PG2 PURGED.
11.25.33?


        PURGE SHELF. .CHAP.PAGE                      — Purge PAGE (resides on
ALIB.SHELF.BOOK.CHAP.PAGE PURGED.                      different shelf and chapter)
11.25.06?


        SHOW PAGES ALIB. . . .                       — Display the library to show
ALIB   .GRANT  .BOOK    .CHOP   .PGALL                 there are 2 less pages in
        .        .       .       .CCA                  it now
        .        .       .       .PG1
        .SHELF  .BK     .CH     .CG
        .        .       .       .XXX

14.46.04?FRAME W1:STRING:                            — Declare frame

14.46.17?W1/USE ALIB.NEWVER.CKCASES.EXPIMP.COPY     — Put a page into W1 frame

ALIB.NEWVER.CKCASES.EXPIMP.COPY USED AS W1
14.53.12?
        W1/PURGE EXPIMP.                            — Purge a page in W1 frame

ALIB.NEWVER.CKCASES.EXPIMP.COPY PURGED.
```

[frame-id/]    VOID


This command disposes of the contents of the ACTIVE frame or the specified frame but it retains the library page name assigned to the frame.  To void other frames the user must precede the command with the frame name and a slash.


EXAMPLE:


```
08.52.41?USE ALIB.NEWVER.CKCASES.TEST.PROGRM        - The page to be edited is read
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK          from the data base library.
08.53.18?


08.53.37?LIST .1/.3
   0.1    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);         - List first few lines of page.
   0.2    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   0.3    =
08.54.03?


08.54.13?VOID                                        - VOID the active frame
08.55.25?                                              contents.


        LIST ALL
NO ITEMS IN SPECIFIED RANGE.                         - Contents have been voided.
08.55.37?


        SHOW NAME
ALIB.NEWVER.CKCASES.TEST.PROGRM  IS NAME OF WORK     - Name remains in tact.
08.55.47?


14.56.11?FRAME A1,A2:STRING;                         - Declare frames

14.56.59?A1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM     - Put a page into A1 frame

ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS A1
14.57.34?A1/VOID                                     - VOID A1 frame contents

14.57.59?A1/SHOW NAME                                - A1 frame still retains
                                                       the pagename
ALIB.NEWVER.CKCASES.TEST.PROGRAM  IS NAME OF A1
```

Library Statements


STORE    library.shelf.book.chapter.page


    The STORE statement allows the user to save the current environment on a specified page in the library.  The current environment includes TYPES, VARS, ABBREVS and FRAMES.


EXAMPLE:

```
09.51.00?TYPE DIRS: REAL;
09.51.20?VAR PI,X,Y,Z: REAL;
09.51.48?ABBREV P: PRINTLN;
09.52.02?ABBREV I: INSERT;
09.52.14?FRAME F1,F2: STRING;
09.52.33?X=15;  Y=20;  PI=3.14;
09.52.54?
```

      } - Current environment.

```
        STORE ALIB.TEST.PROGRAM.ENVIR.AUG3
ALIB.TEST.PROGRAM.ENVIR.AUG3 SAVED.
```
  - Store the above environment in a library page.

RESTORE     library.shelf.book.chapter.page


        The RESTORE statement retrieves an environment previously stored
in a page.  This means that all current VARS, TYPES, ABBREVS, and
FRAMES will be replaced with the environment previously stored on a
library page.  If RESTORE is contained within a group of statements
being executed, the statements following the RESTORE are not executed.
At present, values of VARS and contents of FRAMES are not being restored.


EXAMPLE:
```
09.51.02?TYPE DIRS: REAL;
09.51.20?VAR PI,X,Y,Z: REAL;
09.51.48?ABBREV P: PRINTLN;
09.52.02?ABBREV I: INSERT;
09.52.14?FRAME F1,F2: STRING;
09.52.33?X=15;  Y=20;  PI=3.14;
09.52.54?
        STORE ALIB.TEST.PROGRAM.ENVIR.AUG3
ALIB.TEST.PROGRAM.ENVIR.AUG3 SAVED.
09.53.44?
        VAR A,B,C: REAL;
09.54.04?ABBREV D:DELETE;
09.54.24?FRAME XXX: STRING;
09.54.37?  X=5.1;   Y=4.;
09.54.56?
        SHOW ABBREVS
'D        ': DELETE
'I        ': INSERT
'P        ': PRINTLN
09.55.13?SHOW VARS
'A        ': REAL;
'B        ': REAL;
'C        ': REAL;
'PI       ': REAL;
'X        ': REAL;
'Y        ': REAL;
'Z        ': REAL;
09.55.26?RESTORE ALIB.TEST.PROGRAM.ENVIR.AUG3
CONTEXT RESTORED: ALIB.TEST.PROGRAM.ENVIR.AUG3
09.56.05?
        SHOW ABBREVS; SHOW VARS; P X,'=X',Y,'Y=';
'I        ': INSERT
'P        ': PRINTLN
'PI       ': REAL;
'X        ': REAL;
'Y        ': REAL;
'Z        ': REAL;
  1.5000000000000E+001=X   2.0000000000000E+001Y=
```

- This environment was
  stored on a library
  page.

- This is current
  environment.

- Restore the environment
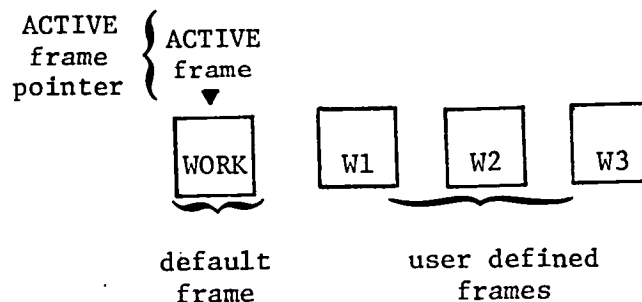  to the one saved on this
  page.

- These statements
  show the saved
  environment has
  been restored.

TEXT EDITING STATEMENT

Frame Concept

A working frame is used for temporary storage during editing. It may be a copy of a library page which is being modified or may be entered entirely by the user. There are 10 working frames available to the user. A default frame named WORK is provided and 9 other frames which must be named by the user before they can be used. The frames are named using the declarative statement, FRAME (FRAME W1,W2,W3:STRING;). The user can select a page from the library, put a copy in a working frame where the code may then be modified using any of the Edit Statement verbs. The working frames are also used for temporary storage of code for examination or for use with read only statement verbs such as LIST, RUN, EXEC, and COUNT. Any of these working pages may at any time become the current ACTIVE frame by designating a particular frame using the statement verb, ACTIVE frame. The ACTIVE frame does not require the frame name as a prefix to the statement verb. In other words, the statement ACTIVE acts like a pointer. If a frame name is not specified in the command, then editing automatically takes place in the ACTIVE page. The default ACTIVE frame is the same as the default working frame, WORK. The example below may help explain the frame concept better.

FRAME W1,W2,W3:STRING;  — user declares working frames

```
ACTIVE    ⎧ ACTIVE
frame     ⎨ frame
pointer   ⎩   ▼
         ┌──────┐   ┌──────┐   ┌──────┐   ┌──────┐
         │ WORK │   │  W1  │   │  W2  │   │  W3  │
         └──────┘   └──────┘   └──────┘   └──────┘
          default         user defined
          frame              frames
```
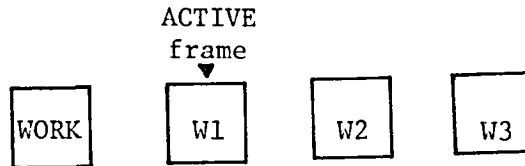
LIST      - list the ACTIVE frame which is WORK

W2/LIST - list the W2 frame (W2 is not ACTIVE and must prefix
                                the LIST statement)

ACTIVE W1 - the W1 frame is declared to be the new ACTIVE frame

```
                    ACTIVE
                    frame
                      ▼
    ┌─────┐     ┌─────┐     ┌─────┐     ┌─────┐
    │WORK │     │ W1  │     │ W2  │     │ W3  │
    └─────┘     └─────┘     └─────┘     └─────┘
```

so now

LIST        - list ACTIVE frame which is now W1 - W1 was declared
                the ACTIVE frame in the above statement.
WORK/LIST - list the default frame WORK (inactive now) by preceding
                command with the frame name.


Since the editor must retrieve pages from the library to be modified and
store information in the library, an interface between the editor and library
is required.  Associated with each editor frame is the name of a library page
into which that frame will be SAVEd or from which library information will
be retrieved by a USE command.  This page name is specified and changed by
the SET NAME or USE statements.  The specification of this page name via
SET NAME or USE follows the format and abbreviation procedure discussed in
the library statement writeup on page  40.

TEXT EDITING

RANGE Concept

The ISIS text editor differs from other editing systems available on the
CDC Cyber system such as the CDC text editor (EDIT) and the XEDIT system.
These systems are pointer oriented, in that the text to be edited is accessed
by sliding a pointer up and down the page. Any line that the pointer is
pointing to is the line which is operated on by the editing command. In
contrast, the ISIS text editor is line oriented. It does not reference text
with respect to the position of a pointer. Instead, the text is referenced
in an "absolute" sense by designating, within each command, the text to be
modified. This means that each line of text must be capable of being uniquely
identified. The ISIS editor does this by assigning a number to each line of
text. The line numbers run in ascending order and represent reference
points for the specification of the text be accessed by the editing commands.
Those lines affected by a particular edit command are referred to as the RANGE
of that command.

The use of explicit line numbers, as in the context "from line 1 to
line 5," permit the user to operate on everything within a certain area of
the page. This range specification is referred to as an explicit range.
A second means of designating text to be operated on consists of specifying
a search for all lines having a particular characteristic , such as all
those containing the string "ABC." This is the familiar search-for-string
facility available in most text editors, and will be referred to as an
implicit range. Explicit and implicit qualifiers can be combined, as
in "all lines containing 'ABC' between line 1 and line 5."

The explicit range, as mentioned earlier allows the user to look
at a certain area of a page, which can be as small as

one line or as large as the complete page.  Figure 1 shows all of the possibilities for the explicit range.  The explicit range can consist of one line number ($\ell$n).  When the range consists of more than one line, the first line number and the last line number to be considered are separated by a slash ($\ell n_1 / \ell n_2$).  It is also possible to put a limit on the total number of lines to be considered within this particular area. In this case, the total number of lines allowed would appear in parentheses and follow the line numbers specifying the particular area ($\ell n_1 / \ell n_2 (n\ell)$). The first n$\ell$ lines will be operated on.  When the range includes the whole page the range would be specified by the word ALL.  Also available are multiple ranges separated by commas.  This means that more than one range may appear in an edit statement and they do not have to be in any order.  Line numbers used in the explicit range specifications are referred to as KEYS and must be KEY type variables.  Hence, line numbers ($\ell$n) can be any type of KEY operand, a simple number, a KEY variable or a KEY expression.  Operators in KEY expressions are addition (+) and subtraction (−) only.  The limit put on the number of lines (n$\ell$) must be an INT operand such as a simple number, an INT variable, an INT expression, or an INT function. Following are  some examples of explicit ranges used with the different edit statements:

```
LIST      KV2
INSERT    KV2=.5/KV3/.01(IV3-2)
REPLACE   2.9,SV1/10,9/KV3-.8(IV2)
CHANGE    'CH' to CAT (SV5,'1') IN KV2/KV3-1(2)
```

The implicit range allows the user to look at all lines containing a particular characteristic. The characteristic is made up of a string of characters and must be a STRING operand.  The STRING (string-id) can be a simple string enclosed in quotes, a STRING

variable or a STRING function.  The string of characters may be specified
to begin in a specific column (col) or begin  some where between two
columns ($col_1$,$col_2$).  This part of the range will be enclosed in parenthesis
and follow the string characteristic ('DECLARE'(2,8)).  The column number
(col) must be an INT type operand, a simple number, a variable, an expression,
or an INT function.  It is also possible to look at all lines <u>NOT</u>
containing a specific characteristic.  This is accomplished by placing a
NOT in the range in front of the string of characters (NOT 'DECLARE'(4)).
Lines containing two or more characteristics may be specified as follows:
The first string characteristic is followed by an AND or OR which in turn
is followed by the second characteristic ('DECLARE'(2,8) AND 'PARTI').
Following are some examples of the implicit ranges used with some of the
edit statements:

```
LIST   'CH'
REPLACE SV2(34,IV3)
CHANGE SV1 to 'XX'  IN  'CH' (29,36):M
ADD  '*'  AT SV1  IN '777B'
```

The implicit and explicit ranges can be used together.  This allows the
common character string search to be restricted to a particular area
of the page.  This implicit part of the range  is     stated first followed
by the word  IN  which is followed by the explicit range as shown
figure 1.  Examples of this range combination are shown below:

```
LIST  SV1 AND NOT ('IN')  IN 15.4/20
REPLACE  'CHQUO'  IN  SV1/SV2+5(4)
CHANGE  'MAX'  to 'MIN'  IN 'BITS'  IN 3/4
ADD  '#'   AT 20  IN 'LB'  IN .5, KV2-KV1
```

53

The INSERT command, since it does not deal with existing text, differs radically from the other edit commands in its permissible range specifications. Obviously, the implicit range (string search) has no meaning since the text does not exist. For the same reason the explicit range, ALL,.F,.L, and .C, are not applicable to INSERT either. The explicit range for the INSERT statement differs slightly by allowing the user to specify an incremental option (inc) separated from the line numbers by a slash ($\ell n_1/\ell n_2/inc$). This is the increment between line numbers that ISIS uses when assigning line numbers to the text being inserted. The incremental value (inc) must also be a KEY operand, but may also include an integer function.
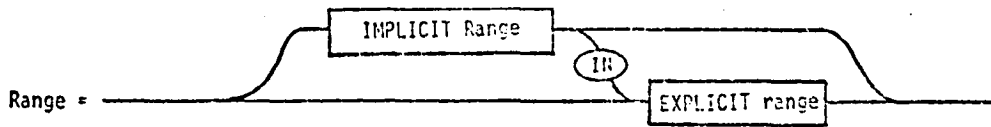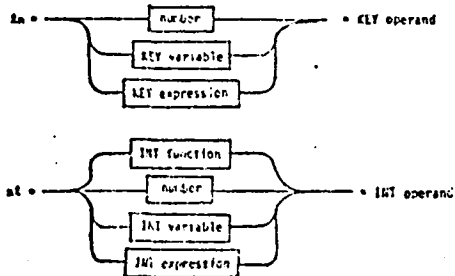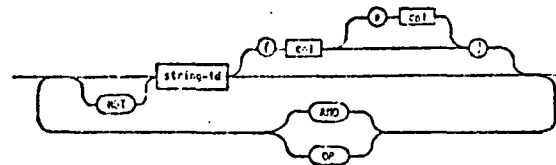
| EXPLICIT Range Syntax | IMPLICIT Range Syntax |
|---|---|
|  |  |
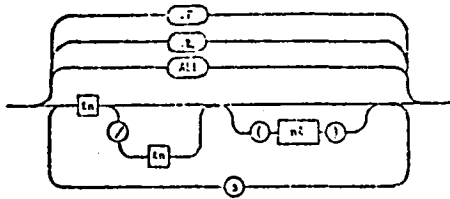
Range Options:

| | |
|---|---|
| ℓn | – edit line number ℓn |
| ℓn (nℓ) | – beginning with line number ℓn, edit nℓ number of lines |
| $ℓn_1/ℓn_2$ | – edit all lines between and including $ℓn_1$ & $ℓn_2$ |
| $ℓn_1/ℓn_2(nℓ)$ | – edit lines between and including $ℓn_1$ & $ℓn_2$ without exceeding nℓ number of lines |
| $ℓn/ℓn_2,ℓn_3,ℓn_4$ | – edit multiple ranges. They do not have to be in any order. Any of the above ranges may be used. |
| All | – edit all lines in the frame. |
| .F | – edit the first line in the frame. |
| .L | – edit the last line in the frame. |
| .C | – edit the current line in the frame. |

| | |
|---|---|
| string-id | – edit all lines containing this string characteristic |
| string-id (col) | – edit all lines containing this string in the column number col |
| string-id $(col_1,col_2)$ | – edit all lines containing this string and appearing between the columns, $col_1$ and $col_2$ |
| NOT string-id | – edit all lines not containing the string-id where it may be any of the above options |
| string-id {AND OR} string-id | – edit all lines containing string-id {AND OR} string-id where the string-ids may be any of the above options |
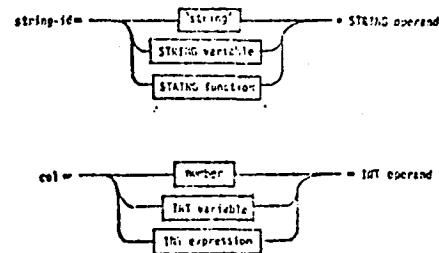
Figure 1.– Range concept for text editing commands.

FRAME    frame-id(s)   :    STRING

The FRAME statement allows the user to declare the working frames.  There
is one default frame-id, WORK, but the remaining available frames (9) must be
declared by the user with this statement before they can be used.

EXAMPLE:

```
FRAME W1,W2: STRING;
  14.21.59?
          W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1
```

ACTIVE    frame-id


This command allows the user to activate any working frame.  This means that the statement verbs referring to a particular frame will not have to be preceded by the frame-id now.


EXAMPLE:

```
FRAME W1,W2: STRING;
  14.21.59?
          W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1
  14.23.57?
          ACTIVE W1
  14.27.63?
          SHOW NAME
ALIB.NEWVER.CKCASES.TEST.PROGRAM  IS NAME OF W1
```

ERASE  {abbrev-id(s)|type-id(s)|var-id(s)|frame-id(s)}

 

The ERASE statement removes the specified types, variable-ids or frame name from the identifier tables.  More than one id may be erased at one time with ids separated by a comma.  Caution should be exercised when using ERASE. Erasure of the ACTIVE frame is not allowed.

EXAMPLE:

```
13.34.00?FRAME F1,F2,X1,X2:STRING      - Declare frames
13.34.28?
         SHOW FRAMES                   - Show frames
'F1          ': WORK4      : STRING;
'F2          ': WORK3      : STRING;
'X1          ': WORK2      : STRING;
'X2          ': WORK1      : STRING;
13.34.41?
         ERASE F1,X1                   - Erase frames, F1,X1
13.35.24?SHOW FRAMES                   - Frames F1,X1 now erased
'F2          ': WORK3      : STRING;
'X2          ': WORK1      : STRING;
13.35.37?
         ACTIVE X2                     - Make X2 the ACTIVE frame
13.36.32?
         SHOW FRAMES
'F2          ': WORK3      : STRING;
'X2          ': WORK1      : STRING;
13.36.43?
         ERASE X2                      - User cannot erase the ACTIVE frame
XXXXXXXX         ↑ MAY NOT BE ERASED.
```

[frame-id/]    LIST    [range] $\left[ :[ \{ NI | NK \} ], [V], [T] \right]$

The LIST statement is used to view all or part of the ACTIVE frame  or specified frame.  The range is optional and if not specified, the entire page is listed.  The range may consist of the implicit and/or explicit range discussed on page 55.  In addition to this range are two range options: .F  which lists the first line of the frame and  .L  which lists the last line of the frame.  The display option follows the range specification and is described below:

Display Option:    NI (NO ITEM)        - do not display the contents of the line, only the line number.

NK (NO KEY)        - do not display the line number (key) only the contents of the line.

V (VETO)           - user may VETO or Verify Listing by responding to 'OK?' with:
                     Y - YES        continue listing
                     N - NO         do not list
                     K - KILL       terminate listing and abort command

T (TAG)            - Display tag-ids

EXAMPLES:    IMPLiCIT

```
13.45.34?USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
13.46.24?
          VAR SV1,SV2:STRING;
13.46.53?VAR IV1,IV2:INT;
13.47.16?
          SV1='CH';   SV2='LEN';
13.48.00?  IV1=14;
13.48.23?
          LIST CAT(SV1,'NOT')
  1.2   =    CHNOT = '#';
13.49.09?

          LIST SV2(6)
  1.5   =    IDLEN = 10;
  1.6   =    NMLEN = 7;
  1.7   =    INLEN = 82;
  6.    =    VNLEN = 7;
13.49.43?

          LIST '777'(IV1,IV1+2)
  2.3   =    BLKMAX = 3777B;
  3.6   =    BVDMAX = 777777B;
  5.3   =    LRUMAX = 777777B;
13.50.25?

          LIST SV1 AND NOT 'IN'
  0.7   =    CHBLANK = ' ';
  0.8   =    CHQUOTE = '''';
  0.9   =    CHZERO = '0';
  1.    =    CHEOL = '$';
  1.1   =    CHSEMI = ';';
 ·1.2   =    CHNOT = '#';
```

- The page to be edited is read from the data base library into the ACTIVE frame.

- List all lines containing the concatination of 'CH' (SV1) and 'NOT' - CHNOT -

- List all lines containing 'LEN' (SV2) in column number 6.

- List all lines containing '777' in columns 14(IV1) thru 16 (IV1+2).

- List all lines containing 'CH' (SV1) and not the string 'IN'.

59

```
LIST              * : EXPLICIT

  09.31.46?USE ALIB.NEWVER.CKCASES.TEST.PROGRM      - Read the page to be edited
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK          from the data base library
  09.32.37?                                           into the ACTIVE frame.
          VAR KV1,KV2,KV3,KV4:KEY;
  09.33.17?VAR IV2: INT
  09.33.29?VAR IV3:INT
  09.33.49?     IV2=2;   IV3=5;
  09.34.06?     KV1=20;  KV2=.2;  KV3=2;   KV4=6;
  09.34.45?
          LIST KV2                                  - List line .2 (KV2).
  0.2   =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
  09.34.58?


          LIST KV2+1(3)                             - List lines starting at 1.2
  1.2   = ·  CHNOT = '#';                              (KV2+1) with a limit of 3
  1.3   =    NDBLANKS = '                              lines.
  1.4   =
  09.35.30?


          LIST .9/KV3-.8;                           - List lines beginning with
  0.9   =    CHZERO = '0';                            .9 through 1.2 (KV3-.8)
  1.    =    CHEOL = '$';
  1.1   =    CHSEMI = ';';
  1.2   =    CHNOT = '#';
  09.35.54?


          LIST 3.5/KV1-15(SQR(IV2))                 - List lines 3.5 through 5.
  3.5   =    BVDLEN = 250;                             (KV1-15) with a limit of
  3.6   =    BVDMAX = 777777B;                         4 (KV2**2) total lines.
  3.7   =    BVDSYN = -1;
  3.8   =    BVDFIRST = 2;
  09.36.28?


          LIST KV2, 5.1/KV4-KV2(IV2), KV3+.1/KV3+.2; - List 3 separate
  0.2   =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);            ranges.
  5.1   =
  5.2   =    STATUSBLK = 1;                            1) line .2 (KV2)
  2.1   =    BLKSZ = 256;                              2) line 5.1 through
  2.2   =    BLKLEN = 250;                                5.8 with a limit
  09.37.33?                                                of 2 lines
                                                       3) line 2.1 (KV3+.1)
                                                          through line 2.2
                                                          (KV3+.2).


  09.16.27?FRAME W1:STRING;                         - ·Declare frame
  09.16.44?
  09.16.54?W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM  - Put a page into the W1
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1            frame.
  09.18.04?
          W1/LIST .3/.5
  0.3   =                                           - LIST a few lines of the
  0.4   =CONST  (* GLOBAL CONSTANTS *)                 W1 frame.
  0.5   =    CHFIRST = ':';
```

60

Text Editing Statement

[frame-id/]   INSERT   range

The INSERT statement is used to add new lines to the ACTIVE frame or specified frame.  INSERT differs from the other edit statements because it adds new lines to a frame instead of operating on existing lines.  Since INSERT is not working with an already existing line, an implicit range for INSERT does not make sense. The explicit ranges, ALL, .F, .L, and .C are not applicable to this command either.

The explicit range differs slightly by allowing the user to specify an incremental option (inc) separated from the line numbers by a slash ($\ell n_1 / \ell n_2 / inc$). This is the increment ISIS uses when assigning line numbers to the inserted text. The incremental value (inc) must be a key operand (see below).  If inc is not specified in the range, then it will assume the current value of the system variable, SYSTEM.DELTA (see page 14).  The user is prompted for successive lines to fill the range.  Insertion may be halted by depressing the BREAK key.  A range must be specified and may be taken from the table below.

Range =
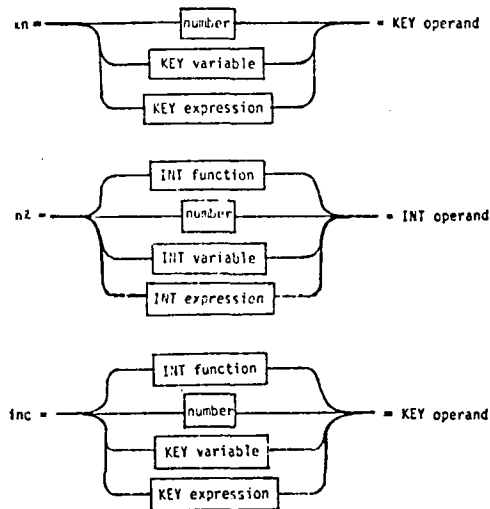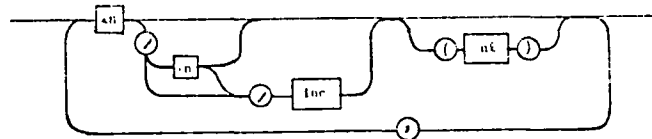




Range with inc:   $\ell n_1 / \ell n_2 / inc$   - insert lines between and including $\ell n_1$ and $\ell n_2$ incrementing the line number by inc.

$\ell n / / inc$   - insert lines beginning with $\ell n$, incrementing the line number by inc, until the user halts insertion by depressing the BREAK key.

NOTE:  The user should take care and not insert lines overlapping already existing line numbers.  Insertion must be made between two existing line numbers.

EXAMPLES:

```
  16.57.42?USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
  16.58.30?
            VAR KV1,KV2,KV3,KV4:KEY;
  16.58.54?VAR IV1,IV2,IV3,IV4:INT;
  16.59.11?
            KV1=1.41;  KV2=7.;  KV3=.001;
  16.59.40?IV1=1;    IV2=3;   IV3=5;
  17.00.59?
            INSERT .41
    6.41  =
  17.02.02?


            INSERT KV2-.5//KV3
    6.5  =
    6.501 =
    6.502 =
    6.503 =

INSERT TERMINATED.


  17.14.00?INSERT KV1/KV1+.05/.01(IV3-IV2)
    1.41 =
    1.42 =
  17.14.57?


            INSERT KV1+.05/1.469/KV3+KV3, KV2+1
    1.46  =
    1.462 =
    1.464 =
    1.466 =
    1.468 =
    8.    =
  17.23.44?


  14.56.11?FRAME A1,A2:STRING;


  15.00.07?A1/SET NAME ALIB.OLDVER.CKCASES.TEST.PROGRAM

ALIB.OLDVER.CKCASES.TEST.PROGRAM  IS THE NAME OF A1


  15.03.48?VAR KINC:KEY; KINC=.5;

  15.07.34?A1/INSERT KV2/8./KINC

    7.    =
    7.5   =
    8.    =
```

- The page to be edited is read from the data base library into the ACTIVE frame.

- Insert line number .41.

- INSERT beginning with line 6.5 and an increment of .001. Insertion may continue until user depresses the BREAK key which will then abort the command.

- Insert beginning with line 1.41 through line 1.46 with an increment of .01, but with a limit of 2 lines.

- Insert using different ranges.
  (1) Insert beginning with line 1.46 going through 1.469 with an increment of .002.
  (2) Insert a line at 8.

- Declare frames

- Set pagename of A1 frame

- Declare variables

- Insert information on the page residing in the A1 frame

Text Editing Statements

[frame-id]/READ  string-var $\{\Delta|,\}$  ℓn

     The READ statement is used to read a single line of code from the
ACTIVE or designated frame into a string variable.  A frame option will
allow the user to read a line from any of the working frames.  The
variable where the code is to be stored must be a predefined STRING-
variable.  The variable name is separated with a blank or a comma from
the line number (ℓn) where the code is read from.  The line number (ℓn)
is a KEY operand and may be simply the line number, a KEY variable, or a
KEY expression as shown below.

ℓn =

- Number
- KEY variable
- KEY expression
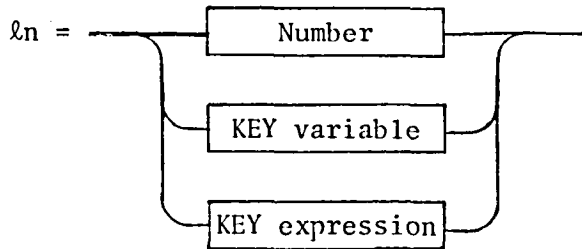
If the line number (ℓn) specified does not exist, the STRING variable is
set to a null string and no indication is given.

EXAMPLE:

```
14.20.33?USE HLIB.NENVER.CKCASES.TEST.PROGRAM          - The page to be edited
HLIB.NENVER.CKCASES.TEST.PROGRAM USED AS WORK            is read from the data
                                                         base library.

14.21.07?LIST .3/.6
   0.3   =
   0.4   =CONST  (* GLOBAL CONSTANTS *)
   0.5   =   CHFIRST = ';';
   0.6   =   CHLAST = ';';
14.21.24?
         VAR X:STRING; READ X,.5; PRINT X;           - Read the contents of line
   CHFIRST = ';';                                      .5 , store it in the
14.22.13?                                              string variable X and
                                                       PRINT it to check for
                                                       correctness.
```

READ   continued

```
09.29.20?FRAME W1:STRING;                          - Declare frame
09.29.26?
09.29.29?W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1        - Put a page into the W1
09.29.55?                                            frame.
        W1/LIST .3/.4                               - LIST lines
  0.3   =      .
  0.32  =THIS IS A COMMENT LINE
  0.4   =CONST   (* GLOBAL CONSTANTS *)
09.32.20?
        VAR X:STRING;
09.33.46?
        W1/READ X,.32                              - Read a line in W1 frame.
09.34.52?PRINT X;
THIS IS A COMMENT LINE
```

Text Editing Statements

[frame-id/] WRITE  string-id  {Δ|,}  ℓn

The WRITE statement is used to add a single line of code to the ACTIVE
working frame.  This command is a simple and quick version of the INSERT or REPLACE
statement (since there is no prompt).  A frame option will allow the user
to add code to any of the working frames.  The code to be inserted (string-id)
may be in the form of a simple string, a predefined variable or a string
function as shown in the diagram below.  The code is separated with a blank
or a comma from the line number (ℓn) where it will be inserted.  The line number
is a KEY operand and may be simply the line number, a KEY variable or a KEY
expression as shown below.  Please note that this command assumes that you
know what you are doing - for example, if the line already exists, then it will
be replaced without notifying the user.

ℓn =
- number
- KEY variable
- KEY expression

string-id =
- 'string'
- string variable
- string function
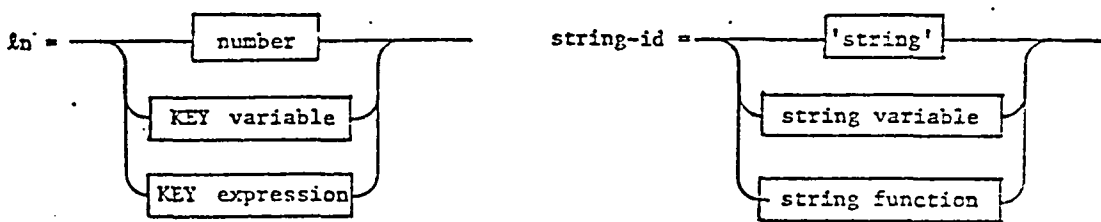
EXAMPLES:

```
 15.12.01?USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
 15.15.38?
          VAR KV1:KEY;
 15.16.40?VAR SV1:STRING;
 15.17.20? VAR R1,SQ1:REAL
 15.18.19?   KV1=.41;  R1=2; SV1='SQ1=SQR(R1);'
 15.20.21?
 15.20.53?WRITE 'ALPMAX=1.047',.45
 15.21.18?LIST .45
   0.45  =ALPMAX=1.047
 15.22.11?

         WRITE SV1,KV1+.06
 15.22.38?LIST .47
  0.47  =SQ1=SQR(R1);
 15.22.58?

         WRITE CAT(SV1,'  PRINTLN SQR') KV1
 15.23.40?LIST.41
   0.41  =SQ1=SQR(R1);   PRINTLN SQR
```

— The page to be edited is read
   from the data base library.

— Insert line number. .45 with the code,
   ALPMAX = 1.047.

— Insert line number .47 with the code,
   SQ1 = SQR(R1).

— Insert line number .41 with the
   code ΔΔΔ PRINTLN SQR.

```
09.29.23?FRAME W1:STRING;                           - Declare frame
09.29.26?
09.29.29?W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1         - Put a page into the W1
09.29.55?                                             frame.
        W1/WRITE 'THIS IS A COMMENT LINE',.32        - INSERT a new line (.32)
09.30.26?
        W1/LIST .3/.4                                - LIST new line
  0.3   =
  0.32  =THIS IS A COMMENT LINE
  0.4   =CONST  (* GLOBAL CONSTANTS *)
09.30.42?
```

[frame-id/]  DELETE   range   [:$\left\{\begin{matrix} NL|NK|NI \end{matrix}\right\}$], [V]]

The DELETE statement is used to remove lines from the ACTIVE frame or specified frame.  The range must be specified and may consist of the implicit  and/or explicit range discussed on page  55.  Upon deletion, the deleted lines and their line numbers will be displayed for the user's convenience unless the user has selected a display option.  The display option follows the range specification and is described below.

Display Option:    NL (NO LIST)    - do not display line number or the contents
                                      of the line being replaced

                   NI (NO ITEM)    - do not display the contents of the line,
                                      only the line number

                   NK (NO KEY)     - do not display the line number (KEY)
                                      only the contents of the line.

                   V (VETO)        - user may VETO or verify deletion by
                                      responding to 'OK?' with:

                                          Y - YES  - delete
                                          N - NO   - do not delete
                                          K - KILL - do not delete and abort the
                                                     command.

EXAMPLES:     IMPLICIT
 09.10.29?
 09.10.51?VAR KV1,KV2,KV3,KV4:KEY;
 09.10.59?VAR IV1,IV2,IV3,IV4:INT;
 09.11.04? KV2=.2;  KV3=2; KV4=6;  IV2=2;  IV3=5;
 09.11.09?
 09.11.13? USE ALIB.NEWVER.CKCASES.TEST.PROGRM    - Put a library page into the
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK        ACTIVE frame for editing.
 09.11.38?
 09.11.46?DELETE 'CHEOL'                          - Delete all lines containing
    1.   =    CHEOL = '$';                           CHEOL
 09.12.23?

            DELETE 'CH'(4)                        - Delete all lines containing
    0.5   =    CHFIRST = ':';                        CH beginning in column 4
    0.6   =    CHLAST = ';';
    0.7   =    CHBLANK = ' ';
    0.8   =    CHQUOTE = '''';
    0.9   =    CHZERO = '0';
    1.1   =    CHSEMI = ';';
    1.2   =    CHNOT = '#';
 09.13.01?

            DELETE 'IN'(44,45):NL                 - Delete all lines containing IN
13 ITEMS IN SPECIFIED RANGE.                         and beginning between columns
 09.13.59?                                           44 and 45.

            VAR SV1:STRING;   SV1='MAX';          - Delete all lines containing
 09.18.55? DELETE SV1 AND NOT 'IN' :NL               MAX (SV1) and not IN.  Do not
37 ITEMS IN SPECIFIED RANGE.                         want a print out (NL option).
 09.19.27?                                           ISIS informs the user of the
                                                     number of lines deleted (37)

67

EXPLICIT


```
08.58.01?VAR KV1,KV2,KV3,KV4:KEY;
08.59.26?VAR IV1,IV2,IV3,IV4:INT;
09.00.22? KV2=.2;  KV3=2; KV4=6;   IV2=2;   IV3=5;
09.01.08?
09.02.44? USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
09.03.26?
09.03.35?DELETE KV2
   0.2    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
09.03.53?
```
– Delete line .2 (KV2)

```
          DELETE .4(IV2)
  0.4    =CONST   (# GLOBAL CONSTANTS #)
  0.5    =   CHFIRST = ':';
09.04.24?
```
– Delete beginning with line .4 and delete up to 2 (IV2) lines.

```
          DELETE 1.1-KV2/KV3-.2:NL
10 ITEMS IN SPECIFIED RANGE.
 09.05.39?
```
– Delete lines beginning at .9 through 1.8.

```
          DELETE KV3/2.5(IV3-IV2)
  2.    =   BUFLEN = 7;
  2.1   =   BLKSZ = 256;
  2.2   =   BLKLEN = 250;
09.06.55?
```
– Delete lines beginning at 2 (KV3) through 2.5 with a limit of 3 lines.

```
     DELETE KV2+2, 5.1/KV4-KV2(IV2),KV3+1.4/KV4-KV3:NI
  5.1
  5.2
  3.4
  3.5
  3.6
  3.7
  3.8
  3.9
  4.
09.08.14?
```
– Delete 3 differe: ranges with the option t print only the line number. NOTE: The first range (line 2.2) has already been deleted in the above example.

```
15.14.00?FRAME W2:STRING;
 15.17.36?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2

15.19.20?W2/LIST .6
  0.6   =   CHLAST = ';';

15.19.30?W2/DELETE .6(IV2)
  0.6   =   CHLAST = ';';
  0.7   =   CHBLANK = ' ';
```
– Declare frame
– Put a library page into the W2 frame



– Delete lines residing in W2 frame

68

Text Editing Statement


[frame-id/]  REPLACE  range    [: NL ]



The REPLACE statement is used to replace existing lines in the ACTIVE frame
or specified frame.  The range must be specified and may consist of the implicit
and/or explicit range discussed on page 55.  The user is prompted for successive
lines to replace the lines in the specified range.  The prompt is a display of
the line to be replaced.  Following the range specification is the display
specification which is optional and described below.

If you don't want to replace the line, hit the  BREAK  key and the command
is aborted.

Display Options:      NL (NO LIST)    - do not list line numbers or
                                         contents of the line being
                                         replaced.


**EXAMPLES:**    IMPLICIT

USE ALIB.NEWVER.CKCASES.TEST.PROGRM   - The page to be edited is read from
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK     the data base library into the
                                                 ACTIVE frame.


```
09.16.53? VAR SV1,SV2,SV3,SV4,SV5: STRING;
09.17.12? VAR IV3,IV4: INT;
09.17.15?     SV1='BLK';  SV2='CH';  SV4='LEN';  SV5='BLO';
09.17.17?     IV3=36;     IV4=4;     SV3='LENGTH';
09.17.20?
          REPLACE 'CHQUO'
  0.8  =,  CHQUOTE = '''';
  0.8  =
09.17.45?
          REPLACE CAT(SV2,'ZERO')
  0.9  =   CHZERO = '0';
  0.9  =
09.18.08?
          REPLACE SV2(34,IV3)
  0.5  =   CHFIRST = ';';
  0.5  =
  0.6  =   CHLAST = ';';
  0.6  =
09.18.33?
          REPLACE CAT('DIR',SV4)(IV4)
  4.   =   DIRLEN = 62;
  4.   =
09.19.40?
          REPLACE SV1 AND NOT 'LEN'
  2.1  =   BLKSZ = 256;
  2.1  =
  2.3  =   BLKMAX = 3777B;
  2.3  =
  5.2  =   STATUSBLK = 1;
  5.2  =
```

- Replace all lines containing 'CHQUO'.
  REPLACE prompts the user with line number(s)
  and the user types the replacement line(s).

- Replace all lines containing the concati-
  nation of 'CH' (SV2) and 'ZERO'

- Replace all lines containing 'CH' (SV2) in
  columns 34 thru 36 (IV3).  REPLACE continually
  prompts the user with line numbers until all
  lines have been replaced.

- Replace all lines containing the concatination
  of 'DIR' and 'LEN' (SV4) with a limit of 4
  lines.

- Replace all lines containing 'BLK' (SV1) and
  NOT 'LEN'

**EXPLICIT**

```
09.20.31?VAR KV1,KV2,KV3,KV4: KEY;
09.20.34?VAR IV2:INT;
09.20.37?KV1=20., KV2=.2; KV3=2; KV4=6; IV2=2;
09.21.21?
        USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS WORK
09.21.47?
        REPLACE KV2
   0.2   =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   0.2   =
09.22.05?
        REPLACE KV1-19.5:NL;
   0.5   =
1 ITEMS IN SPECIFIED RANGE.
09.22.33?
        REPLACE 2.9/KV4-2(IV2)
   2.9   =    LFNDLEN = 250;
   2.9   =
   3.    =    HASHSZ = 1;
   3.    =
09.23.46?
        REPLACE KV2, 5.1/KV4-KV2(IV2), KV4-.1/KV4
   0.2   =
   0.2   =
   5.1   =
   5.1   =
   5.2   =    STATUSBLK = 1;
   5.2   =
   5.9   =    WVN = 0;
   5.9   =
   6.    =    VNLEN = 7;
   6.    =
09.24.38?
```

| - Read the page to be edited from the data base library into the ACTIVE frame |
| - Replace line .2 (KV2) |

- Replace line .5(KV1-19.5) with NO LIST.

- Replace lines 2.9 through and including 4(KV4-2) with a limit of 2 lines to total.

- Replace 3 separate ranges.

  1) line .2(KV2)
  2) line 5.1 through 5.8 (KV4-KV2) with a limit of 2 (IV2) lines.
  3) line 5.9 (KV4-.1) through 6 (KV4).

```
15.14.00?FRAME W2:STRING;
15.17.36?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2
```

- Declare frames
- Put a library page into the W2 frame

```
15.19.30?W2/REPLACE .9(IV2)
   0.9   =    CHZERO = '0';
   0.9   =
   1.    =    CHEOL = '$';
   1.    =

15.24.43?
```

- Replace lines in the W2 frame

Text Editing Statement

[frame-id/] CHANGE string-id$_1$ [(col$_1$ [,col$_2$])] TO string-id$_2$

IN range [:[NL|NI|NK],[E] ,[V] ,[M]]

The CHANGE statement is used to modify existing lines on the ACTIVE frame or specified frame. An item to be changed (string-id$_1$) and the change (string-id$_2$) are string operands. They can be a literal string enclosed in quotes, 'ABC,' a string variable or a STRING expression. The string to be changed (string-id) may be required to begin in a specific column (col) or begin between two columns (col$_1$,col$_2$).

The column specification is optional and if used must be enclosed in parenthesis following string-id. The column number must be an INT operand. It can be a simple number, INT variable or an INT expression. The range may consist of implicit and/or explicit ranges discussed on page 55. It should be noted that only the first occurrence of the string on a line is changed, unless the M or multiple occurrence option is selected. The display option follows the range specification and is described below.

| | | |
|---|---|---|
| Display Option: | NL (NO LIST) | - do not display the line number or contents of the line being changed. |
| | NI (NO ITEM) | - do not display the contents of theline, only the number. |
| | NK (NO KEY) | - do not display the line number (key) only the contents of the line. |
| | E (ECHO) | - display the old version of the line and the new version after the change has been made. |
| | V (VETO) | - user may veto or verify CHANGE by responding to: 'OK?' with:   Y - YES  - make the change<br>N - NO  - don't make the change<br>K - KILL - don't make the change and abort the command. |
| Multiple Occurrence Option | M | - change all occurrences of the string-id appearing on a single line. |

EXAMPLES: IMPLICIT

```
14.40.56?USE ALIB.NEWVER.CKCASES.TEST.PROGRM  - Page to be edited is read from
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK     the data base library.
  14.41.25?
          VAR SV1,SV2:STRING;
  14.42.14?  SV1='CH';  SV2='BVD';
  14.42.48?

          CHANGE SV1 TO CAT(SV1,'10') IN 'CH' AND 'IN'- Change CH to CH10
    0.5   =   CH10FIRST = ':';                          in all lines containir
    0.6   =   CH10LAST = ';';                           CH and IN.
```

```
14.43.38?CHANGE SV1 TO 'XX' IN 'CH'(29,36):M    - Change CH to XX in all lines
   0.6   =    XX10LAST = ';';                      containing a CH beginning in line
   0.7   =    XXBLANK = ' ';                       29 through 36.  Also if more than
14.44.27?                                          one occurence of CH appears in a
                                                   line change them all (opt = M).


           CHANGE '1' TO '150' IN SV2          - Change 1 to 150 in all lines
   3.4   =    BVDSZ = 150;                       containing BVD.
   3.6   =    BVDMAX = 777777B;
   3.7   =    BVDSYN = -150;
14.45.16?




14.45.49?CHANGE 'LEN' TO 'LLL' IN 'LEN' AND '250'   - Change LEN to LLL in all
   2.2   =    BLKLLL = 250;                            lines containing LEN and 250
   2.5   =    TRLRLLL = 250;
   2.7   =    NTNDLLL = 250;
   2.9   =    LFNDLLL = 250;
   3.1   =    HASHLLL = 250;
   3.5   =    BVDLLL = 250;
   4.8   =    PDXLLL = 250;
14.46.22?
```

EXPLICIT


USE PROGRM

```
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK  - The page to be edited is read
   14.30.47?VAR KV1,KV2,KV3,KV4:KEY;                from the data base library.

   14.31.16?VAR IV1,IV2,IV3,IV4:INT

   14.31.34?VAR SV1,SV2:STRING;
   14.32.50?
           KV2=.2;  KV3=2;  KV4=6;   KV1=.1;
   14.33.40?SV1='CH';   SV2='MAIN';
   14.34.05?
           CHANGE 'CHFIRST' TO CAT(SV1,'1') IN .5 - Change CHFIRST to CH1 in line 5
   0.5   =    CH1 = ':';
   14.34.51?


           CHANGE SV2 TO 'TEST' IN KV1(2)          - Change MAIN to TEST in line .1
   0.1   =PROGRAM TEST(SFILE,DFILE,OUTPUT+);          for 2 lines.
   0.2   =PROGRAM TEST(SFILE,DFILE,OUTPUT+);
   14.35.19?


           CHANGE SV1(29,36) TO 'XCH' IN KV2/KV3-1(2); - Change CH beginning in
   0.5   =    CH1 = ':';                                  line 29 through 36 to XCH
   0.6   =    CHLAST = ';';                                in lines .2 through 1 with
   14.36.10?                                               a limit of 2 lines.
```

72

```
        CHANGE 'MAX' TO 'MIN' IN KV3,2.3,KV4-.7/KV4(2):M  -- Change MAX
2. .   =   BUFLEN = 7;                                        MIN in 3 ranges.  Also chang
2.3    =   BLKMIN = 3777B;                                    all occurences of the string
5.3    =   LRUMIN = 777777B;                                  appearing in one line (M).
5.4    =   HOLDMIN = 7B;
                                                          (1) line 2
                                                          (2) line 2.3
                                                          (3) line 5.3 through 6 with a
                                                              limit of 2 lines.
```

## IMPLICIT AND EXPLICIT

```
14.50.05?USE ALIB.NEWVER.CKCASES.TEST.PROGRM  -- The page to be edited is read
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK     from the data base library.
14.50.15?
        VAR SV1,SV2:STRING;
14.50.22?  SV1='CH';  SV2='BVD';
```

```
COUNT 'MAX'                                   -- Count all lines containing MAX.
6 ITEMS IN SPECIFIED RANGE.
```

```
14.50.54?CHANGE 'MAX' TO 'MIN' IN 'BITS' IN 3/4 -- Change MAX to MIN in all
   3.6   =   BVDMIN = 777777B;                      lines containing BITS within
14.51.23?                                           the line numbers 3 through 4
```

```
09.35.46?FRAME W1:STRING;                     -- Declare frame
09.35.49?
09.35.52?W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1   -- Put a page into the W1
09.36.00?                                        frame.
        W1/LIST .1
   0.1   =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);  -- LIST a line
09.39.48?W1/CHANGE 'MAIN' TO 'EXPER' IN .1
   0.1   =PROGRAM EXPER(SFILE,DFILE,OUTPUT+);  -- Change a line in W1
                                                 frame.
```

[frame-id/]  ADD    string-id  [AT  col]  IN  range  [:[NL|NK|NI] ,[E] ,[V] ]

The ADD statement is used to alter existing lines on the ACTIVE frame or specified frame. The item to be added must be any type of STRING operand. It can be a simple string enclosed in quotes, a STRING variable, or a STRING function. The string-id may be required to be added AT a particular column (col). This AT col specification is optional and if not specified, the string-id will be appended to the end of the line. The column number must be an INT operand such as a simple number, an INT variable, an INT expression or an INT function. The range must be included and may consist of implicit and/or explicit ranges discussed on page 55. The altered line contents and line number are listed unless a display option has been specified. The display options are described below.

Display Option:     NL (NO LIST)     – do not display the line number or the
                                        contents of the line being added.

                    NI (NO ITEM)     – do not display the contents, only the
                                        number.

                    NK (NO KEY)      – do not display the line number (key) only
                                        the contents of the line.

                    ECHO             – display the old version of the line, and the
                                        altered version.

                    V (VETO)         – user may veto or verify the additions by responding
                                        to 'OK?' with:   Y – YES    – make addition
                                                         N – NO     – don't make addition
                                                         K – KILL   – don't make addition and
                                                                       abort the command.

EXAMPLES:      IMPLICIT

```
16.05.13?USE ALIB.NEWVER.CKCASES.TEST.PROGRM - The page to be edited is read
16.15.21?R.CKCASES.TEST.PROGRM USED AS WORK    from the data base library.
16.15.28?
        VAR SV1,SV2,SV3,SV4:STRING;
16.15.35?VAR KV1,KV2,KV3:KEY;
16.15.39?VAR IV1,IV2:INT;
16.15.43?
16.15.46?SV1='   PRINTLN;';  SV2='(#ADD-NO COL OPT#)';
16.15.50?SV3='X';  KV1=1.;  KV2=1.7;  IV1=20;
16.15.54?
16.15.58?ADD '*' AT IV1 IN '777B'- Add an * in column 20 in all lines containing 777B.
   2.3   =    BLKMAX = 3777B; *        (# MAX NUM OF BLOCKS/PAGE; 11 BITS #)
   3.6   =    BVDMAX = 777777B#;       (# MAX NUM OF BLOCKS/LIBRARY; 18 BITS #)
   5.3   =    LRUMAX = 777777B#;       (# MAX VALUE OF LRU; 18 BITS #)
16.16.40?


        ADD SV3 AT 9 IN 'LEN'(6)- Add an X in column 9 in all lines containing LEN
   1.5   =    IDLENX = 10;            (# LENGTH OF AN IDENTIFIER #) in column 6.
   1.6   =    NMLENX = 7;             (# LENGTH OF A DIRECTORY NAME #)
   1.7   =    INLENX = 82;            (# LENGTH OF AN INPUT BUFFER #)
   6.    =    VNLENX = 7;             (# STORED VERSIONS ALLOWED PER PAGE #)
```

EXPLICIT


16.05.13?USE ALIB.NEWVER.CKCASES.TEST.PROGRM  - The page to be edited is read from
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK    data base library.
16.05.36?
          VAR SV1,SV2,SV3,SV4:STRING;
16.07.13?VAR KV1,KV2,KV3:KEY;
16.07.37?VAR IV1,IV2:INT;
16.07.53?
16.07.57?SV1='   PRINTLN;';  SV2='(*ADD-NO COL OPT*)';
16.09.00?SV3='X';  KV1=1.;  KV2=1.7;  IV1=20;
16.09.46?
16.09.49?ADD '   PRINTLN;' AT IV1+9 IN .8(2)  - Add PRINTLN; at column 29 in lines
   0.8   =    CHQUOTE = '''';               PRINTLN; beginning with .8 for 2 lines.
   0.9   =    CHZERO = '0';                 PRINTLN;
16.10.23?                                   PRINTLN;
16.10.39?

          ADD SV1 IN KV2-.3                 - Add PRINTLN; to line 1.4
   1.4   =       PRINTLN;
16.10.59?


          ADD SV2 IN KV1/KV1+.2             - Add (*ADD-NO COL OPT*) in lines 1.
   1.   =    CHEOL = '$';(*ADD-NO COL OPT*)        through 1.2
   1.1  =    CHSEMI = ';';(*ADD-NO COL OPT*)
   1.2  =    CHNOT = '#';         (* ONLY USE OF THIS CHARACTER *)(*ADD-NO COL O
T*)
16.11.29?


          ADD '#' AT 20 IN .5,KV2-KV1       - Add # at column 20 in 2 ranges -
   0.5   =    CHFIRST = ';';  #      (* FIRST CHARACTER IN CHAR *) (1) line .5
   0.7   =    CHBLANK = ' ';  #      (* CHARACTER IDENTIFIERS *)   (2) line .7
16.14.05?




   15.14.00?FRAME W2:STRING;                        - Declare frames
   15.17.36?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRM  - Put a library page into the
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2           W2 frame

   15.19.30?W2/ADD '*' AT 21 IN '777B'              - Add lines in the W2 frame
   2.3   =    BLKMAX = 3777B;  *
   3.5   =    BVDMAX = 777777B;*
   5.3   =    LRUMAX = 777777B;*

MODIFY   range   [:S]


     The MODIFY statement allows the user to make changes in a line of code
without retyping the entire line.  Modify displays the line and then asks the
user for alterations to the line with the prompt, "Alters?".  The user then types
in the alterations using the following modify commands:

 

    space bar       - This leaves the character unchanged.

    B              - B will delete the character appearing directly
                      above it and replace it with a blank space.

    D              - D will delete the character appearing directly
                      above it and the rest of the text on the line
                      is shifted to the left one character.

    I              - I will insert a string of characters <u>before</u>
                      the character that appears directly above it.
                      The characters being inserted must be enclosed
                      in quotes and directly follow the I command.
                      NOTE that the characters appearing above this
                      inserted string cannot now be modified.  All
                      commands apply to the character that appear
                      directly above it.  The multiple pass option
                      can be used to solve this problem.

    R              - R  will replace any number of characters with
                      a new string beginning with the character
                      appearing directly above the R.  The string must
                      be enclosed in quotes and follow the R command.
                      NOTE that the characters appearing above the
                      replacement string cannot be modified.  All
                      commands apply to the characters that appears
                      directly above it.  The multiple pass option
                      can be used to solve this problem.

**The range must be specified and may consist of implicit and/or explicit range
discussed on page**  51 .  Modify provides multiple prompts to the user for a single
line, as many times as the user sees necessary to complete the modifications.
A carriage return       with no modifications preceding it will discontinue
prompts for the current line and go onto the next line in the range.  The BREAK
key will abort the command with no modification made.

A single modification option (:S) is available which provides the user with
only one prompt for modifying line.

EXAMPLE:

```
   13.37.27?USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS WORK

   13.38.17?MODIFY .1/.3
      0.1    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
      ALTERS?    DDD R'SUBP'    I'PA'
      0.1    =PROG SUBP(SFILE,PADFILE,OUTPUT+);
      ALTERS?         B  I'UB'
      0.1    =PROG SUB (SUBFILE,PADFILE,OUTPUT+);
      ALTERS?
      0.2    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
      ALTERS?
      0.3    =
      ALTERS?
   13.42.36?
```

- Modify lines 1 through 3
  with multiple modes per line.


- No more mods. Hit CR
- Modify line .2
- No mods. Hit CR
- Modify line .3
- No mods.
- Hit CR to abort command.

```
   09.51.47?FRAME W1:STRING;
   09.51.51?
   09.51.55?W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1
   09.52.00?
         W1/LIST .2
      0.2    =PROGRAM TEMP(SF,DF,OUTPUT+);
   09.52.05?
         W1/MODIFY .2:S
      0.2    =PROGRAM TEMP(SF,DF,OUTPUT+);
      ALTERS?         B      I'ILE'
      0.2    =PROGRAM T MP(SFILE,DF,OUTPUT+);
   09.53.45?
         W1/MODIFY .2
      0.2    =PROGRAM T MP(SFILE,DF,OUTPUT+);
      ALTERS?        D          I'ILE'
      0.2    =PROGRAM TMP(SFILE,DFILE,OUTPUT+);
      ALTERS?                        DDD
      0.2    =PROGRAM TMP(SFILE,DFILE,OUT+);
      ALTERS?

COMMAND ABORTED.  (ADDRESS:   17)
```

- Declare frame
- Put a page into the W1
  frame.

- LIST a line of W1 frame.

- Modify a single line in
  W1 frame. [:S]

- More Modifications required
  to line .2.

[frame-id/]   COPY    range   TO   {ℓn[//inc]}

The COPY command allows the user to copy an existing line or lines of code to another location within the ACTIVE frame or from a specified frame to the ACTIVE frame.  The frame copied to is always the ACTIVE frame.  The line or lines being copied remains unchanged and will now appear in both locations.  A range must be specified and may consist of implicit and/or explicit ranges discussed on page 55. The new location ( ℓn) must be specified.  The increment (inc) is optional and has a default value of SYSTEM.DELTA.  The new line location and increment are both KEY operands and can be any of the following.

ℓn and inc =

- number
- KEY variable
- KEY expression
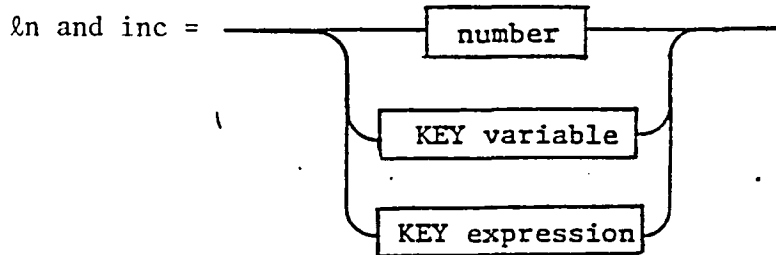
Upon execution of COPY, ISIS displays the number of lines copied in the last line number.

EXAMPLE:    IMPLICIT

```
16.04.39?  VAR IV3:INT;
16.05.40?  VAR STRT,INC1,INC2:KEY;
16.06.04?  VAR SV1,SV2,SV4:STRING;
16.06.23?
        USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
  16.06.54?
        IV3=36;   SV1='BLK';   SV2='CH';   SV4='LEN';
  16.08.09?INC1=.001;  INC2=.002;  STRT=7.4;
  16.09.26?
        COPY 'CHEOL' TO 7//INC2
1 LINES INSERTED. LAST LINE INSERTED IN WORK    :    7.
  16.09.53?
  16.10.09?COPY CAT(SV2,'ZERO') TO STRT//.001
1 LINES INSERTED. LAST LINE INSERTED IN WORK    :   7.4
  16.10.34?

        COPY SV2(34,IV3) TO STRT+.002//INC1+INC2
2 LINES INSERTED. LAST LINE INSERTED IN WORK    :  7.405
  16.11.29?


     COPY CAT('HASH',SV4) AND CAT(SV1,SV4) TO 7.9
1 LINES INSERTED. LAST LINE INSERTED IN WORK    :   7.9
  16.12.23?LIST 7.9
   7.9  =   HASHLEN = 250;      (* BLKLEN/HASHSZ *)
  16.12.43?
```

- Put a library page into the ACTIVE working frame for editing.

- Copy all lines containing CHEOL to line 7 with an inc of .002

- Copy all lines containing CHZERO string function to line 7.4 with .001 increment

- Copy all lines containing CH beginning in col 34 thru 36 to line 7.402 with a .003 increment.

- Copy all lines containing HASHLEN and BLKLEN to line 7.9 with a 1 increment (default)

EXPLICIT

```
09.51.56? USE ALIB.NEWVER.CKCASES.TEST.PROGRM    - Put a library page into the
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK       page for editing.
 09.52.54?
 09.53.02?   VAR S1,V1,V2,INC2,LN1,LN2,STR1:KEY;
 09.53.49?   VAR NL:INT;
 09.54.02?      V1=.1;  S1=.02;  V2=2.;  INC2=.001;
 09.54.52?      V5=5.;  NL=3;  LN1=1.1;  LN2=1.2;
VAR V5        : KEY
 09.55.58?      STRT=7.4;
VAR STRT      : KEY
 09.56.32?                                          - Copy line 1 to line
 09.56.46?COPY 1 TO STRT+S1
1 LINES INSERTED. LAST LINE INSERTED IN WORK      :   7.42  7.42
 09.57.10?LIST 7.42
   7.42  =    CHEOL = '$';


 09.59.13?LIST LN1(2); COPY LN1(2) TO V2+.01//.001; LIST 2.01/2.02
   1.1  =   CHSEMI = ';';
   1.2  =   CHNOT = '#';            (* ONLY USE OF THIS CHARACTER *)
2 LINES INSERTED. LAST LINE INSERTED IN WORK      :   2.011
   2.01  =   CHSEMI = ';';
   2.011 =   CHNOT = '#';          (* ONLY USE OF THIS CHARACTER *)
 10.05.41?


         LIST 1.1,1.2
   1.1  =   CHSEMI = ';';
   1.2  =   CHNOT = '#';          (* ONLY USE OF THIS CHARACTER *)
 10.07.57?



         COPY LN2-V1/2.+V1(SQR(NL-1)) TO V5+.03//.001; LIST 5.03/5.04
4 LINES INSERTED. LAST LINE INSERTED IN WORK      :   5.033
   5.03  =   CHSEMI = ';';
   5.031 =   CHNOT = '#';          (* ONLY USE OF THIS CHARACTER *)
   5.032 =   WDBLANKS = '       ';
   5.033 =

 08.45.57?FRAME W2:STRING                        - Declare frame name
 08.46.14?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM  - Put a library page into the
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2          W2 frame
 08.48.44?
         W2/COPY .3/.4 TO 10//.001                 - Copy lines from W2
2 ITEMS INSERTED. LAST ITEM INSERTED IN WORK  :  10.001   frame to the ACTIVE
                                                           frame
 08.51.40?LIST 10.001                             - List the list lines copied to
  10.001 =CONST  (* GLOBAL CONSTANTS *)             the Active page
 08.52.02?
         W2/LIST .3/.4                            - List the lines in W2 which were
   0.3  =                                           copied
   0.4  =CONST  (* GLOBAL CONSTANTS *)
```

[frame-id/]　　MOVE　　range　　TO　　{ ℓn[//inc] }

The MOVE command allows the user to move an existing line or lines of code from one location with the ACTIVE frame or from a specified frame to the ACTIVE frame. (The frame moved to is always the ACTIVE frame.) The line(s) being copied in the old location will no longer exist. A range must be specified and may consist of implicit and/or explicit ranges discussed on page 55. The new line location ( ℓn) and the incremental value (inc) are KEY operands. The increment (inc) is optional but must be specified when moving more than one line. They can be in the following form:

ℓn & inc =

```
+-----------------+
|     number      |
+-----------------+

+-----------------+
|  KEY variable   |
+-----------------+

+-----------------+
|  KEY expression |
+-----------------+
```
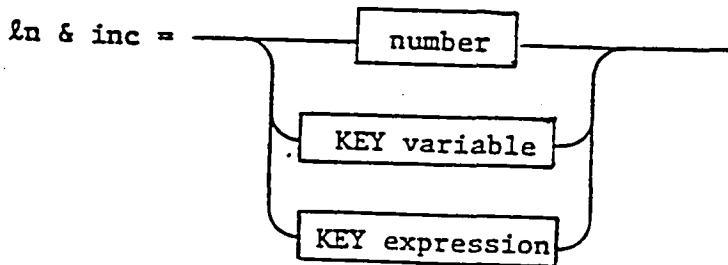
Upon execution of MOVE, ISIS displays the number of lines moved and the last line number.

EXAMPLES:　　IMPLICIT MOVE

```
16.15.16?   VAR SV1,SV3:STRING;
16.15.50?   VAR IV4,NUM:INT;
16.16.06?   VAR INC1,STRT:KEY;
16.16.24?     SV1='BLK';  SV3='CHS';   IV4=4;   INC1=.001;
16.17.05?     NUM=58;  STRT=7.4;
16.17.25?
```

USE ALIB.NEWVER.CKCASES.TEST.PROGRM — Put a library page into the ACTIVE
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK　　frame for editing.
16.17.59?

MOVE SV3 TO 7.1//INC1
1 LINES INSERTED. LAST LINE INSERTED IN WORK　　　:　　7.1 — Move all lines containing CHS to line
16.18.28?　　　　　　　　　　　　　　　　　　　　　　　　　　　7.1 with a .001 increment

LIST SV3
7.1　=　　CHSEMI = ';';
16.18.46?

— This listing shows the line has been deleted from its original location.

MOVE CAT(SV1,'MAX')(IV4) TO STRT+.1//INC1
1 LINES INSERTED. LAST LINE INSERTED IN WORK　　:　　7.5 — Move all lines containing BLKMAX in col 4 to line 7.5 with a .001 increment
16.19.31?

MOVE 'USE'(NUM DIV 2) TO STRT+.3//.001
2 LINES INSERTED. LAST LINE INSERTED IN WORK　　:　　7.701 — Move all lines containing USE beginning in col 29 to line 7.7 with a .001 increment
16.20.12?

LIST 'USE'
1.2　=　　CHNOT = '#';　　　　(* ONLY USE OF THIS CHARACTER *)
7.7　=　　INLIMIT = 80;　　　　(* USEFUL LENGTH OF INPUT BUFFER *)
7.701 =　　BLKLEN = 250;　　　　(* USEFUL CELLS PER BLOCK *)
16.20.34?

This LIST shows the lines containing USE in col 29 have been deleted from their original

```
          MOVE SV1 AND NOT 'LEN' TO 8//.001
.3 LINES INSERTED. LAST LINE INSERTED IN WORK        :    8.002
16.21.46?LIST 8/9
   8.   =    BLKSZ = 256;          (# SIZE OF BLOCKREC IN CELLS #)
   8.001 =   STATUSBLK = 1;        (# STATUS BLOCK NUMBER; MUST BE 1 #)
   8.002 =   BLKMAX = 3777B;       (# MAX NUM OF BLOCKS/PAGE; 11 BITS #)
16.22.15?
```

```
USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
09.31.48?
09.34.02? VAR V3,V5,LN1,STRT,INC2:KEY
09.34.57?VAR NL:INT
09.35.10?  V3=3;   V5=5;   LN1=1.1;   STRT=7.4;
09.35.59?  INC2=.002;      NL=3;
09.36.36?
09.36.42?MOVE LN1 TO STRT//.001;  LIST 7.4
1 LINES INSERTED. LAST LINE INSERTED IN WORK         :    7.4
   7.4  =    CHSEMI = ';';
09.37.20?
```

- The page to be edited is read from the data base library.

- Move line 1.1 to line 7.4 with .001 increment

```
          MOVE V3-.5(NL+NL) TO 7.9//INC2;  LIST 7.9/8.
6 LINES INSERTED. LAST LINE INSERTED IN WORK         :    7.91
   7.9  =    TRLRLEN = 250;
   7.902 =   NTNDSZ = 1;
   7.904 =   NTNDLEN = 250;
   7.906 =   LFNDSZ = 1;
   7.908 =   LFNDLEN = 250;
   7.91 =    HASHSZ = 1;
09.38.35?
```

- Move lines beginning with 2.5 for 6 lines to lines beginning at 7.9 with an increment of .002

```
          MOVE 4.5/V5-.4 TO 8.2//INC2;   LIST 8.2/9
2 LINES INSERTED. LAST LINE INSERTED IN WORK         :    8.202
   8.2  =    NDXSYN = -6;
   8.202 =   NDXFIRST = 7;
09.39.28?
```

- Move lines between 4.5 and 4.6 to line 7.9 with .002 increment

```
          MOVE V5/V5+.4(NL) TO STRT+1.3//INC2; LIST   8.7/9
3 LINES INSERTED. LAST LINE INSERTED IN WORK         :    8.704
   8.7  =    PDXFIRST = 8;
   8.702 =
   8.704 =   STATUSBLK = 1;
09.43.46?
          LIST V5/V5+.4(NL)
   5.3  =    LRUMAX = 777777B;
   5.4  =    HOLDMAX = 7B;
09.44.39?
```

- Move lines at 5 through 5.4 with limit of 3 lines to line 8.7 with an increment of .002.

```
08.45.57?FRAME W2:STRING
08.46.14?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2
08.48.44?
          W2/MOVE .3/.4 TO 8//.01
2 ITEMS INSERTED. LAST ITEM INSERTED IN WORK
08.59.36?
          LIST 8/9
   8.   =
   8.01 =CONST  (# GLOBAL CONSTANTS #)
09.00.35?
          W2/LIST .3/.4
NO ITEMS IN SPECIFIED RANGE.
```

- Declare frame
- Put a page into the W1 frame.

- Move lines from W2 frame to the Active frame

- LIST new lines of the WORK frame.

- Lines no longer appear in W2 frame.

[frame-id/]    REKEY  range    TO  { ℓn//inc}

The REKEY statement allows the user to change the line number (KEYS) assigned to a single line of code or a group of lines in the ACTIVE frame.  It should be noted that when rekeying, the line numbers must remain in ascending progression order.  The range for REKEY is limited.  The implicit part of range does not apply and the explicit range is limited, i.e., multiple ranges are not allowed and the nℓ total line designator is not permitted. nℓn is the beginning line number for the range being REKEYed and inc is the new increment for line numbers.

EXAMPLE:

```
09.15.17? VAR INC2,V4:KEY;
09.15.51?    INC2=.001;   V4=4;
09.16.13?
09.19.11?USE ALIB.NEWVER.CKCASES.TEST.PROGRM
ALIB.NEWVER.CKCASES.TEST.PROGRM USED AS WORK
09.20.34?
```
– The page to be edited is read from the data base library.

```
09.23.27?REKEY V4-.1/V4+.2 TO V4-.1//.001; LIST 3.9/4
4 ITEMS IN SPECIFIED RANGE. LAST KEY:   3.903
   3.9   =   DIRSZ = 4;
   3.901 =   DIRLEN = 62;
   3.902 =   DIRPGSYN = -5;
   3.903 =   DIRFIRST = 3;
09.24.18?
```
– Rekey from line 3.9 through 4.2 to 3.9 with a .00 increment this time.  The new lines are listed to show the new line numbers.

```
        LIST .F;LIST .L;   REKEY ALL TO .1//.01; LIST .F; LIST .L;
   0.1  =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   6.   =   VNLEN = 7;
60 ITEMS IN SPECIFIED RANGE. LAST KEY:    0.69
   0.1  =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   0.69 =   VNLEN = 7;
09.26.15?
```
In this example, the first and last lines of a page were listed; then the entire page was rekeyed then the first and last lines listed again show the entire page was rekeyed using the new line increment of .01.

```
12.54.17?FRAME W2:STRING;
12.54.40?W2/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W2
12.55.08?
```
– Declare frame name
– Put a library page into the W2 frame

```
        W2/LIST .F; W2/LIST .L
   0.1  =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   6.   =   VNLEN = 7;
12.55.38?
```
– List the first and last lines of the W2 frame

```
        W2/REKEY ALL TO 10.//.1
60 ITEMS REKEYED. LAST ITEM REKEYED IN W2
12.56.31?
```
– REKEY the complete page. New line number should begin with 10 and increase by .1

```
        W2/LIST .F; W2/LIST .L;
   10.  =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   15.9 =   VNLEN = 7;
12.57.01?
```
– List the first and last line numbers after rekeying

Text Editing Statements


[frame-id/]   COUNT   [range]   [:NC]


The COUNT statement allows the user to count the number of lines in the ACTIVE specified frame within a range.  The range is optional and, if not specified, the default is ALL.  A message indicating the number of items in the specified range is automatically printed unless otherwise indicated by the display option  NC, described below.


Display Option:   NC (No Confirmation)      - Avoids printing the COUNT message, "x items in specified range"

EXAMPLES:

```
18.12.17?LIST
   4.      =ISISTS1,T200,CM160000.
   6.      =USER,901300N.
   8.      =CHARGE,101481,LRC.
  10.      =GET,HALGO.
  12.      =GET,LST5=ATOM912.
  14.      =PACK,LST5.
 ·16.      =COPYBF,INPUT,INFIL.
  18.      =REWIND,INFIL.
  20.      =RFL,160000.
  22.      =REDUCE,-.
  24.      =NOEXIT.
  26.      =HALGO.
  28.      =)
```

- List the ACTIVE frame contents, a typical HAL/S applications program text

- EOR separator

```
18.12.31?COUNT
13 ITEMS IN SPECIFIED RANGE.
```

- Count number of lines in ACTIVE frame

```
18.13.15?
        COUNT 4/10
4 ITEMS IN SPECIFIED RANGE.
```

- Count number of lines between line number 4 and line number 10

```
18.13.34?COUNT 4/10,12/28
13 ITEMS IN SPECIFIED RANGE.
```

- Count total number of lines 2 ranges.


```
12.58.03?FRAME W3:STRING;
12.58.23?W3/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W3
12.58.45?
        W2/COUNT 'PROGRAM'
2 ITEMS IN SPECIFIED RANGE.
```

- Declare frame name
- Put a library page into the W3 frame

- Count the number of lines containing 'PROGRAM' in W3 frame

```
12.59.26?W3/LIST 'PROGRAM'
   0.1    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
   0.2    =PROGRAM MAIN(SFILE,DFILE,OUTPUT+);
12.59.47?
```

- Now list all lines containing 'PROGRAM' in W3 frame

[frame-id/]   EXEC   [range][:E ]


    The EXEC statement allows the user to execute the  contents of the ACTIVE or
specified frame.  Any portion of the frame may be executed by using the range option.
It should be noted that declarative statements (VAR, TYPE, etc) can only be
executed  once.        An error message is given if  an identifier is declared twice.


    Display Option:           E (ECHO)    - displays a line of code as it is executed.


    EXAMPLE:

```
14.04.51?LIST                          - List ACTIVE  frame
    1.      =A=10
    2.      =B=11
    3.      =X=A+B
    4.      =Y=X+A
    5.      =PRINTLN A,B,X,Y
14.05.31?EXEC                          - Execute the ACTIVE frame
  1.000000000000E+001   1.100000000000E+001   2.100000000000E+001   3.100000000
000E+001
```


```
14.12.32?EXEC :E                       - Execute the ACTIVE frame and print out each
    1.      =A=10                        line of code as it is executed.
    2.      =B=22
    3.      =X=A+B
    4.      =Y=X+A
    5.      =PRINTLN A,B,X,Y
  1.000000000000E+001   2.200000000000E+001   3.200000000000E+001   4.200000000
000E+001
```

**EXEC continued**

```
10.41.02?FRAME A1,A3:STRING;                                      - Define working frames
10.42.2?USE ALIB.ISIS.JOB.CONTROL.BUILD2                          - Read a library page into the ACTIVE
ALIB.ISIS.JOB.CONTROL.BUILD2 USED AS WORK                           frame
 0.43.12?LIST                                                     - List ACTIVE frame
   1.      A3/USE ALIB.ISIS.JOB.CONTROL.HEADER;  A3/RUN:NK
   2.      A3/USE ALIB.ISIS.JOB.CONTROL.COPY2;   A3/RUN:NK
   3.      A3/USE JOELIB.ISIS.SOURCE.PASINTF.SEPT18; A3/RUN
10.43.2?, A1/USE COPY                                             - Read a library page into the A1
ALIB.ISIS.JOB.CONTROL.COPY USED AS A1                               working frame
10.44.33?A1/LIST                                                  - List the A1 frame contents
   1.      =ATTACH,ISISCIO/M=W.
   2.      =REWIND,ISISCIO.
   3.      =COPYEI,INPUT,ISISCIO.
   4.      =)
 10.44.45?
         EXEC                                                     - Execute the ACTIVE frame
ALIB.ISIS.JOB.CONTROL.HEADER USED AS A3                           - USE statement executed
3 ITEMS IN SPECIFIED RANGE.                                       - RUN statement executed
ALIB.ISIS.JOB.CONTROL.COPY USED AS A3                             - Second line
4 ITEMS IN SPECIFIED RANGE.
JOELIB.ISIS.SOURCE.PASINTF.SEPT18 USED AS A3
699 ITEMS IN SPECIFIED RANGE..                                   - Third line
 10.46.28?
         SEND
'AWVIBBP' SENT TO BATCH EXECUTION.                               - INPUT file (generated by RUN) is sent
                                                                   to the CDC 6600 computer
```

Tool Invocation Statements

## RUN FILE (RFILE)

The RUN file is a file set up by the user as an input file for submission
to the NOS internal reader.  Records making up the RUN file are separated using
a right parenthesis [)] in column 1 as the EOR mark.  The RUN file is created
using the RUN statement and is executed (submitted to NOS) using the SEND
statement.  See Tool Invocation Statements for examples.  At present the
last line in the file must be a right parenthesis [)] in column 1.

Tool Invocation Statements

[frame-id/]  RUN  [range] [:[E], [NK]]

The RUN statement allows the user to create an INPUT file (control cards, program source and/or data) for submission to the NOS internal reader. This file, referred to as RUN file in ISIS, has the NOS name RFILE. A right parenthesis []] in the first column is used as an EOR mark for separating records such as the control cards, program source, and data on the RUN (input) file. If control cards are stored on one frame, program source on another, and data on still another, the user simply executes the RUN statement three times in succession. RUN will concatenate the indicated frames to the RUN (input) file for each time it is executed. The range is optional and if not specified, will be the entire frame. The line numbers (KEYS) will be stored in columns 81-98 if the NK option (described below) is not used.

Display Option:   E  (ECHO)   - Displays the frame contents as it is being added to the RUN file (RFILE)

NK (NO KEY)   - The line numbers are deleted as the lines are transferred to the RUN file.

EXAMPLES:

```
                FRAME A1,A3:STRING;           - Define working frames
  18.18.33?USE ALIB.ISIS.JOB.CONTROL.BUILD2   - Copy page from library into ACTIVE
ALIB.ISIS.JOB.CONTROL.BUILD2 USED AS WORK        working page.
  18.19.03?
          A3/LIST                            - List the ACTIVE frame
  1.      A3/USE ALIB.ISIS.JOB.CONTROL.HEADER;      A3/RUN:NK
  2.      A3/USE ALIB.ISIS.JOB.CONTROL.COPY2;       A3/RUN:NK
  3.      A3/USE JOELIB.ISIS.SOURCE.PASINTF.SEPT18;      A3/RUN
  18.19.23?
          A1/USE COPY                        - Read a library page into A1 working
ALIB.ISIS.JOB.CONTROL.COPY USED AS A1            frame
  18.19.50?LIST
  1.      A3/USE ALIB.ISIS.JOB.CONTROL.HEADER;      A3/RUN:NK
  2.      A3/USE ALIB.ISIS.JOB.CONTROL.COPY2;       A3/RUN:NK
  3.      A3/USE JOELIB.ISIS.SOURCE.PASINTF.SEPT18;      A3/RUN
  18.19.58?
          EXEC                               - Execute ACTIVE frame
ALIB.ISIS.JOB.CONTROL.HEADER USED AS A3      - Execution of USE statement }  1st line
3 ITEMS IN SPECIFIED RANGE.                  - Execution of RUN statement }  of ACTIVE
ALIB.ISIS.JOB.CONTROL.COPY2 USED AS A3} 2nd line                             frame is
5 ITEMS IN SPECIFIED RANGE.                                                  being
JOELIB.ISIS.SOURCE.PASINTF.SEPT18 USED AS A3} 3rd line                       executed
699 ITEMS IN SPECIFIED RANGE.
  18.20.37?
```

```
10.53.23?A1 USE ALIB.GRANT.BOOK.CHOP.CCA     - Put library page into working frame A1
ALIB.GRANT.BOOK.CHOP.CCA USED AS A1
 10.54.06?      A1/LIST                        - List working frame A1
    4.      =ISISTST,T200,CM160000.               RM1150      GRANTHAM
    6.      =USER,961300N.
    8.      =CHARGE,101481,LRC.
   10.      =GET,HALGO.
   12.      =GET,LST5=ATOM912.
   14.      =PACK,LST5.                          - A1 contains
   16.      =COPYBF,INPUT,INFIL.                   NOS control cards
   18.      =REWIND,INFIL.
   20.      =RFL,160000.
   22.      =REDUCE,-.
   24.      =NOEXIT.
   26.      =HALGO.
   28.      =)                                   - EOR separator
 10.54.21?
          USE PG1
ALIB.GRANT.BOOK.CHOP.PG1 USED AS ACTIVE       - Put another library page into ACTIVE
 10.54.57?LIST                                    frame
    2.      =     GOPROC:PROGRAM;                - List ACTIVE frame
    4.      =C      HALMAT TEST CASE ARRAYS
    6.      =C      R,S,T BEING INTEGER ARRAYS OF 10
    8.      =C      U BEING A 5X5 INTEGER ARRAY
   10.      =     DECLARE R ARRAY(10) INTEGER;
   12.      =     DECLARE S ARRAY(10) INTEGER;
   14.      =     DECLARE T ARRAY(10) INTEGER;
   16.      =     DECLARE U ARRAY(10) INTEGER;
   18.      =     DECLARE U ARRAY(5,5) INTEGER;
   20.      =     DECLARE INTEGER,A,B,C,D;
   22.      =     A=20;
   24.      =     B=18;
   26.      =     C=16;                          - User's source program
   28.      =     R$(5)=10;                        (HAL/S)
   30.      =     R$(7)=12;
   32.      =     U$(3,4)=15;
   34.      =     D=(A+B)+C;
   36.      =     D=A+(B+C);
   38.      =     A=R$(5);
   40.      =     S$(2)=R$(7);
   42.      =     A=U$(3,4);
   44.      =     U$(4,3)=A;
   46.      =     CLOSE GOPROC;
   48.      =)                                   - EOR separator
 10.56.58?
          A1/RUN                                - Create RUN (input) file (control cards)
13 ITEMS IN SPECIFIED RANGE.
 10.57.19?US
          RUN                                   - Add ACTIVE frame to RUN(input) file
24 ITEMS IN SPECIFIED RANGE.                      (program source)
 10.57.42?
          SHOW RUN                              - Display the number of lines on the
37 LINES IN RUN.                                  RUN(input) file.
 10.57.54?
          SEND                                  - Send RUN(input) file to NOS internal reader
'AWVIKXZ' SENT TO BATCH EXECUTION.             - NOS job identification is AWVIKXZ
 10.58.14?
```

88

SEND


The SEND command allows the user to submit the RUN(input) file to the NOS internal reader.  NOS will respond by printing the 7 character identification code for the job.  If the RUN(input) file happened to be empty, then a message is typed indicating this.


EXAMPLE:

```
14.57.38?SEND
'AWVIOYA' SENT TO BATCH EXECUTION.      - NOS accepts job and returns
14.57.59?                                 the job ID name 'AWVIOYA'
```


```
15.40.55?SEND
NOTHING TO SEND.                          - Empty RUN file
COMMAND ABORTED.   (ADDRESS:    8)
```

TOOL INVOCATION STATEMENTS


STOP:SEND


The STOP:SEND command allows the user to submit the contents of the RUN
(INPUT) file to the NOS internal reader in an interactive mode.  ISIS auto-
matically performs a STORE operation (page 47 ) to preserve the current
environment and transfers control to the control sequence defined by the RUN
file.  Upon completion of this sequence, control is returned to ISIS and
the current environment is automatically RESTORED (page  48 ).


EXAMPLE:


Interactive Commands


    USE    MYLIB. INTACT.TOOL.CONTROL.CARDS

    RUN
    STOP:SEND


Contents of    MYLIB. INTACT.TOOL.CONTROL.CARDS:
    .PROC name
    GET,INTERACTIVE_TOOL.
    ISISGET,INFILE.      MYLIB.INTACT.TOOL.INPUT.DATA
    INTERACTIVE_TOOL,INFILE,OUTFILE.
    REWIND,OUTFILE.
    ISISPUT,OUTFILE.     MYLIB.INTACT.TOOL.OUTPUT.INFO


This control card sequence gets the interactive tool and then calls
ISISGET to retrieve the tool's input data from the ISIS library page names
MYLIB.INTACT.TOOL.INPUT.DATA       and  put it on an NOS file named INFILE.
After tool execution (in the interactive mode), ISISPUT is called to
store the tools output, OUTFILE, on an ISIS library page named MYLIB.INTACT.
TOOL.OUTPUT.INFO.  Completion of the control card sequence returns control
to ISIS.

Interrogation Statements

SHOWN FILE

The SHOWN file will contain the output of the last SHOW statement which was executed using the KEEP option.  The information is saved here for the user in case he wishes to refer to it later on.  A message is displayed giving the total number of lines inserted and the line number of the last item on the file.

EXAMPLE:

```
    12.38.46?SHOW PAGES SUITLIB.CG.BOOK.EXEC. :KEEP
    6 ITEMS INSERTED. LAST ITEM INSERTED IN SHOWN      :    6.
```

          total number                    last line number
            of lines

SHOW SHOWS  [:KEEP]


A self-help type of command.  The SHOW SHOWS command prints a list
of all statements which the user may SHOW, or the words he may select to
follow the SHOW command verb.  The KEEP option will store the command output
on the SHOWN file.

EXAMPLE:

```
 10.21.38?SHOW SHOWS
  ABBREYS     AVAIL      CLEARS     COLUMNS    FRAMES    *FUNCS
   ID         INDEX      INDEXES    NAME       OPTIONS    PAGES
  *PROCS      RESERVED   RUN        SETS       SHOWS      STATEMENTS
   SUBS       TAGS      →TRAPS      TYPES      VARS      *VERSIONS
```



*Not available at present.
→Not available to user

Interrogation Statements

SHOW RESERVED [:KEEP]


The SHOW RESERVED statement is used to determine what words have been reserved for ISIS and cannot be used as identifier names in user programs. The KEEP option will store the command output on the SHOWN file.


EXAMPLE:

```
10.23.00?SHOW RESERVED
ABBREV     ACTIVE     ADD        ALL        AND        ARRAY
ASK        AT         BY         CALL       CHANGE     CLEAR
COMPARE    COMPILE    COPY       COUNT      DELETE     DIV
DOWNTO     DO         DUMP       ELSE       END        ERASE
EXEC       EXITIF     FALSE      FOREACH    FOR        FRAME
FROM       FUNC       IF         INSERT     INVOKE     IN
LIST       LOOP       MODIFY     MOD        MOVE       NOT
OF         ON         OR         OVER       PRINTLN    PRINT
PROC       PURGE      READ       RECORD     REKEY      REMOVE
REPEAT     REPLACE    RESTORE    RUN        SAVE       SEND
SET        SHOW       STOP       STORE      THEN       TO
TRUE       TYPE       UNION      UNTIL      USE        VAR
VOID       WHILE      WRITE      XEQ
```

SHOW STATEMENTS   [:KEEP]


    The SHOW STATEMENTS statement is used to display all available statement verbs in the ISIS system.  The KEEP option will store the command output on the SHOWN file.


EXAMPLE:


```
 10.24.02?SHOW STATEMENTS
ABBREV     ACTIVE     ADD        ASK        CALL       CHANGE
CLEAR      COMPARE    COMPILE    COPY       COUNT      DELETE
DUMP       ERASE      EXEC       FOREACH    FOR        FRAME
FUNC       IF         INSERT     INVOKE     LIST       LOOP
MODIFY     MOVE       PRINTLN    PRINT      PROC       PURGE
READ       REKEY      REMOVE     REPEAT     REPLACE    RESTORE
RUN        SAVE       SEND       SET        SHOW       STOP
STORE      TYPE       USE        VAR        VOID       WHILE
WRITE      XEQ
```


*Not implemented at present

Interrogation Statements


SHOW AVAIL  [:KEEP]



This statement allows the user to obtain an estimate of program resources which are still available.  The resource space already in use and remaining available are given in internal units.  Program resources listed are

        STRING    - string variable space
        TYPE      - program TYPES
        ID        - program identifiers
        PF        - procedures and function/names
        STACK,CORE,& XEQ  - system information (not directly controllable by the user)

          STACK  - values of all nonstring variables (temporarily during execution)
          CORE   - generated code(intermediate language)
          XEQ    - nesting  of XEQ, EXEC, and ASK

The KEEP option will store the command output on the SHOWN file.


EXAMPLE:


```
   10.24.53?SHOW AVAIL
            IN USE    AVAILABLE
            ------    ---------
   STRING:    32         118
   TYPE  :    12          51
   ID    :    34         116
   PF    :     0          15
   STACK :     7         993
   CORE  :    15         385
   XEQ   :     0          20
```


95

SHOW ABBREVS   [:KEEP]


The SHOW ABBREVS statement is used to determine which statement verbs have
been abbreviated by the user and their abbreviation(s).   The KEEP option will
store the command output on the SHOWN file.


EXAMPLE:


```
  10.25.23?SHOW ABBREVS
** NONE **
  10.25.32?ABBREV P,PR:PRINT
  10.25.53?SHOW ABBREVS
'PR         ': PRINT
'P          ': PRINT
```

SHOW TYPES  [:KEEP]


     The SHOW TYPES statement is used to allow the user to display all types which he has placed in the type-id table (systems declared TYPES are not presented). If the user wishes to inquire about one specific type, he should use the SHOW ID statement.  The KEEP option will store the command output on the SHOWN file.


EXAMPLE:

```
          SHOW TYPES                                        - No initial data types
** NONE **
 15.36.19?TYPE PERSONS:REAL                                 - Declare data types
 15.36.38?TYPE ADDR:RECORD   NUM:INT; NAM:STRING; END;
 15.37.16?
          SHOW TYPES                                        - New data types have
'ADDR      ': RECORD                                              been  added
               NUM: INT;
               NAM: STRING;
             END;
'PERSONS    ': REAL;
```

SHOW VARS  [:KEEP]

The SHOW VARS statement is used to determine which variables have been placed in the identifier table.  All declared variables are printed.  If the user wishes to inquire about one specific variable he should use the SHOW ID statement. The KEEP option will store the command output on the shown file.

EXAMPLE:

```
  14.00.47?SHOW VARS
** NONE **

  14.02.45?TYPE PERSONS:REAL
  14.03.03?TYPE MESS : ARRAY[1..5] OF BOOL          - Define new data types
  14.03.40?TYPE REC1 : RECORD NUM:INT; FLAG:BOOL; NAM:STRING; END;
  14.04.28?TYPE RECM : ARRAY[1..3] OF REC1;
  14.05.16?
           VAR ST1,ST2:STRING
  14.05.32?VAR ABC:REAL
  14.05.44?VAR B1,B2:BOOL
  14.05.56?VAR KIM,KATE:PERSONS                     - Declare new variables
  14.06.12?VAR AREC: REC1
  14.06.23?VAR BREC: RECM
  14.06.37?
           SHOW VARS
'ABC       ': REAL;
'AREC       ': RECORD
                NUM: INT;
                FLAG: BOOL;
                NAM: STRING;
             END;
'BREC       ': ARRAY [1..3] OF
                RECORD                              - Shows addition of new
                   NUM: INT;                          variables in the symbol
                   FLAG: BOOL;                        table
                   NAM: STRING;
                END;
'B1        ': BOOL;
'B2        ': BOOL;
'KATE      ': REAL;
'KIM       ': REAL;
'ST1       ': STRING;
'ST2       ': STRING;
```

Interrogation Statements

SHOW ID    {abbrev-id|var-id|system-id|frame-id}    [:KEEP]

The SHOW ID statement allows the user to obtain a description of program or system identifiers.  The identifier name, type, and usage is displayed.  The KEEP option will store the command output on the SHOWN file.

EXAMPLE:

```
14.28.22?VAR X: REAL;
14.28.31?VAR B,C: BOOL;
14.28.54?VAR I,J,K: INT;
14.29.19?VAR IND: STRING;
14.29.39?VAR AREC: RECORD NUM:INT; FLG:BOOL; END;
14.30.13?VAR RAY: ARRAY[1..3] OF REAL;
14.30.40?
          TYPE PERSONS=REAL;
14.30.59?
          AREC.NUM=22;
14.31.10?J=11;
14.31.32?       RAY[2]=125.35;
14.31.55?
          SHOW ID X,B,J,RAY,AREC,PERSONS
VARIABLE
'X         ': REAL;
VARIABLE
'B         ': BOOL;
VARIABLE
'J         ': INT;
VARIABLE
'RAY       ': ARRAY [1..3] OF REAL;
VARIABLE
'AREC      ': RECORD
                 NUM: INT;
                 FLG: BOOL;
              END;
TYPE
'PERSONS   ': REAL;
```

SHOW SETS  [:KEEP]


     The SHOW SETS statement displays all items the user may SET.  Most SET
statements may be reversed by a corresponding CLEAR statement.  The KEEP
option stores the command output on the SHOWN file.


EXAMPLE:


     10.26.47?SHOW SETS
      INDEX        NAME         TAG         TRACE       → TRAP       → TRAPS
    * VERSION


    *Not available at present
    →Not available to user

SHOW CLEARS   [:KEEP]


     The SHOW CLEARS statement is used to determine which items can be cleared.
CLEAR is used to reverse the effect of a SET command.   The KEEP option stores
the command output on the SHOWN file.



EXAMPLE:


```
    10.27.04?SHOW CLEARS
   INDEX         INDEXES     NAME          RUN          TAG          TRACE
  →TRAP          →TRAPS      *VERSION
```


    *Not available at present
    →Not available to user

[frame-id/]  SHOW NAME


     The SHOW NAME displays the current name of the ACTIVE or specified frame.
The user also has the option of displaying the library names of frames which
are not ACTIVE by preceding the command with the frame name and a slash.


EXAMPLE:


     10.27.24?SHOW NAME                           - The library page associated
     ....  IS NAME OF WORK                          with the ACTIVE frame has not
     10.27.37?                                      yet been named.
             SET NAME IALIB.SHELF.BOOK.CHAPTER.PAGE    - Set name of the page
     IALIB.SHELF.BOOK.CHAPTER.PAGE  IS THE NAME OF WORK   associated with the
     10.28.40?                                         ACTIVE frame.
             SHOW NAME                              - Show the library page
     IALIB.SHELF.BOOK.CHAPTER.PAGE  IS NAME OF WORK      associated with the
     10.29.02?                                         ACTIVE frame.
             FRAME W1,W2:STRING
     10.30.21?
             W1/USE ALIB.NEWVER.CKCASES.TEST.PROGRAM
     ALIB.NEWVER.CKCASES.TEST.PROGRAM USED AS W1
     10.30.45?
             W1/SHOW NAME
     ALIB.NEWVER.CKCASES.TEST.PROGRAM  IS NAME OF W1
     10.31.20?
             W2/SHOW NAME
     ....  IS NAME OF W2

Interrogation Statements

SHOW PAGES {[library].[shelf].[book].[chapter].[page]}        [:KEEP]


    The SHOW PAGES statement is used to display the structure of a complete
library or a portion of a library.  To display a complete library the user
need only type the top level of the library name followed by 4 dots separated
by blanks (SHOW PAGES ALIB . . . .).  All pages for all levels of the library
will be displayed.  If the library name is not included in the statement
(SHOW PAGES . . . .), it will use the library name of the ACTIVE frame.  Also,
for the users convenience, this command may be shortened by not typing the
last 3 dots (SHOW PAGES .).  This is the only level at which an assumption
is made.  If other level names are not included in the statement, then the library
is searched for the specific combination of these levels which are specified
and all combinations are listed.  The SHOW PAGES may also be used to search
for certain page names.  For example, to find all page names in a specific
chapter, the user would type the following:  SHOW PAGES  LIB.SHL.BK.CHAPTER. .
To find all pages named SAM on a particular shelf, the user would type:
SHOW PAGES .SHL. . .SAM.  All books and chapters on that shelf would be searched
for pages named SAM.  The name specified in SHOW PAGES can thus be used to
define an area of search for all pages or for specifically named pages within
a desired region.  The KEEP option will store the command output in the
SHOWN file.
    Note that this command (SHOW PAGES) does not in any way affect the name
of any frame.  This command only displays the names of pages of a library that
have previously been saved.

EXAMPLE:

```
 08.57.34?set name alib.newver.ckcases.test.program      - Set library page
ALIB.NEWVER.CKCASES.TEST.PROGRAM  IS THE NAME OF WORK       name associated
                                                            with ACTIVE frame
 08.58.37?show pages .                                   - Display the entire library.
ALIB    .NEWVER .CKCASES.TEST     .SHOWS                    (uses library name of ACTIVE frame.)
  .          .        .EXPL    .EXEC
  .          .        .EXEC    .SHOWS
  .          .        .TERM    .SETUP
  .          .EDITOR .TEST     .PROGRAM
  .          .        .EXPL    .DELETE
  .          .        .        .INSERT
  .          .        .        .LIST


 08.59.52?show pages suitlib. . . .                      - Display the entire library.
SUITLIB.SHELF  .BOOK    .SUIT    .NAMLIST                   (specifying the library name.)
  .          .        .        .EXEC
  .          .        .        .NAMBASH
  .          .        .        .ONEPLIN
  .          .        .NAMELST.PROGRAM
  .CG        .BOOK    .CHANGE  .NAMELST
  .          .        .        .NOPAREN
  .          .        .        .PAREN
```

SHOW PAGES   continued

```
  14.18.32?SHOW PAGES ALIB.GRANT. . .            - Show all pages contained in
ALIB    .GRANT  .BOOK    .CHOP    .PG2             ALIB on the GRANT shelf
          .        .        .      .PGALL
          .        .        .      .CCA
          .        .        .      .PG1
  14.18.54?SHOW PAGES SOURCE                      - Show where the SOURCE pages
ALIB    .MUST   .ISIS    .EDITOR .SOURCE            are located
          .        .      .DATBASE.SOURCE
        .MISC   .NEWPOOP.DATBASE.SOURCE
  14.19.39?SHOW PAGES ALIB.MUST.ISIS.DATBASE. - Show all pages in the DATBASE
ALIB    .MUST   .ISIS    .DATBASE.USERDOC          chapter
          .        .        .      .SPECS
          .        .        .      .SOURCE
          .        .        .      .CMDLST
  14.21.18?SHOW PAGES . .ISIS.  .SPECS          - Show all chapters containing
ALIB    .MUST   .ISIS   .EDITOR .SPECS            SPECS residing in the ISIS
          .        .      .DATBASE.SPECS          book
```

Interrogation Statements

SHOW OPTIONS  [:KEEP]


The SHOW OPTIONS statement is used to determine the print options which are available for some editor commands.  These options allow the user to modify the printed output resulting from the editor statements.  The KEEP option will store the command output in the SHOWN file.  Most options are discussed on page    . (CHANGE statement)


EXAMPLE:



    10.39.09?SHOW OPTIONS
    ECHO        NI          NK          NL          M           T
    V

SHOW COLUMNS  [:KEEP]

    The SHOW COLUMNS statement is used to show position of code in a line. A line of 59 characters is printed.  This line is formed by repeating the character sequence "123456789."  The KEEP option will store the command output in the SHOWN file.

EXAMPLE:

```
10.39.31?SHOW COLUMNS
         123456789.123456789.123456789.123456789.123456789.123456789
10.42.52?LIST 1.
   1.    =ISISC,T200,CM60000.                          RM1150       GRANTHAM-ISIS
```

                                              ↑
                                            └This is column 40 in
                                              this line of code

SHOW RUN  [:KEEP]


     The SHOW RUN statement determines the number of lines of code in the
INPUT file and then prints the number.  The KEEP option will store the command
output on the SHOWN file.


EXAMPLE:

```
    13.39.49? LIST
       4.     =ISISTST,T200,CM160000.
       6.     =USER,961300N.
       8.     =CHARGE,101481,LRC.
      10.     =GET,HALGO.
      12.     =GET,LST5=ATOM912.
      14.     =PACK,LST5.
      16.     =COPYBF,INPUT,INFIL.
      18.     =REWIND,INFIL.
      20.     =RFL,160000.
      22.     =REDUCE,-.
      24.     =NOEXIT.
      26.     =HALGO.
      28.     =)
      30.     =   GOPROC%PROGRAM;
      32.     =   DECLARE R ARRAY(10) INTEGER;
      34.     =   DECLARE S ARRAY(10) INTEGER;
      36.     =   DECLARE T ARRAY(10) INTEGER;
      40.     =   DECLARE U ARRAY(5,5) INTEGER;
      42.     =   DECLARE INTEGER,A,B,C,D;
      44.     =   A=20;  B=18;   C=16;
      46.     =   R$(5)=10;
      48.     =   R$(7)=12;
      50.     =   U$(3,4)=15;
      52.     =   D=(A+B)+C;
      54.     =   D=A+(B+C);
      56.     =   A=R$(5);
      57.     =   CLOSE GOPROC;
      60.     =)
      62.     =
    13.40.13?CLEAR RUN
   RUN CLEARED.
    13.41.09?RUN 4/24
   11 ITEMS IN SPECIFIED RANGE.
    13.41.36?RUN 28/60
   16 ITEMS IN SPECIFIED RANGE.
    13.41.54?SHOW RUN
   27 LINES IN RUN.
```

RM1150     GRANTHAM

LaRC/Control card file for
- compiling and executing a
  typical HAL/S program.

- EOR separator

- Typical HAL/S application
  program text

- EOR separator

- Clear INPUT file

- Put lines 4 through 24 on the RUN file.

- Add lines 28 through 60 to RUN file

- Display the total number of lines
  in the RUN file

# Appendix A

## Showing Equivalence Between Statement Verbs and Interrogation Statement

| Programming Statements | Page No. | Interrogation Statements | Page No. |
|---|---|---|---|
| **Declarative Statements** | | | |
| ABBREV abbreviation(s): statement-verb | 18 | SHOW ABBREVS | 96 |
| TYPE type-id(s): type specification | 19 | SHOW TYPES | 97 |
| VAR var-id(s): type-id | 20 | SHOW VARS | 98 |
| ERASE {abbrev-id(s)\|type-id(s)\|var-id(s)} | 21 | SHOW ID | 99 |
| **Action Statements** | | | |
| EXITIF condition | 23 | | |
| IF condition THEN {statement(s) [ELSE statement]} END [EXITIF condition] | 24 | | |
| FOR var-id = initial value TO DOWNTO final-value DO { statement(s) [EXITIF condition] } END | 25 | SHOW SHOWS | 92 |
| | | SHOW RESERVED | 93 |
| LOOP [statement(s)] EXITIF condition [statement(s)] END | 26 | SHOW STATEMENTS | 94 |
| | | SHOW AVAIL | 95 |
| WHILE condition DO { statement(s) [EXITIF condition] } END | 27 | | |
| REPEAT { statement(s) [EXITIF condition] } UNTIL condition | 28 | | |
| [frame-id/]FOREACH string-var DO { statement(s) [EXITIF condition] } END | 29 | | |
| XEQ string-expression | 30 | | |
| SET TAG tag-id | 31 | | |
| CLEAR TAG | 31 | | |
| SET TRACE var-id(s) | 33 | SHOW SETS | 100 |
| CLEAR TRACE var-id(s) | 33 | SHOW CLEARS | 101 |
| ASK response, prompt | 34 | | |
| PRINT, PRINTLN exp [:Format1[:Format2]] | 36 | | |
| CLEAR RUN | 38 | | |
| **Library Statements** | | | |
| [frame-id/]SET NAME [library].[shelf].[book].[chapter].[page] | 42 | SHOW NAME | 102 |
| [frame-id/]USE [library].[shelf].[book].[chapter].[page] | 43 | ⟨SHOW PAGES⟩ | 103 |
| [frame-id/]SAVE [*] | 44 | | |
| [frame-id/]PURGE [library].[shelf].[book].[chapter].[page] | 45 | | |
| [frame-id/]VOID | 46 | | |
| STORE [library].[shelf].[book].[chapter].[page] | 47 | | |
| RESTORE [library].[shelf].[book].[chapter].[page] | 48 | | |
| **Text Editing Statements** | | | |
| FRAME frame-id(s) : STRING | 56 | | |
| ACTIVE frame-id | 57 | | |
| ERASE frame-id(s) | 58 | | |
| [frame-id/]LIST [range] [:{NI\|NK}], [V], [T] ] | 59 | | |
| [frame-id/]INSERT range | 61 | | |
| [frame-id/]READ string-var {Δ\|,} $\ell n$ | 63 | | |
| [frame-id/]WRITE string-id {Δ\|,} $\ell n$ | 65 | | |
| [frame-id/]DELETE range [:[{NL\|NI\|NK}], [V] ] | 67 | | |
| [frame-id/]REPLACE range [:NL] | 69 | | |
| [frame-id/]CHANGE string-id TO string-id IN range [:[{NL\|NI\|NK}], [E], [V], [M] ] | 71 | | |
| [frame-id/]ADD string-id [AT column] IN range [:[{NL\|NI\|NK}], [E], [V] ] | 74 | ⟨SHOW OPTIONS | 105 |
| [frame-id/]MODIFY range [:S] | 76 | SHOW COLUMNS⟩ | 106 |
| [frame-id/]COPY range TO $\ell n$[//inc] | 78 | | |
| [frame-id/]MOVE range TO $\ell n$[//inc] | 80 | | |
| [frame-id/]REKEY range TO $\ell n$[//inc] | 82 | | |
| [frame-id/]COUNT [range] | 83 | | |
| [frame-id/]EXEC [range] [:E] | 84 | | |
| **Tool Invocation** | | | |
| [frame-id/]RUN [range] [:[E], [NK] ] | 87 | SHOW RUN | 107 |
| SEND | 89 | | |
| STOP:SEND | 90 | | |

NOTE: ⟨ ⟩ - no corresponding statement

APPENDIX B

Alphabetical Listing of Statement Verbs

APPENDIX C

Local Files Used by ISIS

|  |  |
|---|---|
| INPUT* | - A system file |
| ISIS | - Binary of ISIS |
| RFILE | - RUN file created by the ISIS RUN command |
| SFILE | - Used internally by SHOW and EXEC commands |
| INPUT | - Terminal input file |
| OUTPUT | - Terminal output file |
| WORK | - ISIS WORK frame |
| SHOWN | - ISIS SHOWN frame |
| WORK1-WORK10 | - ISIS user defined frames |

RFILE is the only file in NOS format that can be looked at outside of ISIS.

## IPL Error Messages

```
'[' EXPECTED
']' EXPECTED
':' EXPECTED
';' EXPECTED
'(' EXPECTED
')' EXPECTED
',' EXPECTED
'.' EXPECTED
'/' EXPECTED
':=' EXPECTED
'=' OR ':' EXPECTED.
'PROC' EXPECTED
'THEN' EXPECTED
'END' EXPECTED
'UNTIL' EXPECTED
'DO' EXPECTED
'OF' EXPECTED
'IN' EXPECTED
'ON' EXPECTED
'ON' OR IDENTIFIER EXPECTED.
'TO' EXPECTED
'TO/DOWNTO' EXPECTED
IDENTIFIER EXPECTED
'.' OR IDENTIFIER EXPECTED.
VARIABLE EXPECTED
PROCEDURE ID EXPECTED
FRAME ID EXPECTED
SCALAR EXPECTED
INTEGER EXPECTED
STRING EXPECTED
STRING VARIABLE EXPECTED
BOOLEAN EXPECTED
KEY EXPECTED
KEY INCREMENT EXPECTED
FIELD ID EXPECTED
OPERATOR EXPECTED
RANGE EXPECTED
KEY RANGE EXPECTED

UNDECLARED IDENTIFIER
IDENTIFIER ALREADY IN USE

PARSE STACK OVERFLOW
LOCATION COUNTER STACK OVERFLOW
CONSTANT TABLE OVERFLOW
ID TABLE OVERFLOW
JUMP TABLE OVERFLOW
PROC/FUNC TABLE OVERFLOW
TYPE TABLE OVERFLOW
FRAME TABLE OVERFLOW
STRING BUFFER OVERFLOW
```

TEMPORARY STRING BUFFER OVERFLOW

MAY NOT BE ERASED
NOT ENOUGH ROOM FOR TYPE
PAGE DOES NOT CONTAIN CODE
UNDECLARED RECORD FIELD
UNDECLARED ARRAY FIELD
MISMATCHED OPERATOR
CORE TOO SMALL
ERROR IN TYPE
LOW BOUND EXCEEDS HIGH BOUND
KEYS IN BAD ORDER
BAD KEY
LIMIT COUNT REQUIRED
ERROR IN SUBROUTINE PARAMETER
ERROR IN FACTOR
BAD BLOCK NUMBER
INCOMPATIBLE TYPES
NOT A SYSTEM VARIABLE
VERSION NUMBER MUST BE INTEGER
READ-ONLY VARIABLE
XEQ STRING NOT COMPLETE COMMAND
BAD INCREMENT VALUE
MISSING OR BAD NAME
MISSING STATEMENT VERB
UNIMPLMENTED STATEMENT
MISSING CLOSING QUOTE FOR STRING
TOO MANY DIGITS IN INTEGER
EXPONENT OUT-OF-RANGE
UNRECOGNIZED STATEMENT
USE 'SHOW TRAPS'
USE 'SHOW SETS'
USE 'SHOW CLEARS'
USE 'SHOW SHOWS'
ONLY ONE KEY RANGE ALLOWED HERE
IMPROPER USE OF RESERVED WORD
MUST PRINT AT LEAST ONE EXPRESSION
TOO MANY DOTS IN NAME
TOO MANY CHARACTERS IN NAME
USE 'SHOW OPTIONS'
OPTION MUST FOLLOW ':'
'LIST:NL' IS NONSENSICAL
'NL' IMPROPER WITH VETO.
INAPPROPRIATE USE OF USING
INAPPROPRIATE DATASET
'ON' AND 'OVER' NOT ALLOWED AS INDEXES
STATEMENT CANNOT BEGIN WITH OPERATOR
UNRECOGNIZED ASK EXPRESSION
ASK EXPRESSION OF INCOMPATIBLE TYPE
INCORRECT SYNTAX FOR COPY/MOVE

References

1.  Jensen, Kathleen; and Wirth, Kiklaus:  PASCAL User Manual and Report.
    2nd edition, Sprunger-Verlag, New York, NY, 1976.

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| NASA TM-80144 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| ISIS Users Manual | March 1980 |
| | 6. Performing Organization Code |

| 7. Author(s) | 8. Performing Organization Report No. |
|---|---|
| Carolyn Grantham | |
| | 10. Work Unit No. |

| 9. Performing Organization Name and Address | 520-72-03-02 |
|---|---|
| NASA Langley Research Center | 11. Contract or Grant No. |
| Hampton, VA 23665 | |
| | 13. Type of Report and Period Covered |

| 12. Sponsoring Agency Name and Address | Technical Memorandum |
|---|---|
| National Aeronautics and Space Administration | 14. Sponsoring Agency Code |
| Washington, DC 20546 | |

**15. Supplementary Notes**

**16. Abstract**

The Interactive Software Invocation System (ISIS) is an interactive data management system. ISIS is being developed to provide the user with a powerful system for developing software in an interactive environment. ISIS will protect the user from the idiosyncracies of the host computer system by providing a complete range of capabilities including desk top calculator, data and text editor, file manager, and tool invoker. The user should have no need for direct access to the host computing system. This documentation covers the operational concepts and syntax of the Interactive Programming Language, IPL, for ISIS.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| Software development<br>Interactive software<br>Text Editor<br>File management | Unclassified - Unlimited<br><br>Subject Category 61 |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price* |
|---|---|---|---|
| Unclassified | Unclassified | 116 | $6.50 |