# SYSTEM 2000 LEVEL 2
## A MULTI-PURPOSE DATA MANAGEMENT SYSTEM

Mini Manual

CYBERNET® SERVICE

CONTROL DATA CORPORATION

# SYSTEM 2000 LEVEL 2

## A MULTI-PURPOSE DATA MANAGEMENT SYSTEM

CYBERNET

Mini Manual

# REVISION RECORD

| REVISION | DESCRIPTION |
|---|---|
| (7-15-72) | Initial Printing |
| A | Complete reformatting and revision supersedes previous |
| (4-15-74) | edition. |
| B | Added a new chapter on the Report Writer feature and |
| (5-15-75) | implemented the KRONOS "Family" concept. |

Publication No.
86615600

Address comments concerning
this manual to:

CONTROL DATA CORPORATIO
Special Services and Publicatioı
HQW05F, P.O. Box 0
Minneapolis, Minnesota 55440

# PREFACE

Control Data's CYBERNET® Service offers the SYSTEM 2000* general purpose data base management system on CDC® 6600 and CDC® 6400 computer systems which run under the SCOPE and KRONOS® operating systems. As a result, SYSTEM 2000 can be employed in both a batch (or remote batch) and interactive time-sharing mode.

This publication provides a brief description of the SYSTEM 2000 (Level 2) commands. It is intended to serve as a quick reference tool for the user. Section numbers appearing in the text of the first five chapters of the mini-manual refer to the corresponding section numbers in the SYSTEM 2000 User Information manual. Section numbers in chapters 6, 7, and 8 refer, respectively, to the corresponding sections in the Immediate Access Feature Supplement, the CDC Machine Dependent Information Supplement, and the Report Writer Feature Supplement manuals.

This manual assumes user familiarity with Control Data's KRONOS and/or SCOPE operating systems.

For further information concerning Basic SYSTEM 2000 usage or the optional features, consult the following manuals:

| Publication Title | Publication Number |
|---|---|
| SYSTEM 2000 Level 2 User Information Manual | 76074000 |
| SYSTEM 2000 Level 2 General Information Manual | 76074100 |
| SYSTEM 2000 Level 2 CDC Machine Dependent Information Supplement | 76074200 |
| SYSTEM 2000 Level 2 Diagnostic Message Supplement | 76074300 |
| SYSTEM 2000 Level 2 Immediate Access Feature Supplement | 76074400 |
| SYSTEM 2000 Level 2 COBOL Procedural Language Supplement | 76074500 |
| SYSTEM 2000 Level 2 FORTRAN Procedural Language Supplement | 76074600 |
| SYSTEM 2000 Level 2 Report Writer Feature Supplement | 76074700 |
| SYSTEM 2000 Level 2 Assembly Language Interface Supplement | 76074800 |
| SYSTEM 2000 Level 2 CONTROL 2000 Demonstration Script | 84003700 |
| SYSTEM 2000 Level 2 PERSONNEL 2000 Demonstration Script | 84003800 |

*SYSTEM 2000 is a service mark of MRI Systems Corporation.

# CONTENTS

# SYSTEM-WIDE OPERATIONS 1

## MODULE CALL COMMANDS (2.2.1)

| Format | Function |
|---|---|
| CONTROL: | Calls control module |
| DEFINE: | Calls define module |
| ACCESS: | Calls access module |

## COMMAND FILE CHANGE NAME COMMAND (2.2.2)

| Format | Function |
|---|---|
| COMMAND FILE IS <file name>: | Changes command file from INPUT to file name. |

## DATA FILE CHANGE COMMAND (2.2.3)

| Format | Function |
|---|---|
| DATA FILE IS <file name>: | Changes data file from INPUT to file name. |

## MESSAGE FILE CHANGE COMMAND (2.2.4)

| Format | Function |
|---|---|
| MESSAGE FILE IS <file name>: | Changes message file from OUTPUT to file name. |

## REPORT FILE CHANGE COMMAND (2.2.5)

| Format | Function |
|---|---|
| REPORT FILE IS <file name>: | Changes report file from OUTPUT to file name. |

## ENTRY TERMINATOR WORD CHANGE COMMAND (2.2.6)

| Format | Function |
|---|---|
| ENTRY TERMINATOR IS <terminator word>: | Changes terminator word from END to terminator word. |

## SYSTEM SEPARATOR CHANGE COMMAND (2.2.7)

| Format | Function |
|---|---|
| SEPARATOR IS <separator symbol>: | Changes system separator (*) to separator symbol. |

## ECHO COMMAND (2.2.8)

| Format | Function |
|---|---|
| ECHO <ON> or <OFF>: | To issue echoes of SYSTEM 2000 commands with their associated output; batch default is ON, interactive default if OFF. |

## EXIT COMMAND (2.2.9)

| Format | Function |
|---|---|
| EXIT: | Terminates the SYSTEM 2000 job. |

# CONTROL MODULE 2

## USER PASSWORD COMMAND (3.2.1)

```
USER, <password>:
```

where:

```
          <password> = a series of alphanumeric, non-blank characters
```

## NEW DATA BASE COMMAND (3.2.2)

```
NEW DATA BASE IS <data base name>:
```

where:

```
     <data base name> = a series of characters (single embedded blanks
                        allowed)
```

## DATA BASE NAME COMMAND (3.2.3)

```
DATA BASE NAME IS <data base name>:
```

where:

```
     <data base name> = a series of characters (single embedded blanks
                        allowed)
```

## VALID PASSWORD COMMAND (3.2.4)

```
VALID PASSWORD IS <password>:
```

where:

```
          <password> = a series of alphanumeric, non-blank characters
```

## INVALID PASSWORD COMMAND (3.2.5)

```
INVALID PASSWORD IS <existing password>:
```

where:

```
   <existing password> = name of any component-level password contained
                         in the data base
```

## ASSIGN AUTHORITY COMMAND (3.2.6)

ASSIGN <authority type list> TO <component list> FOR <password list>:

where:

| | |
|---|---|
| <authority type list> | = R, U, W, and/or V in any combination |
| <component list> | = ALL COMPONENTS, or a list of specific components by C number or name separated by commas |
| <password list> | = ALL PASSWORDS, or a list of specific passwords separated by commas |

## CHANGE PASSWORD COMMAND (3.2.7)

CHANGE PASSWORD <existing password> TO <new password>:

where:

| | |
|---|---|
| <existing password> | = the name held in the data base for either a valid password or the master password |
| <new password> | = a new name intended to replace a previous password |

## LIST PASSWORDS COMMAND (3.2.8)

| Format | Function |
|---|---|
| LIST PASSWORDS: | Obtains a listing of all valid passwords except the master. Only master password holder may use command. |

## LIST PASSWORDS AND AUTHORITIES COMMAND (3.2.9)

LIST PASSWORDS AND AUTHORITIES:

## RELEASE COMMAND (3.2.10)

| Format | Function |
|---|---|
| RELEASE: | Releases data base from disk |

## STATISTICS COMMAND (3.2.11)

| Format | Function |
|---|---|
| STATISTICS: | Obtains statistical information about the data base |

## SHARED DATA BASE NAME COMMAND (3.2.12)

SHARED DATA BASE NAME IS <data base name>:

where:

<data base name> = a series of characters (single embedded blanks allowed)

## SAVE DATA BASE COMMANDS (3.3.1)

(1) SAVE DATA BASE ON <idl>:
(2) SAVE DATA BASE ON <idl>/<id2>: (indirect is default)
(3) SAVE DATA BASE ON <idl>/<id2>/INDIRECT:
(4) SAVE DATA BASE ON <idl>/<id2>/DIRECT:

where:

<idl> = data base tape identification

<id2> = update file tape identification

## LOAD DATA BASE COMMANDS (3.3.2)

(1) LOAD <data base name> FROM <idl>:
(2) LOAD <data base name> FROM <idl>/<id2>: (indirect is default)
(3) LOAD <data base name> FROM <idl>/<id2>/INDIRECT:
(4) LOAD <data base name> FROM <idl>/<id2>/DIRECT:

where:

<data base name> = the name of the data base

<idl> = data base tape identification

<id2> = update file tape identification

## KEEP COMMAND (3.3.3)

KEEP:

## APPLY COMMANDS (3.3.4)

(1) APPLY ALL:
(2) APPLY THRU CYCLE <nn>:

where:

<nn> = last desired data base cycle number recorded on update file

## SUSPEND COMMAND (3.3.5)

SUSPEND:

# DEFINE MODULE 3

## DATA BASE COMPONENTS (4.2)

There are four component types within SYSTEM 2000: elements, repeating groups, user-defined functions, and strings.

All component declarations have a common basic format, regardless of component type. Each declaration is a separate command and each contains the following items:

- Component number
- System separator (default is an asterisk (*))
- Component name (character string of 1-250 characters with restrictions defined)
- Component description (component unique and set off by left and right parentheses)
- Colon (to signal the end of a component declaration command)

$$\left\langle\begin{matrix}\text{component}\\ \text{number}\end{matrix}\right\rangle\left\langle\begin{matrix}\text{system}\\ \text{separator}\end{matrix}\right\rangle\left\langle\begin{matrix}\text{component}\\ \text{name}\end{matrix}\right\rangle\left(\begin{matrix}\text{component}\\ \text{description}\end{matrix}\right):$$

### ELEMENTS (4.2.1)

The five parts of the component description, for an element, are as follows:

$$\left\{\begin{matrix}\text{KEY}\\ \text{NON-KEY}\end{matrix}\right\}\left[\begin{matrix}\text{data}\\ \text{type}\end{matrix}\right]\left\{\begin{matrix}\text{picture}\\ \text{designation}\end{matrix}\right\}\left\{\begin{matrix}\text{repeating group}\\ \text{relationship}\end{matrix}\right\}\left\{\begin{matrix}\text{padding}\\ \text{option}\end{matrix}\right\}$$

The items within braces { } are optional and each has a default setting if no specification is made. The items within brackets [ ] are mandatory.

#### KEY, NON-KEY DESIGNATION (4.2.1.1)

The command words are:

NON-KEY
KEY (default setting)

Examples of a key element declaration:

```
1* ORGANIZATION (KEY NAME):
2* COGNIZANT OFFICIAL (NAME):
```

An example of a non-key element:

```
3* ADDRESS (NON-KEY NAME):
```

## DATA TYPE DESIGNATION (4.2.1.2)

| Mandatory | Optional |
|---|---|
| [NAME] | ---- |
| [TEXT] | ---- |
| [DATE] | ---- |
| [INTEGER or INT] | {NUMBER or NUMB or NUM} |
| [DECIMAL or DEC] | {NUMBER or NUMB or NUM} |
| [MONEY] | {NUMBER or NUMB or NUM} |

## PICTURE DESIGNATION (4.2.1.3)

X = symbol for an alphanumeric character for NAME and TEXT data types
9 = symbol for a numeric character for INTEGER, DECIMAL and MONEY data types
XX or 99 = symbol for two characters
X(n) or 9(n) = symbol for the number of characters specified by (n)

| Data Type | Picture Designation | Picture Default |
|---|---|---|
| NAME or TEXT | {X, XX, XXX, XXXX, XXXXX, X(n)} where n = 1-250 characters | X(7) |
| DATE | None to be specified | |
| INTEGER | {9, 99, 999, 9999, 99999, 9(n)} where n = machine dependent value | 9(7) |

| Data Type | Picture Designation | Picture Default |
|---|---|---|
| DECIMAL | $\left\{\begin{array}{l}9\\99\\999\\9999\\99999\\9(n)\end{array}\right\}.\left\{\begin{array}{l}9\\99\\999\\9999\\99999\\9(n)\end{array}\right\}$ where n = machine dependent value | 9(6).9(2) |
| MONEY | $\left\{\begin{array}{l}\$\\ \\ \end{array}\right\}\left\{\begin{array}{l}\text{same picture}\\\text{specification}\\\text{as for decimal}\end{array}\right\}$ | 9(6).9(2) |

## REPEATING GROUP RELATIONSHIPS (4.2.1.4)

IN <repeating group number>

Example 1

```
12* STOCKS (RG IN 8):
   13* NAME OF STOCK (KEY NAME X(20) IN 12):
   14* TICKER SYMBOL (KEY NAME X(5) IN 12):
```

Example 2

```
1* ORGANIZATION (KEY NAME X(23)):
2* COGNIZANT OFFICIAL (NON-KEY NAME X(10)):
```

The implicit relationship of the above elements would be:

```
1* ORGANIZATION (KEY NAME X(23) IN 0):
                                └implied, but not specified
```

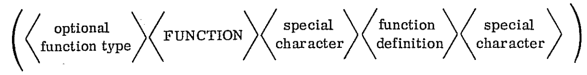
## REPEATING GROUPS (4.2.2)

Examples of defined repeating groups combined with the defined elements are as follows:

```
 1* ORGANIZATION (KEY NAME X(23)):
 6* ZIP CODE (NON-KEY INTEGER NUMBER 99999):
 7* CURRENT DATE (NON-KEY DATE):
 8* PORTFOLIOS (RG):
    9* PORTFOLIO NAME (KEY NAME X(10) IN 8):
   11* MANAGER (NON-KEY NAME X(14) IN 8):
   12* STOCKS (RG IN 8):
       13* NAME OF STOCK (KEY NAME X(20) IN 12):
```

## USER-DEFINED FUNCTIONS (4.2.3)

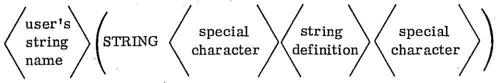
Functions are described by the user in his data base definition and apply only to that data base. Elements whose data values are types INTEGER NUMBER, DECIMAL NUMBER, MONEY or DATE can be used in a function; data types NAME and TEXT cannot be used. The component description format for the user-defined function is as follows:



Examples:

```
DEFINE:
C200*BLOCK COST (FUNCTION $(C30/C29)$):
ACCESS:
PRINT C13, *C200* WHERE NAME OF STOCK EXISTS:
PRINT CURRENT PRICE, *PE RATIO* WHERE CURRECT PRICE LE
   10.00:
PRINT *INTEREST* WHERE PAYMENT DATE EQ 06/12/72:
```

## STRINGS (4.2.4)



### SIMPLE STRING CONCEPT (EXAMPLE)

DEFINE:

500*ABCDEF(STRING/IF ALL OF(CURRENT PRICE GT 10.00*, CURRENT PRICE LT 20.00*, ESTIMATED EARNINGS GE 2.00*,ESTIMATED EARNINGS LE 4.00*) THEN PRINT STOCKS ELSE PRINT NAME OF STOCK WHERE ORGANIZATION EQ CITY TRUST COMPANY*:/):

### EXTENDED STRING CONCEPT (EXAMPLE)

DEFINE:

600*EXAMPLE 1(STRING/IF ALL OF(CURRENT PRICE GE *2**, CURRENT PRICE LT*3**,ESTIMATED EARNINGS GE *4**,ESTIMATED EARNINGS LE *5**) THEN PRINT STOCKS ELSE PRINT NAME OF STOCK WHERE ORGANIZATION EQ *1**:/):

To invoke the simple string, the commands and their sequence are as follows:

```
ACCESS:
QUEUE:
*C500* or *ABCDEF*
TERMINATE:
```

To invoke the extended string, the commands and their sequence are as follows:

```
ACCESS:
QUEUE:
*C600(CITY TRUST COMPANY,10.00,20.00,2.00,4.00)
                         or
*EXAMPLE1(CITY TRUST COMPANY,10.00,20.00,2.00,4.00)
TERMINATE:
```

## DATA BASE CONSTRUCTION (4.3)

The command stream required to construct a data base definition appears as follows:

| Format | Type of Command |
|---|---|
| USER, <password>: | Control module command |
| NEW DATA BASE IS <data base name>: | Control module command |
| Component declaration commands | Define module commands |
| MAP: | Define module commands |

### MAP COMMANDS (4.5.2)

MAP commands terminate and execute define module commands which were specified subsequent to the last DEFINE command.

```
(1) MAP:
(2) MAP WHERE <conditions exist>:
(3) MAP [ORDERED BY] <ordering list> WHERE <conditions exist>:
        [OB        ]
```

where:

<conditions exist> = series of legal WHERE clause conditions (see Section IA3.11)

<ordering list> = see Section IA3.8

### RENUMBER COMMANDS (4.5.3)

RENUMBER commands renumber all components within the data base definition.

```
(1) RENUMBER:                 (default values where <c> and <n> = 1)
(2) RENUMBER WITH <c>:
(3) RENUMBER WITH <c> INCREMENTING BY <n>:
```

where:

<c> = integer number standing for the component number of the first component in the data base after renumbering has taken place

<n> = integer number standing for the increments between component numbers

## CHANGE COMMANDS (4.5.4)

CHANGE commands change the component number, name, or description of a component currently defined in the data base definition.

```
(1) CHANGE <c1> TO <c2>:
(2) CHANGE <c1> TO <c2> <system separator> <component name>:
(3) CHANGE <c1> TO <c2> <system separator> <component name>
    (<component description>):
```

where:

<cn> = user component number without leading C

<component name> = a series of 1-250 characters (see Section Basic 4.2

<component description> = see Section Basic 4.2.1 through Basic 4.2.4

## DELETE COMMANDS (4.5.5)

DELETE commands delete components from the data base definition.

```
        ┌STRING     ┐  <c>, <c>,...<c>:
 DELETE │COMPONENT  │  <c1> THROUGH <c2>:
        │FUNCTION   │  <c>
        └           ┘
```

where:

<c> = user component number with leading C

## ENABLE/DISABLE EXECUTION COMMANDS (4.5.6)

The ENABLE EXECUTION and DISABLE EXECUTION commands are used to edit-only and selectively edit local define module commands.

```
ENABLE EXECUTION:          (default)
DISABLE EXECUTION:
```

## STOP AFTER SCAN COMMANDS (4.5.7)

STOP AFTER SCAN commands let users edit the definition for error conditio and disallow execution of define module commands if any commands are found in error.

```
STOP AFTER SCAN IF ERRORS OCCUR:
```

# LOADING PROCESS 4

The loading process has three general functions:

1. To enter large volumes of data into a data base initially and at any later time during the life of the data base.

2. To edit the user's raw data by checking the data values and data structure against the user's definition of the data base.

3. To give the user a statistical report at the end of a successful load.

## INPUT VALUE STRING DEFINITION EXAMPLE

```
1* GOOD LIFE INSURANCE CO. *2* L.G.OGDEN* 4* SAN FRANCISCO*

5* CALIFORNIA*

END*
```

## ENTRY TERMINATOR FORMAT (5.2.3)

The end of a logical entry is signaled by the entry terminator word followed by a system separator.

Example:

```
END*
```

The end of the value string is signaled by one more system separator after the last entry terminator.

Example:

```
END *  *
END    **
END**
END  *  *
```

## COMMANDS RELATED TO THE LOADING PROCESS (5.5)

SEPARATOR IS <separator symbol>:

ENTRY TERMINATOR IS <terminator word>:

DATA FILE IS <file name>:

ENABLE EXECUTION:

DISABLE EXECUTION:

STOP IF (Section 6.1.6 BASIC)

LOAD:

## OUTPUT FROM THE LOADING PROCESS (5.6)

An example of the report given when the loading process is complete is shown below.

```
LOAD:
       08:44:45 - BEGIN LOADING -
       08:44:46 - SCAN COMPLETED -
       08:44:46 - DATA SET POINTERS CREATED -
       08:44:46 - DATA SETS UPDATE COMPLETED -
       08:44:46 - BEGIN KEYED VALUE FILE SORT -
       08:44:47 - KEYED VALUE FILE SORT COMPLETED -
       08:44:48 - LOADING COMPLETED -
DATA BASE SIZE AFTER LOAD = 22680 CHARACTERS -
```

## ERROR HANDLING (5.7)

If one or more errors are found in a single logical entry, that entire logical entry is excluded from entering the data base.

# ACCESS MODULE 5

## DESCRIBE COMMANDS (6.1.1)

(1) DESCRIBE:
(2) DESCRIBE <component number>:
(3) DESCRIBE <component number> THROUGH <component number>:
(4) DESCRIBE <component number> THRU <component number>:
(5) DESCRIBE $\begin{bmatrix}\text{STRINGS}\\ \text{FUNCTIONS}\end{bmatrix}$ :[1]

where:

<component number> = a letter C followed by an integer component number, e.g., C4, C54, etc.

## LOAD COMMAND (6.1.2)

LOAD:

## STRING CONCEPT (6.1.3)

### SIMPLE STRING

...<system separator> <string reference> <system separator>...

### EXTENDED STRING

...<system separator> <string reference>(<argument$_1$>,<argument$_2$>,... <argument$_{16}$>)

where:

<system separator> = default system separator is the asterisk (*)

<string reference> = string name or string number, e.g., C45

<argument $_{1-16}$> = any string of alphanumeric characters or even another string (it must be a simple string -- one without arguments) enclosed in parentheses. Argument specifications are treated as TEXT-type material concerning use of blanks. Arguments may number up to 16, each separated by commas

[1] The DESCRIBE FUNCTIONS command may only be used in the Immediate Access Feature.

Example:

```
USER,ABCD:
DATA BASE NAME IS SAMMY:
DEFINE:
600*XYZ (STRING/PRINT C1, PRINT C2 WHERE ANY OF (C1 EQ *1**
      C2 EXISTS, C4 NE *2**):/):
ACCESS:
QUEUE:
*C600 (ABC, LEFTOVERSHOE)
```

## CLEAR COMMANDS (6.1.4)

|  |  |
|---|---|
| (1) CLEAR AUTOMATICALLY: | SCOPE default |
| (2) CLEAR: | KRONOS default |

## ENABLE/DISABLE EXECUTION COMMANDS (6.1.5)

|  |  |
|---|---|
| ENABLE EXECUTION: | (default) |
| DISABLE EXECUTION: |  |

## STOP IF COMMANDS (6.1.6)

(1) STOP IMMEDIATELY IF $\left\{\begin{matrix}\text{<n>} \\ \text{ANY}\end{matrix}\right\}$ $\left\{\text{UPDATE}\right\}$ COMMANDS ARE REJECT

(2) STOP AFTER SCAN IF $\left\{\begin{matrix}\text{<n>} \\ \text{AND}\end{matrix}\right\}$ $\left\{\text{UPDATE}\right\}$ COMMANDS ARE REJECTE

where:

<n> = any integer number greater than zero

$\left\{ \quad \right\}$ = optional command components

## CONTINUE COMMANDS (6.1.7)

(1) CONTINUE IF {UPDATE} COMMANDS ARE REJECTED:
(2) CONTINUE AFTER SCAN IF {UPDATE} COMMANDS ARE REJECTED:

where:

{UPDATE} = an optional command specification

## RELOAD COMMANDS (6.1.8)

(1) RELOAD:
(2) RELOAD WHERE <where clause>:
(3) RELOAD ORDERED BY <ordering clause> WHERE <where clause>:

where:

<where clause> = a series of legal WHERE clause conditions (see Section IA3. 8)

<ordering clause> = see Section IA3.16

## REORGANIZE COMMAND (6.1.9)

REORGANIZE <n>:

where:

<n> = an integer number (0, 1, or 2)

## QUEUE PROCESSING (6.2)

In queue processing, the entire command stream, which begins with the command, QUEUE, and ends with the command, TERMINATE, is read before any retrieval or updating occurs.

### COMMAND SCAN

Each command is scanned for syntactic errors, one command at a time, until the TERMINATE command is read, at which time command execution takes place. Each command or iteration of a command is assigned a sequence number. The sequence numbers assigned to PRINT commands are the only ones ever displayed. They are displayed with the command echo and then later with the output. If a command has errors, a diagnostic is printed on the message file; the command is not executed but the run continues unless the user has specified that the run abort because of user errors.

### COMMAND EXECUTION

Once the command stream has been completely examined, all the WHERE clauses are processed in parallel. After the selected data sets are found, operations on the data base are carried out in the following steps in the order shown:

1. ALL PRINT TREE operations are executed.
2. ALL REMOVE TREE operations are executed.
3. ALL APPEND TREE operations are executed in the order in which they occur in the command stream.
4. All other operations are executed in the order in which they occur in the command stream and in the order in which they occur within a single command, subject to the testing of any associated IF clauses.

All operations, (except PRINT TREE) which would otherwise have affected a dat set, are ignored if the data set is part of a tree which has been removed in step 2 as the result of a REMOVE TREE operation.

### OUTPUT

The output data from PRINT operations are sorted and grouped by command sequence-number before being printed on the user Report File. No output is given as a direct result of update operations, per se.

## QUEUE PROCESSING COMMANDS (6.3)

All available commands may be summarized as fitting into one of the following semantic patterns:

- <action clause> WHERE <where clause>:
- IF <if clause> THEN <action clause> WHERE <where clause>:
- IF <if clause> THEN <action clause> ELSE <action clause> WHERE <where clause>:

- *DATA*
- REPEAT pseudo-command
- TERMINATE:

where:

```
<action clause> = (1) element operation[1]

                      PRINT <element>
                      REMOVE <element>

                      ⎡ADD   ⎤
                      ⎢CHANGE⎥  <element> EQ <value> <system
                      ⎣ASSIGN⎦  separator>

                  (2) group operation[1]

                      PRINT <repeating group>
                      REMOVE <repeating group>

                      ⎡ADD   ⎤
                      ⎢CHANGE⎥  <repeating group> EQ <value string>
                      ⎣ASSIGN⎦

                  (3) tree operation[2]

                      PRINT TREE <repeating group>
                      REMOVE TREE <repeating group>
                      APPEND TREE <repeating group> EQ <value
                                  string>
```

where:

```
<where clause> = (1) where condition

                     <element> EXISTS
                     <element> ⎡EQ⎤ <value> <system separator>
                               ⎢NE⎥
                               ⎢GT⎥
                               ⎢GE⎥
                               ⎢LT⎥
                               ⎣LE⎦

                     <repeating group> HAS <where condition>

                 (2) where sub-expression

                     <where condition>
                     ANY {<count>} OF (<where condition1>,<where
                     condition2>...)
                     ALL OF (<where condition1>,<where condition2>
                     ...)
```

[1] Element and group operation statements may be specified in the same action clause, set off by commas, where the repeating group implicitly specified by the element(s) and the repeating group explicitly specified by the group(s) are one and the same.

[2] Multiple element and group operations may not be specified for tree operations.

(3) where expression

<where sub-expression>
ANY {<count>} OF (<where sub-expression$_1$>, <sub-expression$_2$>...)

ALL OF (<where sub-expression$_1$>,<where sub-expression$_2$>...)

where:

<if clause> = (1) if condition

<element> EXISTS
<element> FAILS
<element> $\begin{bmatrix}\text{EQ}\\ \text{NE}\\ \text{GT}\\ \text{GE}\\ \text{LT}\\ \text{LE}\end{bmatrix}$ <value> <system separator>

<element> <system separator> $\begin{bmatrix}\text{EQ}\\ \text{NE}\\ \text{GT}\\ \text{GE}\\ \text{LT}\\ \text{LE}\end{bmatrix}$ <element> <system separator>

(2) if sub-expression

<if condition>
ANY {<count>} OF (<if condition$_1$>,<if condition$_2$>...)

ALL OF (<if condition$_1$>,<if condition$_2$>...)

(3) if expression

<if sub-expression>
ANY {<count>} OF (<if sub-expression$_1$>, <if sub-expression$_2$>...)

ALL OF (<if sub-expression$_1$>,<if sub-expression$_2$>...)

## QUEUE COMMAND (6.3.1)

QUEUE:

## PRINT AND PRINT TREE COMMANDS (6.3.2)

(1) PRINT <element> WHERE <where clause>:
(2) PRINT <repeating group> WHERE <where clause>:
(3) PRINT TREE <repeating group> WHERE <where clause>:

where:

| | |
|---|---|
| <element> | = element name or "C" number, e.g., C45 |
| <repeating group> | = repeating group name or "C" number |
| <where clause> | = any number of legal WHERE clause conditions |

Example:

```
PRINT C13, PRINT C14 WHERE C13 EQ GENERAL MOTORS*:
```

## REMOVE AND REMOVE TREE COMMANDS (6.3.3)

(1) REMOVE <element> WHERE <where clause>:
(2) REMOVE <repeating group> WHERE <where clause>:
(3) REMOVE TREE <repeating group> WHERE <where clause>:

where:

| | |
|---|---|
| <element> | = element name or "C" number, e.g., C45 |
| <repeating group> | = repeating group name or "C" number |
| <where clause> | = series of legal WHERE clause conditions |

Examples:

```
REMOVE C2 WHERE C2 EQ J.B. WISER*:
REMOVE TREE C8 WHERE C9 EQ INCOME*:
```

## ADD COMMANDS (6.3.4)

(1) ADD <element> EQ <value> WHERE <where clause>:
(2) ADD <repeating group> EQ <value string> WHERE <where clause>:

where:

| | |
|---|---|
| <element> | = element name or "C" number, e.g., C45 |
| <repeating group> | = repeating group name or "C" number |
| <value> | = character string terminated by system separator |
| <value string> | = system separator following each value and each component identification and terminated by terminator and system separator, e.g., 1*XXX*2XYYY* END* |
| <where clause> | = series of legal WHERE clause conditions |

Example:

```
ADD C14 EQ UPJ* WHERE C13 EQ UPJOHN*:
```

## CHANGE COMMANDS (6.3.5)

(1) CHANGE <element> EQ <value> WHERE <where clause>:
(2) CHANGE <repeating group> EQ <value string> WHERE <where clause>:

where:

| | |
|---|---|
| <element> = | element name or "C" number, e.g., C45 |
| <repeating group> = | repeating group name or "C" number |
| <value> = | character string terminated by system separator |
| <value string> = | system separator following each value and each component identification and terminated by terminator and system separator, e.g., 1*XXX*2* YYY*END* |
| <where clause> = | series of legal WHERE clause conditions |

Examples:

```
CHANGE MANAGER EQ B.J. DILLARD* WHERE C9 EQ INCOME*:
CHANGE STOCKS EQ 14*REX* 15* OTC* END* WHERE C14 EQ UPJ*:
```

## ASSIGN COMMANDS (6.3.6)

(1) ASSIGN <element> EQ <value> WHERE <where clause>:
(2) ASSIGN <repeating group> EQ <value string> WHERE <where clause>:

where:

| | |
|---|---|
| <element> = | element name or "C" number, e.g., C45 |
| <repeating group> = | repeating group name or "C" number |
| <value> = | character string terminated by system separator |
| <value string> = | system separator following each value and each component identification and terminated by terminator and system separator, e.g., 1*XXX*2*YYY*END* |
| <where clause> = | series of legal WHERE clause conditions. |

Example:

```
ASSIGN DATE EQ 07/15/71* WHERE C1 EXISTS:
```

## APPEND TREE COMMANDS (6.3.7)

(1) APPEND TREE <repeating group> EQ <value string> WHERE <where clause>:
(2) APPEND TREE $\begin{bmatrix}\text{ENTRY EQ}\\ \text{CO}\end{bmatrix}$ <value string>

where:

<repeating group> = repeating group name or "C" number other than ENTRY or CO

<value string> = system separator following each value and each component identification and terminated by terminator and system separator, e.g., 1*XXX*2*YYY*END*

<where clause> = series of legal WHERE clause conditions

Example:

```
APPEND TREE ENTRY EQ 1*GOOD TIME CO. *END*:
```

## ACTION CLAUSES (6.3.8)

(1) <action clause>:
(2) <action clause> WHERE <where clause>:
(3) IF <if clause> THEN <action clause> WHERE <where clause>:
(4) IF <if clause> THEN <action clause> ELSE <action clause> WHERE <where clause>:

where:

<if clause> = see Section Basic 6.3.11

<where clause> = see Section Basic 6.3.9

<action clause> = specific commands falling into three operations as outlined below. All commands have individual sections containing a complete presentation of each.

(a) element operation, or element operation list (multiple operations may be listed, separated by commas, if all elements are members of the same repeating group)

```
PRINT <element>
REMOVE <element>

⎡ADD     ⎤
⎢CHANGE  ⎥   <element> EQ <value>
⎣ASSIGN  ⎦
```

(b) group operation

```
PRINT <repeating group>
REMOVE <repeating group>

⎡ADD     ⎤
⎢CHANGE  ⎥   <repeating group> EQ <value string>
⎣ASSIGN  ⎦
```

(c) tree operation

```
PRINT TREE <repeating group>
REMOVE TREE <repeating group>
APPEND TREE <repeating group> EQ <value
string>
```

## WHERE CLAUSES (6.3.9)

(1) <action clause> WHERE <where clause>:
(2) IF <if clause> THEN <action clause> WHERE <where clause>:
(3) IF <if clause> THEN <action clause> ELSE <action clause> WHERE <where clause>:

where:

<if clause> = see Section 6.3.11

<action clause> = see Section 6.3.8

<where clause> = any one of the following statements:

(a) a where condition

<element> EXISTS

<element> [EQ | NE | GT | GE | LT | LE] <value>

<repeating group> HAS <where condition>

(b) a where sub-expression

<where condition>
ANY {<count>} OF (<where condition$_1$>, <where condition$_2$>...)
ALL OF (<where sub-expression$_1$>, <where sub-expression$_2$>...)

Examples:

WHERE C1 EXISTS:
WHERE ANY 2 OF (C1 EXISTS, C9 EQ TRUST*, C13 EQ GOODYEAR*):

## IF CLAUSES (6.3.11)

(1) IF <if clause> THEN <action clause> WHERE <where clause>:
(2) IF <if clause> THEN <action clause> ELSE <action clause> WHERE <where clause>:

where:

<action clause> = see Section Basic 6.3.8

<where clause> = see Section Basic 6.3.9

<if clause> = any one of the following statements

(a) if condition

<element> [EXISTS | FAILS]

<element>

<element> <system separator> [EQ / NE / LT / LE / GT / GE] [<value> <system separator; / <element> <system separator;]

(b) <u>if sub-expression</u>

<if condition>

ANY {<count>} OF (<if condition$_1$>,<if condition$_2$>,...)

ALL OF (<if sub-expression$_1$>,<if sub-expression$_2$>,...)

## UNARY OPERATORS (6.3.12)

(1) ...WHERE <element> EXISTS:

(2) IF <element> [EXISTS / FAILS] ...

where:

<element> ≐ element name or "C" number

## BINARY OPERATORS (6.3.13)

(1) ...WHERE <element> <binary operator> <specific value>:

(2) IF <element> <binary operator> <specific value>...

(3) IF <element> <system separator> <binary operator> <element> <system separator>...

where:

<element> = element name or "C" number

<binary operator> = [EQ / NE / LT / LE / GT / GE]

<specific value> = a value followed by a system separator

<system separator> = system separator currently is use (default separator is *)

## HAS OPERATOR (6.3.14)

... WHERE <repeating group> [HAS / HAVE / HAVING] <condition>:

where:

<repeating group> = repeating group name or "C" number

<condition> = any of the WHERE clause conditional statements

## BOOLEAN OPERATORS (6.3.15)

(1) ...ANY {<count>} OF (<conditions>...)
(2) ...ALL OF (<conditions>...)

where:

<count> = n where $1 \leq n \leq 7$; default condition sets <count> = 1

<conditions> = any legal WHERE clause and IF clause conditions (see Sections Basic 6.3.9 and 6.3.11)

## DATA FLAG (6.3.16)

<system separator> DATA <system separator> (for example, *DATA*)

where:

<system separator> = by default, the asterisk (*)

Example:

```
IF C9 EQ *DATA* THEN ASSIGN C8 EQ *DATA* WHERE C1 EQ *DATA*:
```

## REPEAT PSEUDO-COMMANDS (6.3.17)

(1) REPEAT <special character> <command stream> <special character>:
(2) REPEAT <special character> <command stream> <special character> <n> TIMES:

where:

<special character> = any machine allowable character not appearing in the enclosed command stream

<command stream> = character string not exceeding 250 characters, including one or more complete commands

<n> = an integer number

Example:

```
REPEAT/APPEND TREE ENTRY EQ *DATA*:/:
```

## TERMINATE COMMANDS (6.3.18)

(1) TERMINATE:
(2) TERMINATE/PRINT:
(3) TERMINATE/UNLOAD:

# IMMEDIATE ACCESS FEATURE

## DESCRIBE COMMANDS (3.1)

(1) DESCRIBE:
(2) DESCRIBE <component number>:
(3) DESCRIBE <component number> THROUGH <component number>:
(4) DESCRIBE <component number> THRU <component number>:
(5) DESCRIBE $\begin{bmatrix}\text{STRINGS}\\\text{FUNCTIONS}\end{bmatrix}$ :

where:

<component number> = a letter C followed by an integer component number, e.g., C4, C54, etc.

## TALLY COMMANDS (3.2)

(1) TALLY <element list>:
(2) TALLY/EACH/ <element list>:
(3) TALLY/ALL/ <element list>:

where:

<element list> = one or more element names or numbers separated by commas

Examples:

```
TALLY/EACH/C26:
TALLY/ALL/C26:
```

## PRINT COMMANDS (3.3)

(1) PRINT <print clause>:
(2) PRINT <print clause> <ordering clause> WHERE <conditions exist>:

where:

<print clause> = <option>,<by clause(s)>, or <option list(s)>

<ordering clause> = an optional portion of the command used or order the output (see Section IA 3.8)

<conditions exist> = series of legal WHERE clause conditions

<option list> = one or more format option statements separated by commas, the entire list set off by slashes. (See Section IA 3.4)

<by clause(s)> = one or more BY clauses separated by commas, each one having its own object list. (See Section IA 3.10)

<object list(s)> = KEY or NON-KEY element references (name or number), repeating group references, in-line and user-defined function references, and system functions

Examples:

```
PRINT C1, C2, C28:
PRINT BY ENTRY, C1, COUNT C9, COUNT C13, BY C8, C9, C13:
```

## FORMAT STATEMENTS (3.4)

(1) PRINT/ <option list>/:
(2) PRINT/ <option list>/<print clause>...:
(3) LIST/ <option list>, TITLE <title specifications>/<list clause>...:
(4) UNLOAD/ <option list>/<unload clause>...:

where:

| | |
|---|---|
| <option list> | = one or more format option statements separated by commas |
| / (slash) | = the mandatory symbol used to set off the option list, when used |

The format options available to the user are as follows:

Format Statements

| | | | |
|---|---|---|---|
| (1) | SINGLE SPACE<br>DOUBLE SPACE | (5) | NULL SUPPRESS<br>NULL |
| (2) | INDENT<br>BLOCK | (6) | TREE<br>GROUP |
| (3) | NUMBER<br>NAME | (7) | REPEAT<br>REPEAT SUPPRESS |
| (4) | STUB<br>STUB SUPPRESS | (8) | ZERO SUPPRESS<br>ZERO |

Example:

```
PRINT/NULL/C1, C2, C3 WHERE C5 EQ A:
```

## SYSTEM FUNCTIONS (3.5)

... <function name> <function object>

where:

| | |
|---|---|
| <function name> | = any of the six functions MIN, MAX, AVG, SUM, COUNT, SIGMA |
| <function object> | = any element, repeating group, in-line or user-defined function (including nested system functions) |

Example:

```
PRINT MAX C1, MIN C1, COUNT C1:
```

## USER-DEFINED FUNCTIONS (3.6)

```
(1) PRINT...*<function reference>*...:
(2) PRINT...*<function reference>*...WHERE...:
(3) LIST/...*<function reference>*...WHERE...:
(4) UNLOAD...*<function reference>*...WHERE...:
   ┌ADD    ┐
(5)│CHANGE │  <element> = ...*<function reference>*...WHERE...:
   │ASSIGN ┘        ┌HIGH┐
(6)└...ORDERED BY   └LOW ┘  *<function reference>*...:
```

Example:

```
PRINT *C101* WHERE C1 EQ SMITH:
```

## IN-LINE FUNCTIONS (3.7)

```
(1) PRINT...(<in-line function>)...:
(2) PRINT...(<in-line function>)...WHERE...:
(3) LIST/.../...(<in-line function>)...WHERE...:
(4) UNLOAD...(<in-line function>)...WHERE...:
   ┌ADD    ┐
(5)│CHANGE │  <element> = ...(<in-line function>)...WHERE...:
   └ASSIGN ┘
                    ┌HIGH┐
(6) ...ORDERED BY   └LOW ┘  (<in-line function>)...
```

Example:

```
PRINT SUM(C24-C30) WHERE C13 EQ AMERICAN CYANAMID:
```

## ORDERING STATEMENTS (3.8)

```
                              ┌ORDERED BY┐
(1) PRINT <object list>,      └OB        ┘  <ordering clause> WHERE...:

                              ┌ORDERED BY┐
(2) LIST/.../ <object list>,  └OB        ┘  <ordering clause> WHERE...:

                              ┌ORDERED BY┐
(3) UNLOAD <object list>,     └OB        ┘  <ordering clause> WHERE...:
```

where:

```
      <object list> = KEY and NON-KEY element references (name or
                      number), repeating group references, in-line and
                      user-defined function references, and system func-
                      tions

                      ┌HIGH┐             ┌HIGH┐
  <ordering clause> = └LOW ┘ <item 1>,   └LOW ┘ <item 2>...

           <item n> = any one of the <object list> items defined above
```

The HIGH and LOW directives are sort words with the following meanings:

LOW = default condition; low-to-high (ascending order); i.e., 1, 2, 3, 4, ... or A, B, C, D ...

HIGH = high-to-low (descending order); i.e., 12, 11, 10, 9, ... or V, U, T, S, ...

## LIST COMMANDS (3.9)

(1) LIST/TITLE <title specifications>/:
(2) LIST/TITLE <title specifications>/<list clause>:
(3) LIST/ <option list>, TITLE <title specifications>/<list clause>, <ordering clause> WHERE <conditions exist>:

where:

| | |
|---|---|
| <title specifications> | = <page heading> <page footing> <column specifications> |
| <option list> | = one or more format option statements separated by commas (see Section IA 3.4) |
| <list clause> | = <by clauses> and <object lists> or just <object lists> |
| <ordering clause> | = ordering option; see Section IA 3.8 |
| <conditions exist> | = series of legal WHERE clause conditions |
| <by clauses> | = one or more BY clauses separated by commas, each having its own object list (see Section IA 3.10) |
| <object lists> | = KEY or NON-KEY element references (name or number), in-line and user-defined function references, and system functions |

Page headings and footings are specified immediately following any format options desired.

| | |
|---|---|
| <page heading> | = D(NN) <test>, |
| <page footing> | = F(MM) <test>, |
| D | = date |
| NN | = beginning column for the text |

The <column specification> may be made up of a single specification or several specifications; specifications are separated by commas. A specification may include two fields of format identification: <type> and <title>. The <type> field is not mandatory. However, if it is not supplied, the <title> field must be specified. The <type> field is identified as:

$$\text{<type>} = \begin{bmatrix} \text{L} \\ \text{R} \\ \text{B} \end{bmatrix} \text{(<integer number>)}$$

where:

L = <title> left-justified
R = <title> right-justified
B = column to be blank and <title>, if any, left-justified

The <title> field is identified as:

<title> = (a) <string of characters>
(b) <string of characters> + <string of characters>
(c) <string of characters> + <string of characters> + <string of characters)
(d) + <string of characters>
(e) + <string of characters> +

(f) ++ <string of characters>
(g) + <string of characters> + <string of characters>
(h) <string of characters> + <string of characters> +
(i) <string of characters> ++ <string of characters>

where:

| | |
|---|---|
| <string of characters> = | any string of characters which will become the column heading, limited in number only by the output device in use, e.g., a maximum number of characters on a teletype equals 69; this maximum is 129 on a printer |
| + = | normally indicates title wraparound, three lines allowable per column; also controls which of the three lines will be blank |

Example:

```
LIST/TITLE L(10)NAME OF +ΔΔSTOCK, ΔΔLATEST + EARNINGS,
+ΔΔΔΔDATEΔΔΔΔ, B(9)SHARES + DESIRED, B(7)PRICE + DESIRED,
R(9)CURRENT + PRICE ΔΔ/C13,C19,C7,C24 WHERE PORTFOLIO
NAME EQ INCOME:
```

## BY CLAUSES (3.10)

(1) PRINT[1] <object list>:
(2) PRINT <object list>,<by clause>,<by clause>:
(3) PRINT <by clause> WHERE <conditions exist>:
(4) PRINT <system function> <function object> <by phrase>:
(5) PRINT <object list>,<by clause>,<system function> <function object> <by phrase> WHERE <condition exist>:

where:

| | |
|---|---|
| <object list> = | any item legally referenced in a <print clause>, including items from disjointed data sets |
| <by clause> = | <by phrase>,<object list> |
| <by phrase> = | BY followed by an RG name or number, including "DATA BASE" |
| <conditions exist> = | any of the legal WHERE clause conditions |
| <system function> = | any of the six built-in system functions (SUM, COUNT, etc.) |
| <function object> = | any component that may be legally operated upon by the associated function |

Example:

```
PRINT BY ENTRY, C1, COUNT C1, C12, COUNT C13:
```

[1] Wherever appropriate, the BY clause may be issued in the PRINT, LIST, and UNLOAD commands.

## WHERE CLAUSES (3.11)

...WHERE <conditions exist>

where:

<conditions exist> = any number of legal WHERE clause conditions as defined by the following statements:

(1) Unary operator condition

```
<element> ┌EXISTS┐
          └FAILS ┘
```

(2) Binary operator condition

```
<element> ┌EQ┐ <value>
          │NE│
          │LT│
          │LE│
          │GT│
          └GE┘
```

(3) Ternary operator condition

```
<element> ┌SPANS┐ <value><system separato
          │NE   │ <value>
          └EQ   ┘
```

(4) Normalized condition

<repeating group> HAS <condition>

Example:

```
PRINT PORTFOLIO NAME, MANAGER WHERE PORTFOLIO NAME
EXISTS:
```

## UNARY OPERATORS: EXISTS, FAILS (3.12)

```
...WHERE <element> ┌EXIST, EXISTS, or EXISTING┐ :
                   └FAIL, FAILS, or FAILING   ┘
```

where:

<element> = element name or "C" number

Example:

```
PRINT C1 WHERE C5 FAILS
```

## BINARY OPERATORS: EQ,NE,LT,LE,GT,GE (3.13)

...WHERE <element> <binary operator> <specific value>:

where:

<element> = element name or "C" number

| | | |
|---|---|---|
| <binary operator> = | EQ | equal |
| | NE | not equal |
| | LT | less than |
| | LE | less than or equal to |
| | GT | greater than |
| | GE | greater than or equal to |

<specific value> = a value

Example:

```
PRINT C1 WHERE C2 GE 12:
```

## BOOLEAN OPERATORS: AND,OR,NOT (3.14)

(1) ... WHERE <condition$_1$> $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$ <condition$_2$>:

(2) ... WHERE NOT <condition$_1$>:

where:

<condition$_n$> = any one of the many WHERE clause conditional statements

Example:

```
PRINT C1 WHERE C5 EQ 10 OR C6 LT 9:
```

## AT OPERATORS (3.15)

(1) ... WHERE ... <condition> AT <n>:
(2) ... WHERE ... (<boolean combination>) AT <n>... :
(3) ... WHERE ... (<condition> AT <n>) AT 0:

where:

<condition> = any of the legal WHERE clause conditions

<n> = positive integer number indicating logical positional relationship in the data base tree structure

<boolean operator> = two or more conditions within the same data set, with or without AT clauses, linked by Boolean operators

Example:

```
PRINT C25 WHERE BLOCK NUMBER EQ 1000 AT 0.
```

## HAS OPERATORS (3.16)

...WHERE <repeating group> $\begin{bmatrix}\text{HAS}\\ \text{HAVE}\\ \text{HAVING}\end{bmatrix}$ <condition>

where:
<repeating group> = repeating group name or "C"

<condition> = any of the WHERE clause condition statements, nested or unnested

Example:

WHERE ENTRY HAS NAME OF STOCK EQ UPJOHN:

## STRING CONCEPTS (3.17)

### SIMPLE STRING

... <system separator> <string reference> <system separator>...

### EXTENDED STRING

... <system separator> <string reference> (<$\text{argument}_1$>,<$\text{argument}_2$>...)

where:
<system separator> = default system separator is the *.

<string reference> = string name or string number (e.g., C45).

<arguments> = any string of alphanumeric characters or even another string (it must be a simple string, without arguments) enclosed in parentheses. Argument specifications are treated as TEXT-type material with respect to use of blanks.

## DITTO OPERATORS (3.18)

(1) DITTO:
(2) DITTO $\begin{bmatrix}\text{WHERE}\\ \text{BEFORE}\\ \text{AFTER}\end{bmatrix}$ <conditions exist>:

where:
<conditions exist> = any of the legal WHERE clause conditions

## SAME OPERATORS (3.19)

...WHERE SAME:

...WHERE SAME $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$ <conditions exist>:

...WHERE <conditions exist> $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$ SAME...:

...WHERE (<$\text{condition}_1$> $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$ SAME) $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$ <$\text{condition}_2$> $\begin{bmatrix}\text{AND}\\ \text{OR}\end{bmatrix}$
SAME...:

...WHERE <etc.>

## SAME DECLARATION COMMAND (3.20)

```
SAME IS [STATIC ] :
        [DYNAMIC]
```

where:

DYNAMIC = the default processing mode of SAME

The SAME declaration command changes the processing mode of the SAME operator in the following way:

STATIC = SAME refers to the last WHERE clause that did not have a SAME operator

DYNAMIC = SAME refers to the last WHERE clause whether or not it had a SAME operator

## LIMIT COMMANDS (3.21)

```
(1) LIMIT <n1>:

(2) LIMIT <n1> [<n2>                ] :
               [,<n2>/<disposition> ]

(3) LIMIT <n1> [<n2>                ] [<n3>              ] :
               [,<n2>/<disposition> ] [,<n3>/<filename>  ]

(4) END LIMIT:
```

where:

<n1>,<n2>, or <n3> = 0 or any positive integer; if equal to 0, then limit is infinite

<disposition> = [CANCEL / TRUNCATE] default is CANCEL

<filename> = filename recognizable to resident operating system

NOTE: <n1> and <n3> must be less than or equal to <n2> except when <n2> = 0.

## UNLOAD COMMANDS (3.22)

(1) UNLOAD:
(2) UNLOAD <unload clause>:
(3) UNLOAD/<option list>/<unload clause>,<ordering clause> WHERE <conditions exist>:

where:

<option list> = see Section IA 3.4

<unload clause> = <by clause> and <object list> or just <object list>

<ordering clause> = ordering option (see Section IA 3.8)

<conditions exist> = series of legal WHERE clause conditions

<by clause> = see Section IA 3.10.

<object lists> = KEY or NON-KEY element references (name or number), repeating group references, in-line and user-defined function references, and system references.

## IMMEDIATE ACCESS UPDATE COMMANDS

### ADD COMMANDS (4.1)

(1) ADD <element>[1] $\left[\begin{array}{l}\text{EQ <value>}\\ \text{= <function>}\end{array}\right]$ WHERE <conditions exist>:

(2) ADD <repeating group>[1] EQ <value string> WHERE <conditions exist>:

where:

| | |
|---|---|
| <element> | = element name or "C" number |
| <value> | = character string terminated by system separator |
| <function> | = any legal in-line, user-defined or system function; the function may be enclosed in a BY clause |
| <conditions exist> | = series of legal WHERE clause conditions |
| <repeating group> | = repeating group name or "C" |
| <value string> | = system separator following each value and each component identification and terminated by the entry terminator, e.g., 1*XXX*2*YYY*END* |

Examples:

```
ADD C5 EQ 07/15/71* WHERE C1 EQ GOOD CO.:
ADD C21 = (0.1*34.50) WHERE C13 EQ GENERAL MOTORS:
```

### CHANGE COMMANDS (4.2)

(1) CHANGE <element>[1] $\left[\begin{array}{l}\text{EQ <value>}\\ \text{= <function>}\end{array}\right]$ WHERE <conditions exist>:

(2) CHANGE <repeating group>[1] EQ <value string> WHERE <conditions exist>

where:

| | |
|---|---|
| <element> | = element name or "C" number |
| <value> | = character string terminated by system separator |
| <function> | = any legal in-line, user-defined or system function; the function may be enclosed in a BY clause |

[1] A trace notation may be entered following the element or repeating group. For an example of trace notation, see Section IA 4.8.

<conditions exist> = series of legal WHERE clause conditions

<repeating group> = repeating group name or "C" number

<value string> = system separator following each value and each component identification and terminated by the entry terminator, e.g., 1*XXX*2*YYY*END*

Examples:

```
CHANGE C25 EQ ABC* WHERE C9 EQ INCOME:
CHANGE C29 = (C29+500) WHERE C13 EXISTS:
```

## REMOVE COMMANDS (4.3)

(1) REMOVE <element>[1] WHERE <conditions exist>:
(2) REMOVE <repeating group>[1] WHERE <conditions exist>:

where:

<element> = element name or "C" number

<conditions exist> = series of legal WHERE clause conditions

<repeating group> = repeating group name or "C" number

Example:

```
REMOVE MANAGER WHERE MANAGER EQ DICK ESSELSEN:
```

## ASSIGN COMMANDS (4.4)

(1) ASSIGN <element>[1] $\left[\begin{array}{l}\text{EQ <value>}\\ \text{= <function>}\end{array}\right]$ WHERE <conditions exist>:
(2) ASSIGN <repeating group>[1] EQ <value string> WHERE <conditions exist>:

where:

<element> = element name or "C" number

<value> = character string terminated by system separator

<function> = any legal in-line, user-defined or system function; the function may be enclosed in a BY clause

<conditions exist> = series of legal WHERE clause conditions

<repeating group> = repeating group name or "C" number

<value string> = system separator following each value and each component identification and terminated by the entry terminator, e.g., 1*XXX*2*YYY*END*

Examples:

```
ASSIGN DATE EQ 02/25/71* WHERE C1 EXISTS:
ASSIGN C22 = MAX C22 BY C9 WHERE C13 EQ GOOD CO.:
```

---

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see Section IA 4.8.

## ASSIGN TREE COMMAND (4.6)

ASSIGN TREE <repeating group>[1] EQ <value string> WHERE <conditions exist>

where:

<repeating group> = repeating group name or "C" number

<value string> = system separator following each value and each component identification and terminated by the entry terminator, e.g., 1*XXX*2*YYY*END*

<conditions exist> = series of legal WHERE clause conditions

Example:

```
ASSIGN TREE PORTFOLIOS EQ 9* PRIVATE* 10* XYZ* 12* 13*
CENTRAL* END* WHERE C9 EQ STOCKS:
```

## INSERT TREE COMMANDS (4.7)

$$\text{(1) INSERT TREE} \left\langle \begin{array}{l}\text{repeating}\\ \text{group}\end{array} \right\rangle \text{ EQ } \left\langle \begin{array}{l}\text{value}\\ \text{string}\end{array} \right\rangle \left[ \begin{array}{l}\text{BEFORE}\\ \text{AFTER}\end{array} \right] \left\langle \begin{array}{l}\text{conditions}\\ \text{exist}\end{array} \right\rangle :$$

$$\text{(2) INSERT TREE} \left\langle \begin{array}{l}\text{repeating}\\ \text{group}\end{array} \right\rangle \left\langle \begin{array}{l}\text{partial trace[1]}\\ \text{notation}\end{array} \right\rangle \text{ EQ } \left\langle \begin{array}{l}\text{value}\\ \text{string}\end{array} \right\rangle \text{ WHERE } \left\langle \begin{array}{l}\text{conditions}\\ \text{exist}\end{array} \right\rangle :$$

$$\text{(3) INSERT TREE} \left\langle \begin{array}{l}\text{repeating}\\ \text{group}\end{array} \right\rangle \left\langle \begin{array}{l}\text{full trace[1]}\\ \text{notation}\end{array} \right\rangle \text{ EQ } \left\langle \begin{array}{l}\text{value}\\ \text{string}\end{array} \right\rangle :$$

where:

<repeating group> = repeating group name or "C" number

<value string> = system separator following each value and each component identification and terminated by the entry terminator, e.g., 1*XXX*2*YYY*END*

<conditions exist> = series of legal WHERE clause conditions

$\left\langle \begin{array}{l}\text{full or partial}\\ \text{trace notation}\end{array} \right\rangle$ = a string of one or more integer numbers each preceded by a system separator, e.g., *1 *3 *2 ...

Examples:

```
INSERT TREE C12 *0 EQ 13* ABC CO.* END* WHERE C9 EXISTS:
IF C25 EQ 26*2025*27*BUY*28*02/25/72*END*AFTER C26 EQ 2024:
```

[1] A trace notation may be entered following the component identification. For an explanation of trace notation, see Section 4.8.

## TRACE NOTATIONS (4.8)

### FULL TRACE

A specific example of a full trace is:

REMOVE TREE ENTRY *4:

In this case, the fourth logical entry is removed from the data base.

Another full trace example is:

ASSIGN PORTFOLIOS *2*0 EQ 9* CLIMBER* 11* J. D. GILPEN* END*:

Here, the second PORTFOLIOS data set in the last logical entry is selected. Its contents are removed and replaced with the value string values.

Finally, a third example is:

INSERT TREE ENTRY*0 EQ 1* ABC CONGLOMERATE* 2* J. P. SMITH* 7* 07/15/71* END*:

In this example, a new logical entry with one data set where the ORGANIZATION is the ABC CONGLOMERATE has been added as the last logical entry to the data base.

### PARTIAL TRACE

An example of a partial trace is:

REMOVE C25*1 WHERE C9 EQ PILFER FUND:

Here, the qualified data set is the level 1 data set for PILFER FUND. The selected ancestral data sets are all C12 repeating groups (STOCKS) for PILFER FUND. The trace extends to the first occurrence of C25 repeating group (TRANSACTION) and removes it.

A second example is:

ASSIGN C25*0*2 EQ 26* 3001* 27* BUY* 28* 02/25/70* 29* 5000* 30* 25.00* END* WHERE C9 EQ RESERVE:

In this case, the qualified data set is the level 1 data set for the RESERVE portfolio. The selected ancestral data sets are all C8 (PORTFOLIOS) repeating groups. Since the WHERE clause qualified only one C8 repeating group (RESERVE portfolio), the selected ancestrat data set is the RESERVE portfolio. Following the trace downward, the last transaction entered for the second occurrent of the STOCK repeating group is assigned new vluaes.

## PREVIOUS OPERATOR (4.9)

$$\left[\begin{array}{l}\text{ADD}\\ \text{CHANGE}\\ \text{ASSIGN}\\ \text{ASSIGN TREE}\\ \text{INSERT TREE}\end{array}\right] \left\langle\begin{array}{l}\text{component}\\ \text{identification}\end{array}\right\rangle \text{ EQ PREVIOUS WHERE } \left\langle\begin{array}{l}\text{conditions}\\ \text{exist}\end{array}\right\rangle \ :$$

# CYBERNET SERVICE MACHINE-DEPENDENT INFORMATION 7

CYBERNET Service offers SYSTEM 2000 on CDC 6600 Computer Systems that function under both KRONOS and SCOPE Operating Systems. KRONOS users can perform their SYSTEM 2000 processing in an interactive time-sharing mode while SCOPE users acquire over-the-counter batch and remote batch access.

## INTERACTIVE KRONOS USAGE

The following procedure lets SYSTEM 2000 users log-in to KRONOS and acquire interactive time-sharing:

```
75/04/04.   11.48.14.                     ┐
                                          │<-KRONOS sends basic log-in
EASTERN CYBERNET CENTER SN166 KRONOS      ┘  message

FAMILY: KC <-------------- KRONOS requests family name; user enters
                           appropriate name.
USER NUMBER: ABC1234 <---- KRONOS asks user to enter his user number;
                           user responds by entering his CDC-assigned
PASSWORD    ┐              user number (i.e., ABC1234)
            │<------------ KRONOS asks for user's password
■■■■■■■■■■  ┘

                      ---- KRONOS specifies the terminal number
TERMINAL: 102,TTY <-       assigned for this session

RECOVER/SYSTEM: NULL<----- KRONOS asks what subsystem to process under
                           SYSTEM 2000 users enter NULL
OLD,NEW, OR LIB FILE: LIB<-- KRONOS asks for file status; SYSTEM
                             2000 users enter LIB
FILE NAME: S2KV2 <-------- KRONOS asks for file name; SYSTEM 2000 user
                           enter S2KV2
READY.

-S2KV2<------------------- After READY response, users enter -S2KV2
                           to execute SYSTEM 2000
74/01/14.   11.26.00.  *BEGIN SYSTEM 2000*RELEASE 2.23E┐<-SYSTEM 2000
                                                        │  processing
---                                                     │  can now
   _____________________________________________________┘  begin
?USER,xxxx: <------------- User identifies himself to SYSTEM 2000 by
                           entering his data base password (i.e.,xxxx)
```

Under KRONOS*, the default ARU limit is $512_{10}$. However, users can issue the following standard SYSTEM 2000 command to establish a different, non-standard limit for a terminal session:

ARU LIMIT IS <n>:

where:

<n> = the new ARU limit; this must be an integer less than 32768

*SCOPE users establish their own ARU limits by submitting an appropriate SCOPE control card.

## SPECIAL CONTROL MODULE COMMAND (MDI-48)

The following new control module command (available for both SCOPE and KRONOS users) lets the user attach a data base (from tape) under an account which is different from the account specified in the user's job control card (SCOPE) or log-in user number (KRONOS).

```
ACCOUNT IS <n>:
```

where:

<n> = the account or user number under which the data base is catalogued

However, certain SYSTEM 2000 commands should not be issued once an ACCOUNT IS command is entered. This limitation applies to the following commands:

(1) LOAD <data base name> FROM <tape id>:
(2) NDB IS <data base name>:
(3) KEEP:
(4) REORGANIZE:

# REPORT WRITER FEATURE 8

The section numbers in this chapter refer to the corresponding sections in the SYSTEM 2000 Level 2 Report Writer Feature Supplement manual, publication number 76074700.

## REPORT DEFINITION (2.0)

The general form of a complete report definition is:

```
FOR REPORT <reportname>,                   ─┐
    PHYSICAL PAGE...:                       │
    LOGICAL PAGE...:                        │
    INCLUDE...:                             │
    SUPPRESS...:                            │
    DECLARE...:                             │
        :    (there may be several          │◄──── FOR REPORT
        :    DECLARE commands)              │        Block
    SELECT }RECORD{ IF...:                  │
    ORDER BY...:                            │
        <other report-wide actions>         │
    AT END,                                 │
        <end-of-report actions>            ─┘
    FOR PAGE,                              ─┐
        <page actions>                      │◄──── FOR PAGE
    AT END,                                 │       Block
        <end-of-page actions>              ─┘
    FOR <component>,                       ─┐
        <breakpoint actions>                │◄──── FOR <component>
    AT END,                                 │      Blocks, up to 62
        <end-of-block actions>             ─┘      per Report
    FOR RECORD,                            ─┐
        <breakpoint actions>                │
    AT END,                                 │◄──── FOR RECORD
        <end-of-block actions>              │        Block
    END }REPORT{ :                         ─┘
```

## REPORT RECORDS (2.1)

The concept of "record" and its relationship to the hierarchy of data base components is extremely important for accurate report definition. All data base components named anywhere in the report definition must be members of the same "family"; that is, they must lie along a single vertical path of the data base definition. A report record is the collection of values corresponding to the named components along a single vertical path of the data itself.

## ACTION CONTROL (2.2)

There are two basic ways to control the execution of Report Writer action clauses. The FOR and AT END phrases of individual report blocks specify the domain, or context, in which the actions take place. The IF... THEN... ELSE.. command specifies the conditions which must be met within that domain and designates alternate actions depending on whether these conditions are satisfied or not.

### FOR REPORT Phrase (2.2.1)

FOR REPORT <reportname>,

where: <reportname> = the report identifier. It may be 1 to 8 alphanumeric characters, the first of which must be alphabetic.

### FOR PAGE Phrase (2.2.2)

FOR PAGE,

### FOR <component> Phrase (2.2.3)

FOR <component>,

where: <component> = data base element or RG name or component number as defined in Section Basic 4.2 of the SYSTEM 2000 User Information Manual.

### FOR RECORD Phrase (2.2.4)

FOR RECORD,

### AT END Phrase (2.2.5)

AT END,

## IF...THEN...ELSE (2.2.6)

IF <if-clause> THEN <action clauses> }ELSE <action clauses>{ :

where: <action clauses> = a combination of PRINT, DPRINT, SPRINT, COMPUTE, RESET, SKIP, and CALL clauses (sections RW 2.4.1 through RW 2.4.5) separated by commas.

<if-clause> = any of the following:

(a) <u>if-condition</u>

(1) $\begin{bmatrix}\text{<element>}\\ \text{<derived name>}\end{bmatrix} \begin{bmatrix}\text{EXISTS}\\ \text{FAILS}\end{bmatrix}$

(2) <repeating group> OCCURS

(3) $\begin{bmatrix}\text{<element>}\\ \text{<derived name>}\end{bmatrix}$ <relop><value><separator>

(4) $\begin{bmatrix}\text{<element>}\\ \text{<derived name>}\end{bmatrix}$ <separator> <relop> $\begin{bmatrix}\text{<element>}\\ \text{<derived name>}\end{bmatrix}$ <separator>

where: <element> = element name or "C" number, e.g., C45.

<derived name> = user-specified identifier for a computed value previously defined in a DECLARE command (Section RW 2.3.4). It may not be a type NAME or TEXT value if it is used on the left side of a relational operator.

<repeating group> = repeating group name or "C" number.

<relop> = one of the relational operators EQ, NE, LT, LE, GT, GE.

<value> = alphanumeric character string.

<separator> = system separator currently in use (default is the asterisk).

(b) <u>if-subexpression</u>

<if-condition>

ANY }<count>{ OF (<if-condition$_1$>,<if-condition$_2$>,...<if-condition$_n$>)

ALL OF (<if-condition$_1$>,<if-condition$_2$,...<if-condition$_n$>)

where: <u>n</u> is less than or equal to 7 and <count> is an integer less than or equal to <u>n</u>.

(c) if-expression

```
<if-subexpression>

ANY }<count>{ OF (<if-subexpression_1>,<if-subexpression_2>,...

        <if-subexpression_n>)

ALL OF (<if-subexpression_1>,<if-subexpression_2>,...<if-subexpression_n>)
```

where: n is less than or equal to 7 and <count> is an integer less than or equal to n.

## REPORT-WIDE SPECIFICATIONS (2.3)

A set of commands is available in the FOR REPORT block to define variables, select and sort report records, control printing of detail and summary print lines, and define the dimensions of the report page.

### PHYSICAL PAGE Command (2.3.1)

PHYSICAL PAGE IS <width> BY <length>:

| | |
|---|---|
| where: <width> | = the number of print positions per physical line, up to 180. |
| <length> | = the number of print lines per physical page. If length is 0 (zero), page length is infinite. |

### LOGICAL PAGE Command (2.3.2)

LOGICAL PAGE IS <width> BY <length>:

| | |
|---|---|
| where: <width> | = the number of print positions per logical page. |
| <length> | = the number of lines per logical page. |

Neither <width> nor <length> may exceed its PHYSICAL PAGE counterpart.

### INCLUDE/SUPPRESS Commands (2.3.3)

```
[INCLUDE ]
[SUPPRESS]  DETAIL:

[INCLUDE ]
[SUPPRESS]  SUMMARY:
```

## DECLARE Command (2.3.4)

```
          ⎧ DECIMAL ⎫
DECLARE   ⎨ INTEGER ⎬   <derived name> =
          ⎪ MONEY   ⎪
          ⎩ DATE    ⎭

    ⎡ ⎡CNT ⎤          ⎡<separator><function><separator>⎤ ⎤
    ⎢ ⎢RCNT⎥  {OF}    ⎢(<arithmetic expression>)       ⎥ ⎥
    ⎢ ⎢SUM ⎥          ⎢<component>                     ⎥ ⎥
    ⎢ ⎣RSUM⎦          ⎣<derived name>                  ⎦ ⎥
    ⎢                                                    ⎥
    ⎢ (<arithmetic expression>)                          ⎥
    ⎣ <separator><function><separator>                   ⎦
```

| | |
|---|---|
| where: <derived name> = | user-specified identifier for the calculated value. It may be 1 to 6 alphanumeric characters, the first of which must be alphabetic. Whenever a <derived name> appears to the right of the equals sign, it <u>must</u> have been defined by a previous DECLARE. |
| CNT = | identifies the <derived name> as a counter associated with the named item. |
| RCNT = | the same as CNT, but automatically reset to zero after printing. |
| SUM = | identifies the <derived name> as an accumulator associated with the named item. |
| RSUM = | same as SUM, but automatically reset to null after printing. |
| <separator> = | the system separator. |
| <function> = | component name or number identifying a user-defined function. If it is an extended function, the final <system separator> is replaced by the argument list enclosed in parentheses, i.e., <separator><function>(<argument$_1$>. . . , <argument$_n$>) User-defined functions are discussed in detail in Section Basic 4.2.3 of the <u>SYSTEM 2000 User Information Manual</u>. |
| <arithmetic expression> = | an arithmetic expression composed of type INTEGER, DECIMAL, MONEY, or DATE data base elements, constants, previously declared derived names, and the operators +, -, *, and /. Date constants must be specified in the format mm.dd.yy to avoid confusing a slash with a divide instruction. |
| <component> = | data base element or repeating group name or number - when used with SUM or RSUM, it must be a type DATE, INTEGER, DECIMAL, or MONEY element. |

The term DECLARE may be abbreviated as DE. INTEGER and DECIMAL may be abbreviated as INT and DEC.

### ORDER BY Command (2.3.7)

ORDER BY <ordering list>:

where: <ordering list> = $\left(\begin{matrix}\text{LOW}\\ \text{HIGH}\end{matrix}\right)$ <item$_1$>, $\left\{\left(\begin{matrix}\text{LOW}\\ \text{HIGH}\end{matrix}\right)\right.$ <item$_2$>... $\left.\right\}$ :

<item$_n$> = <repeating group> or <element> or <derived name> Either the component name or number may be used for <repeating group> and <element>.

## ACTION CLAUSES (2.4)

Report Writer action clauses include the PRINT, DPRINT and SPRINT print clauses; and the COMPUTE and RESET arithmetic clauses.

### PRINT Clauses (2.4.1)

$\begin{bmatrix}\text{PRINT}\\ \text{DPRINT}\\ \text{SPRINT}\end{bmatrix}$ <print phrase> {, <print phrase>} ...

where: <print phrase> is one of the following:

(a) to print a literal

} (<col>) { <special character><literal><special character>

where: <col> = the left-most print position

<special character> = any legal special character except the system separator, colon, or teletype carriage return. The same special character must occur on both sides of the literal. If (<col>) is omitted, the special character must be a slash (/).

<literal> = the actual sequence of characters to be printed. Blanks are preserved in the same manner as for type TEXT data [MDI].

(b) to print data

} $\begin{bmatrix}\text{L}\\ \text{R}\end{bmatrix}$ (<col> } , <edit picture> { ) { <element>

$\begin{bmatrix}\text{L}\\ \text{R}\end{bmatrix}$ (<col>, <edit picture>)<derived name>

where: L and R = left-justify (L) or right-justify (R) the data item. Only NAME and TEXT values may be left-justified.

<col> = left-most print position of the field described in the edit picture.

| | |
|---|---|
| <edit picture> = | specifies the format and number of characters in the print field for this item (see Sections RW 2.4.1.1 and RW 2.4.1.2). |
| <element> = | data base element name or "C" number. |
| <derived name> = | PAGE, LINE, or the name of an item in a DECLARE or DECLARE EXTERNAL command. |

PRINT, DPRINT, and SPRINT may be abbreviated as PR, DPR and SPR.

### EDIT PICTURE INSERTION CHARACTERS (2.4.1.1)

The user may enhance the readability of output values by including the following insertion characters in the edit picture of the item:

$ - + . , 0 B [ ] / CR DB

In addition, the following are insertion characters for NAME, TEXT, and DATE items (but replacement characters for DECIMAL, INTEGER, and MONEY; see Section RW 2.4.1.2):

| | |
|---|---|
| Z | floating + |
| * | floating - |
| floating $ | floating [ |

where "floating" means multiple consecutive occurrences of the character.

### EDIT PICTURE REPLACEMENT CHARACTERS (2.4.1.2)

Replacement characters in the edit pictures of numeric data suppress leading zeroes and replace them with other characters in printing the data item. Only one type of replacement character may be used in an edit picture, although pictures with replacement characters may also contain insertion characters according to the rules in the following discussion. The replacement characters are:

| | |
|---|---|
| Z | floating + |
| * | floating - |
| floating $ | floating [ |

where "floating" means multiple occurrences of the character. These characters are insertion characters in nonnumeric data items.

## COMPUTE Clause (2.4.2)

COMPUTE <derived name> $\{$ , <derived name> $\}$ ...

where: <derived name> = a derived value defined in a DECLARE command as a system function CNT, SUM, RCNT, or RSUM.

COMPUTE may be abbreviated as CO.

## RESET Clause (2.4.3)

RESET <derived name> }, <derived name>{ ...

where: <derived name> = name of a DECLARE'd variable that is defined as a system function CNT, SUM, RCNT, RSUM.

RESET may be abbreviated as RS.

## SKIP Clause (2.4.4)

(1) SKIP }TO NEW{ PAGE

(2) SKIP <n> LINE}S{

where: <m> = an integer $0 \leq m \leq 63$, causing $\underline{m}$ lines to be skipped. If m = 0, overprinting results.

## REPORT EXECUTION (3.0)

```
GENERATE [ ALL                                      ] }WHERE <where clause>{ :
         [ <reportname> } , <reportname>. . .{      ]
```

where: <reportname> = the name assigned to the report by the FOR REPORT phrase.

ALL = specifies that all reports defined since COMPOSE are to be generated in the order of their definitions.

<where clause> = any legal SYSTEM 2000 WHERE clause (Section IA 3.11) except SAME to determine which sections of the data base are to be selected.

WHERE may be abbreviated as WH.

## SAMPLE REPORTS (4.0)

For sample reports refer to the Portfolio Data Base foldout in the user information manual and the Personnel Data Base definition in the Report Writer supplement manual, page RW 4-23.

## REPORT WRITER ERROR MESSAGES (5.0)

For a listing of error messages refer to Section 5.0 of the Report Writer supplement manual.

# INDEX