

## NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM  
MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT  
CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED  
IN THE INTEREST OF MAKING AVAILABLE AS MUCH  
INFORMATION AS POSSIBLE

"Made available under NASA sponsorship  
in the interest of early and wide dis-  
semination of Earth Resources Survey  
Program information and without liability  
for any use made thereof."

T 78-10734 #  
JSC- 13789  
**80-10254**

DETAIL DESIGN SPECIFICATION  
FOR  
ENHANCEMENT OF THE AUTOMATIC STATUS  
AND TRACKING SYSTEM SOFTWARE

NASA CR-  
160634

Job Order 71-695  
(TIRF 77-0035)

Prepared By  
Lockheed Electronics Company, Inc.  
System and Services Division  
Houston, Texas  
Contract NAS 9-15200

For  
EARTH OBSERVATIONS DIVISION  
SCIENCE AND APPLICATIONS DIRECTORATE

(E80-10254) DETAIL DESIGN SPECIFICATION FOR  
ENHANCEMENT OF THE AUTOMATIC STATUS AND  
TRACKING SYSTEM SOFTWARE (Lockheed  
Electronics Co.) 102 p HC A06/MF A01

N80-30837  
Unclass  
00254

CSCL 05B G3/43



*National Aeronautics and Space Administration*  
**LYNDON B. JOHNSON SPACE CENTER**  
*Houston, Texas*

November 1977

LEC- 11512

JSC- 13789

DETAIL DESIGN SPECIFICATION  
FOR  
ENHANCEMENT OF THE AUTOMATIC STATUS  
AND TRACKING SYSTEM SOFTWARE

Job Order 71-695

(TIRF 77-0035)

Prepared By

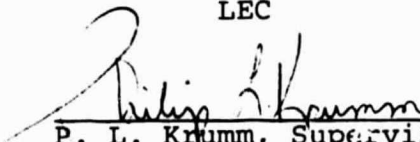
D. K. McCarley

J. M. Everette


K. P. Eckel

APPROVED BY

LEC

  
\_\_\_\_\_  
P. L. Krumm, Supervisor  
Applications Software Section

NASA

  
\_\_\_\_\_  
V. M. Dauphin  
Systems & Facilities Branch

Prepared By

Lockheed Electronics Company, Inc.

For

Earth Observations Division

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION  
LYNDON B. JOHNSON SPACE CENTER  
HOUSTON, TEXAS

November 1977

LEC- 11512

CONTENTS

| Section  | Page |
|--|------|
| 1. INTRODUCTION . . . . .                                    | 1-1  |
| 2. REFERENCES . . . . .                                      | 2-1  |
| 3. OVERVIEW OF MODIFICATION . . . . .                        | 3-1  |
| 3.1 <u>GENERAL</u> . . . . .                                 | 3-1  |
| 3.2 <u>PROGRAM ORGANIZATION</u> . . . . .                    | 3-2  |
| 4. DETAIL DESIGN. . . . .                                    | 4-1  |
| 4.1 <u>GENERAL</u> . . . . .                                 | 4-1  |
| 4.2 <u>NEW SUBROUTINE DETAIL DESIGN</u> . . . . .            | 4-2  |
| 4.3 <u>DESCRIPTION OF SUBROUTINE MODIFICATIONS</u> . . . . . | 4-69 |
| 4.4 <u>DESCRIPTION OF NEW SYSTEM TABLES.</u> . . . . .       | 4-77 |

| Figures  | Page |
|--|------|
| 3.2-1 Subroutines required for commands not using arithmetic operators . . . . .           | 3-4  |
| 3.2-2 Subroutines required for commands using arithmetic operators . . . . .               | 3-5  |
| 3.2-3 Structure Diagram for Change Field Command . . . . .                                 | 3-6  |
| 3.2-4 Structure Diagram for Display Formatted and Joint Display Formatted command. . . . . | 3-7  |
| 3.2-5 Structure Diagram for the Select Non-key and Joint Select Non-key Commands . . . . . | 3-8  |
| 3.2-6 Structure Diagram for Report and Joint Report Commands . . . . .                     | 3-9  |

12-6-74 32

## 1. INTRODUCTION

This document provides a detail design for enhanced data management capabilities of the LACIE Automated Status and Tracking System (ASATS) as requested in Reference 2. ASATS was implemented on the PDP 11/45 using the Regional Information Management System (RIMS), a generalized data base management system. The requested enhancements will be made to RIMS. They included:

- a. Additional Data Base Protection - In order to prevent inadvertent destruction of the data base, additional user interaction to verify the user's desire to execute the command is requested for certain data base update commands.
- b. Null Set Detection and Control - In order to prevent production of headers for reports containing no data, a test and jump command is requested.
- c. Arithmetic Operators - The ability to allow arithmetic operations on fields of data for certain operations is requested.
- d. Interdata Base Comparisons and Arithmetic - RIMS currently has the ability to generate sets by specifying arithmetic relationships between fields and literal values. The ability to specify arithmetic relationships between fields is requested. These relationships should allow the use of arithmetic operators and allow the comparison of fields between a FLOCON record and its parent DAPTS record.
- e. Subgrouping by Field with Maximum, Minimum and Count Functions-The ability to specify fields for which records are to be grouped by value and print field values, maximum or minimum field values, or count of records for the resulting groups is requested. The capability should

include multi-level groupings.

Reference 9 identifies new RIMS commands to be added and existing RIMS commands which will be modified to provide the required enhancements. Reference 10 describes the new commands and changes to existing commands. It also functionally describes new subroutines and modifications to existing subroutines for satisfying those commands.

Section 3 of this document provides an overview to the modifications required to implement the required enhancements.

Section 4 provides a detail description of each new and modified subroutine.

## 2. REFERENCES

The following documents provide a baseline for overall development and implementation of the enhancements to the Automatic Status and Tracking System:

- Reference 1 - Implementation Specification for LACIE ASATS, JSC-11401, Revision A.
- Reference 2 - TIRF 77-0035.
- Reference 3 - Operators Guide for LACIE ASATS, JSC-12729.
- Reference 4 - Users Guide for ASATS, JSC-12535, Revision A.
- Reference 5 - Users Guide for RIMS, LEC-9301, Revision A.
- Reference 6 - "As-Built" Design Specification for ASATS, JSC-12743, Revision A.
- Reference 7 - Functional Design for ASATS, JSC-11835.
- Reference 8 - RIMS Maintenance Document, LEC-9566.
- Reference 9 - Project Development Plan for the Enhancement of the Software of the LACIE Automatic Status and Tracking System.
- Reference 10 - Functional Design Specification for Enhancement of the Automatic Status and tracking system, LEC-11199 and JSC 13110

### 3.0 OVERVIEW OF MODIFICATION

#### 3.1 GENERAL

In order to satisfy the requirements stated in Reference 2 certain RIMS commands will be modified and certain others will be added. Reference 10 describes the function and Syntax for each of these commands. The modified commands are:

- Delete Set (DS)
- Delete Record (DR)
- Delete Key (DK)
- No Key (NK)
- Select Non-key (SN)
- Change Field (CF)
- Display Formatted (DF)
- Joint Display Formatted (JF)

The new commands are

- Test and Jump (JT)
- Label (LA)
- Joint Select Non-key (JN)
- Joint Sort (JS)
- Report (RP)
- Joint Report (JP)

Delete set, Delete Record, Delete Key, and No Key are being modified to satisfy the data base protection requirement. Test and Jump and Label are being implemented to satisfy the null set detection and transfer requirement. Select Non key, Change Field, Display Formatted, and Joint Display Formatted are being modified and Joint Select Non key is being implemented to satisfy the arithmetic operations and interdata base comparison requirements. Joint Sort, Report, and Joint Report are being implemented to satisfy the sub-grouping requirement.



### 3.2 PROGRAM ORGANIZATION

Subroutines which comprise the RIMS system can generally be categorized as (1) system control routines, (2) individual command control routines, (3) common routines and (4) elementary routines. The system control routines are SEL (the main routine) and JLASYS. Usually, command control routines exist for each RIMS command. Common routines are used for those functions common to several commands. Elementary routines perform those functions common to all of RIMS such as string manipulation and data base I/O.

The new data base protection feature will be incorporated by defining two routines which are referenced from the system control routine. The command will be verified before executing the appropriate command control routine.

The Jump Test and Label commands will be implemented by two command control routines referenced from the system control routine.

The Joint Sort command will be implemented by modifying the current sort command control routine.

All other new and modified commands must be able to handle arithmetic operators. Several new subroutines common to these commands have been defined for interpreting command lines, building internal tables and formats, and performing arithmetic operations. While these commands have many common functions, a separate control routine will exist for each command and its associated joint command. Reference 10

describes the general approach for implementing commands which require arithmetic operations.

Figure 3.2-1 through 3.2-6 provides detail program information to illustrate program structure of the modifications and additions. It also provides information for constructing new system overlays. Contents of the figures are:

- Figure 3.2-1- Subroutines required to execute (1) commands requiring input verification (2) Jump Test command (3) label command, and (4) Joint Sort command.
- Figure 3.2-2- Subroutines required for each command which handles arithmetic operators.
- Figure 3.2.3- Structure Diagram illustrating the hierarchical relationship of change Field Subroutines.
- Figure 3.2-4- Structure diagram illustrating the hierarchical relationship of subroutines used by the Display Formatted and Joint Display Formatted Commands.
- Figure 3.2-5- Structure diagram illustrating the hierarchical relationship of subroutines used by the Select Non-key and Joint Select Non-key commands.
- Figure 3.2-6- Structure diagram illustrating the hierarchical relationship of subroutines used by the Report command and Joint Report command.

Subroutines classified as elementary routines; SUBSTR, GET, PUT, ROLL, INDEX, ETC., are shown on any of the figures because of their extensive usage.

Subroutines identified in Figure 3.2-1 through 3.2-6 are described in Section 4 of this document, Reference 6, and Reference 8.

| <u>DS</u> | <u>DR</u> | <u>DK</u> | <u>NK</u> | <u>JT</u> | <u>LA</u> | <u>JS, SO</u> |
|-----------|-----------|-----------|-----------|-----------|-----------|---------------|
| SEL       | SEL       | SEL       | SEL       | SEL       | SEL       | SEL           |
| JLASYS    | JLASYS    | JLASYS    | JLASYS    | JLASYS    | JLASYS    | JLASYS        |
| DBPRØ     | DBPRØ     | DBPRØ     | DBPRØ     | JLASYS    | JLASYS    | JLASYS        |
| CRESTS *  | DELREC *  | DELKEY *  | CRESTS *  | TJUMP     |           |               |
| LØDFMT *  | DELR *    | LØCATE *  | LØDFMT *  |           |           | SØRTP         |
| RESTRX *  | LØCREC *  |           | RESTRX *  |           |           | SORTS         |
| APSINT *  |           |           | APSINT *  |           |           | GETREC        |
| SETINI *  |           |           | SETINI *  |           |           | SETINI        |
| XXINI *   |           |           | XXINI *   |           |           | SETØUT        |
| GETREC *  |           |           | GETREC *  |           |           | SSORT         |
| LMVTAB *  |           |           | LMVTAB *  |           |           | XXINI         |
| TFORM *   |           |           | TFORM *   |           |           | XXØUT         |
| APSTUP *  |           |           | APSTUP *  |           |           | PART          |
| DELR *    |           |           | APSCNT *  |           |           |               |
| APSCNT *  |           |           | AUPOST *  |           |           |               |
| AUPOST *  |           |           |           |           |           |               |

\*UNMODIFIED EXISTING ROUTINE

Figure 3.2-1 Subroutines required for commands  
not using arithmetic operators

| <u>CF</u> | <u>SN, JN</u> | <u>DF, JF</u> | <u>RP, JP</u> |
|-----------|---------------|---------------|---------------|
| SEL       | SEL           | SEL           | SEL           |
| JLASYS    | JLASYS        | JLASYS        | JLASYS        |
| CFCR      | JNSNCR        | JFDFCR        | JPRPCR        |
| AEINIT    | AEINIT        | AEINIT        | AEINIT        |
| SQZE      | SQZE          | SQZE          | SQZE          |
| CICFDF    | RLCLPR        | LØDFMT        | SETIN1 *      |
| RLCLPR    | ADDLT         | CICFDF        | XXIN1 *       |
| ADDLT     | ADDNM         | RLCLPR        | CIRP          |
| ADDNM     | ADDDT         | ADDFN         | ADDFN         |
| ADDDT     | DTEINT        | ADDLT         | AEPR          |
| DTEINT    | ADDFN         | ADDNM         | HIER          |
| ADDFN     | AEPR          | ADDDT         | ADDLT         |
| AEPR      | HIER          | DTEINT        | FTFMT         |
| APSINT    | SETIN1        | AEPR          | LØCREC *      |
| SETIN1    | SETOUT        | HIER          | LØDFMT *      |
| XXIN1     | XXIN1         | RPCLPR        | PRNTID        |
| FTFMT     | FTFMT         | BLDTBF        | FTCMP         |
| LØCREC    | LØCREC        | SETIN1        | GETREC        |
| LØDFMT    | LØDFMT        | XXIN1         | TFØRMW        |
| PRNTID    | PRNTID        | FTFMT         | EXCMDS        |
| FTCMP     | FTCMP         | LØCREC        | EXCMD         |
| GETREC    | GETREC        | PRNTID        | DTEINT        |
| TFØRMW    | TFØRMW        | FTCMP         | CHAR          |
| EXCMDS    | EXCMDS        | GETREC        | TFØRMZ        |
| EXCMD     | EXCMD         | TFØRMW        | DISFMT *      |
| CHAR      | CHAR          | EXCMDS        |               |
| APSTUP    | CHAR          | EXCMD         |               |
| TFØRMZ    | XXOUT         | CHAR          |               |
| REPR      | ENDSET        | TFØRMZ        |               |
| APSCNT    |               | DISFMT        |               |
| AUPOST    |               |               |               |
| RPCLPR    |               |               |               |
| HIER      |               |               |               |

\*UNMODIFIED EXISTING ROUTINE

Figure 3.2.2 Subroutines required for commands using arithmetic operators

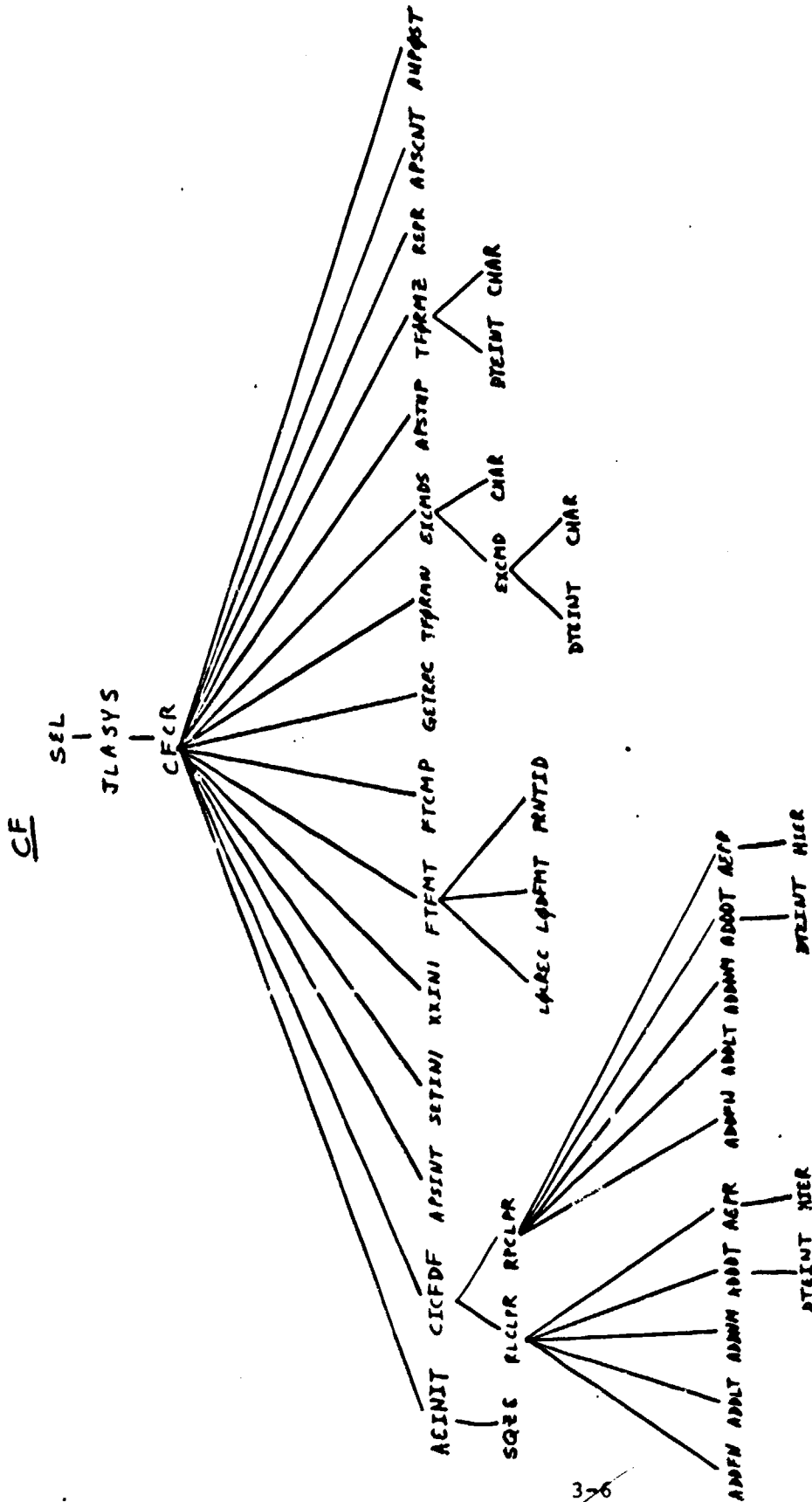


Figure 3.2-3 Structure Diagram for Change Field Command

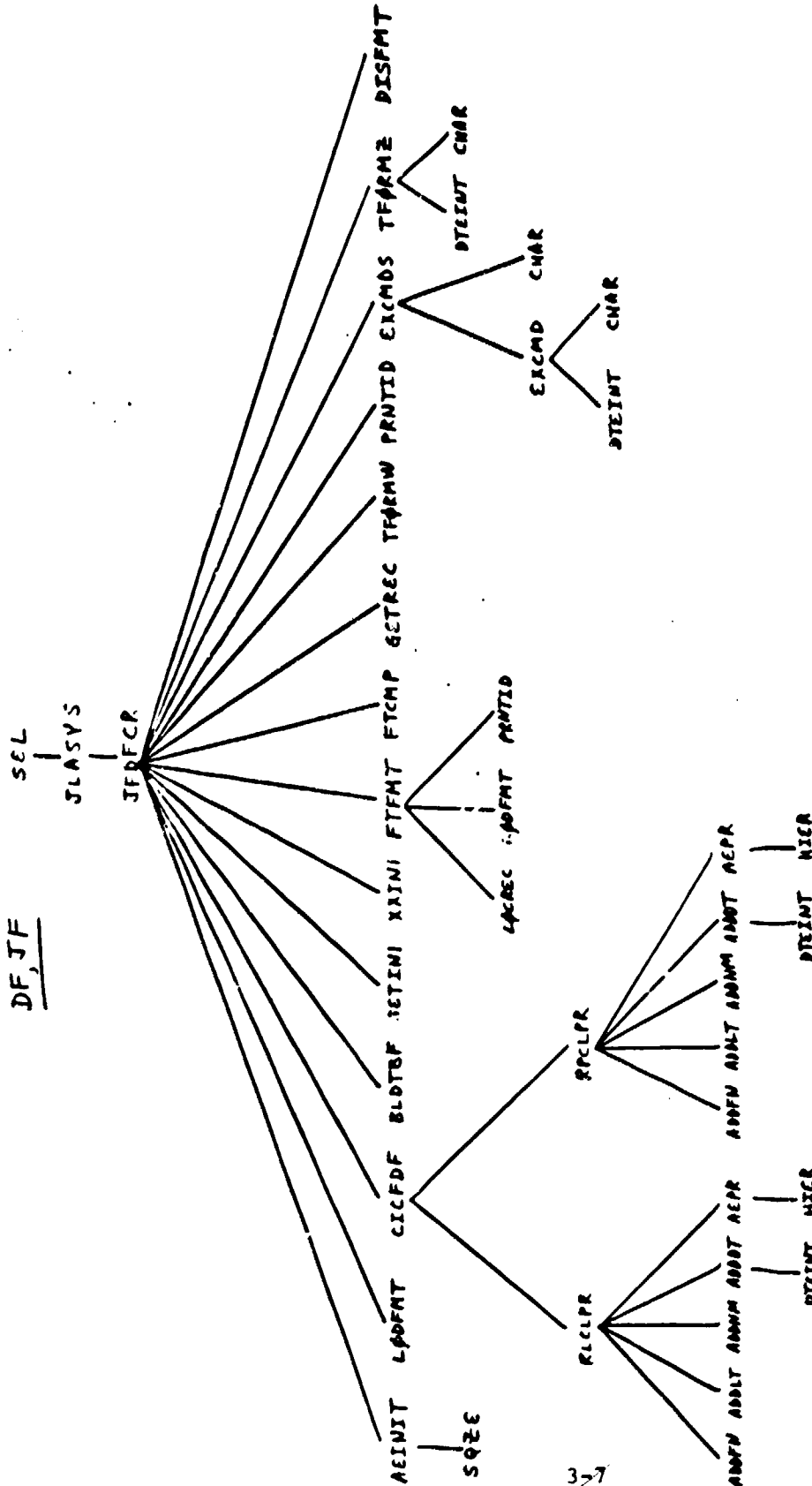


Figure 3.2-4 Structure Diagram for Display Formatted and Joint Display Formatted Command.

SN, JN

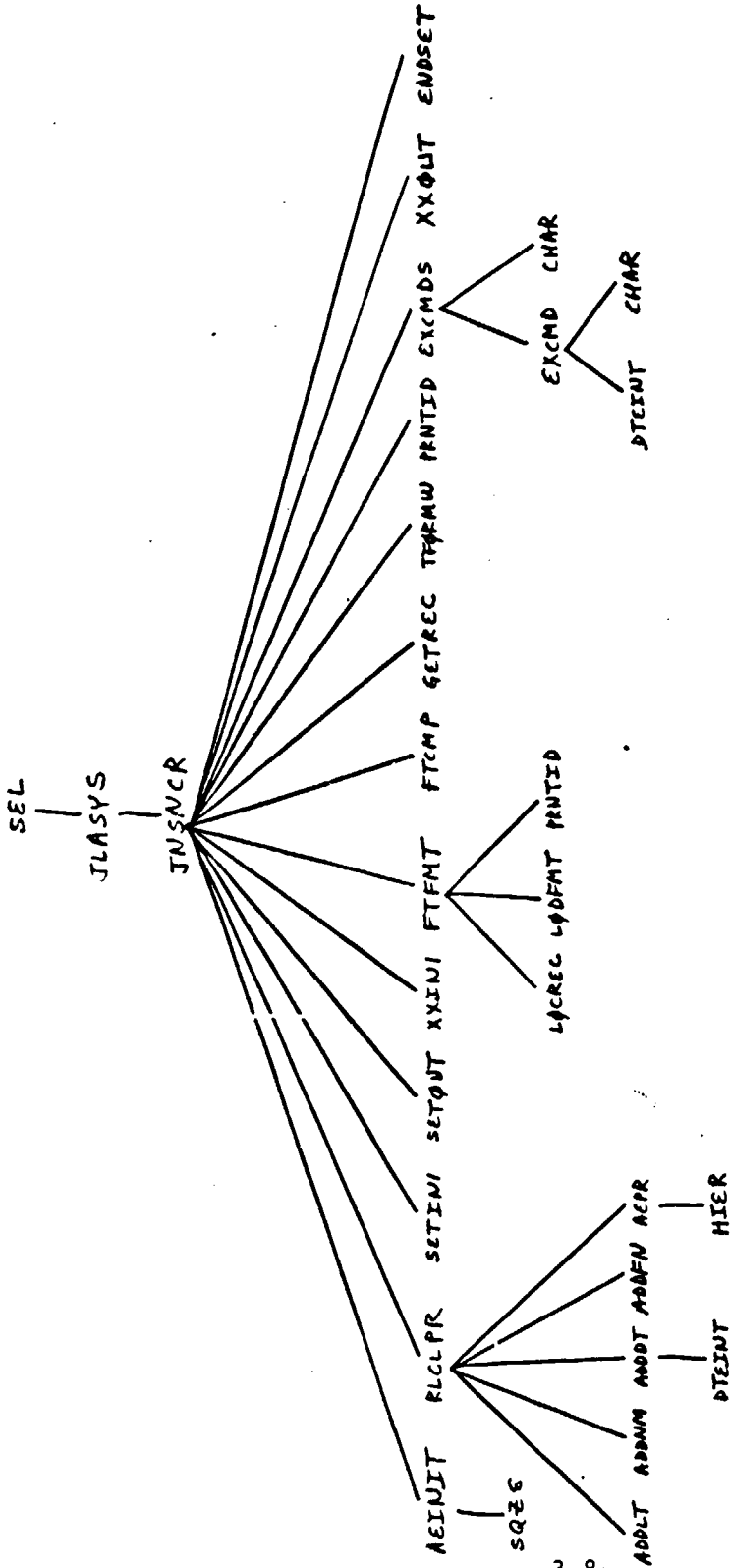


Figure 3.2-5 Structure Diagram for the Select Non-key and Joint Selects Non-key Commands

RP,JP

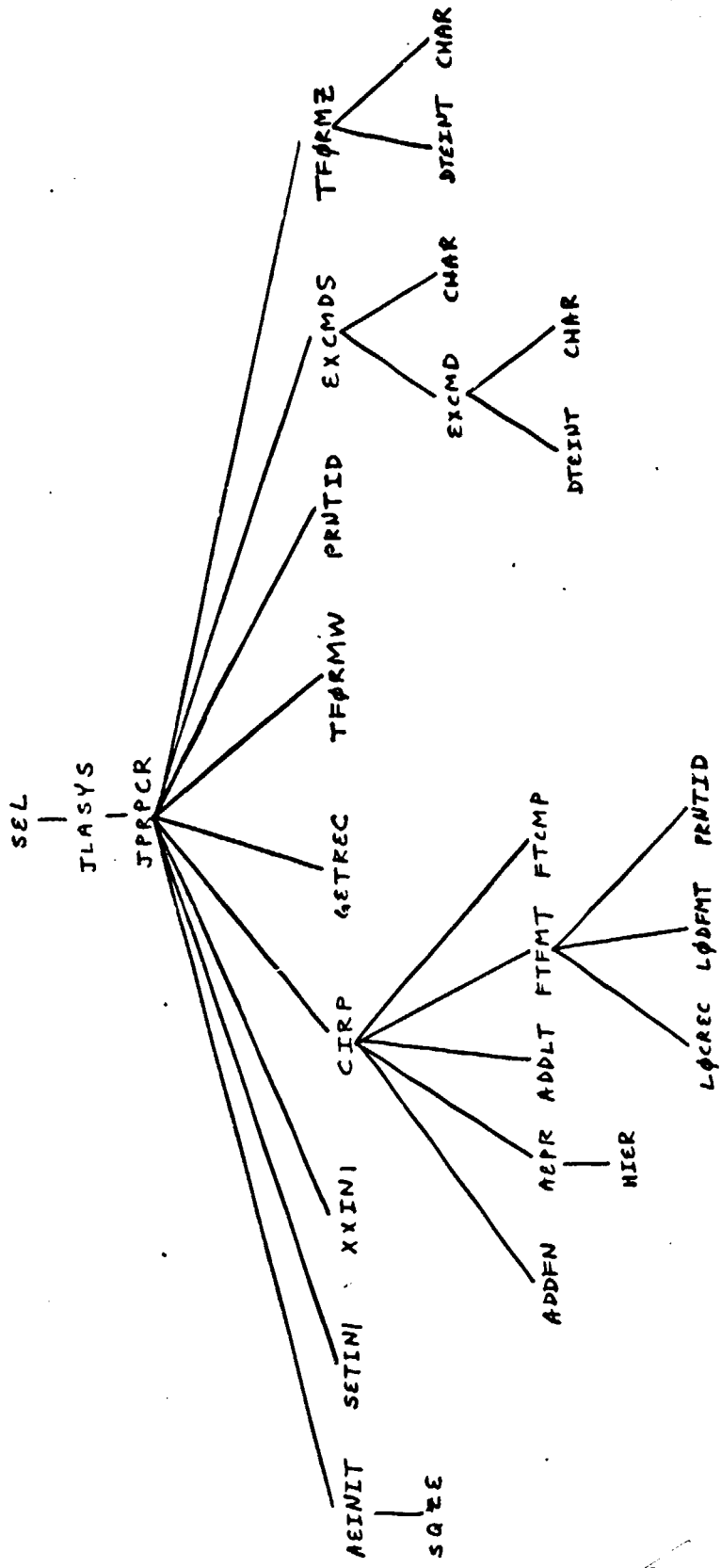


Figure 3.2-6 Structure Diagram for Report and Joint Report Commands



#### 4. DETAIL DESIGN

##### 4.1 GENERAL

The following sections contain detail program descriptions for all new subroutines, description of all subroutine modifications, description of new system tables, and a description of new common blocks.

#### 4.2 NEW SUBROUTINE DETAIL DESIGN

This section contains the Detail design for all new subroutines.  
They are listed in alphabetical order by subroutine name.

Name: ADDDT

Purpose: To initialize the Working Buffer Format for a date encountered in the input command.

Linkage:

- Calling sequence: CALL ADDDT (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: DTEINT
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the date starts (# sign).  
NC = integer variable; number of characters in the date literal, including the # sign.

Output Description: ROW = integer variable; the row number of the Working Buffer Format into which the date reference is placed.

Process Description: The pointer to the last used row of WBF, the Working Buffer Format, is incremented by one. \$L is stored into the second word of the row of WBF. The value of 4 is stored into the sixth word of the row of WBF. The value of -1 is stored into the seventh word of the row of WBF. The subroutine DTEINT is used to convert the date from a character string in the command line to a binary integer in the first word of the row of WBF. The row number is stored in ROW, and a return to the calling routine is made.

Name: ADDFN

Purpose: To initialize the Working Buffer Format for a field name encountered in the input command line.

Linkage:

- Calling sequence: CALL ADDFN (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: COMSTR, SUBSTR
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the field name starts.  
NC = integer variable; number of characters in the field name.

Output Description: ROW = integer variable; the row number of the Working Buffer Format, WBF, into which the field name reference is placed.

Process Description: Column two of WBF is searched for the field name. If it is found, the row number is stored into ROW, and a return is made to the calling routine. If it is not found, the pointer to the last used row of WBF is incremented by one, the field name is placed in column two of that row, that row number is stored in ROW, and a return is made to the calling routine.

Name: ADDLT

Purpose: To store a reference to an alphanumeric literal into the Working Buffer Format

Linkage:

- Calling sequence: CALL ADDLT (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: None
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where literal starts (first quote mark).  
NC = integer variable; number of characters in the literal, including the beginning and ending quote marks.

Output Description: ROW = integer variable; the row number of the Working Buffer Format into which the reference to the literal is stored.

Process Description: The pointer to the last used row of WBF, the Working Buffer Format, is incremented by one. \$T is stored into the second word of the row of WBF. The value of FC+1 is stored into the fifth word of the row of WBF. The value of NC-2 is stored into the sixth word of the row of WBF. Zero is stored into the seventh word of the row of WBF. The row number is stored into ROW, and a return to the calling routine is made.

Name: ADDNM

Purpose: To initialize the Working Buffer Format for a number literal encountered in the input command.

Linkage:

- Calling sequence: CALL ADDNM (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: INPARM
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the number starts.  
NC = integer variable; number of characters in the number.

Output Description: ROW = integer variable; the row number of the Working Buffer Format, WBF, into which the number reference is placed.

Process Description: The pointer, NWBF, to the last used row of WBF is incremented by one. Then \$L, 4, and -1 are stored in WBF (2, NWBF), WBF (6, NWBF), and WBF (7, NWBF), respectively. INPARM is used to convert the number from a character string in CMD to a binary integer in WBF (1, NWBF). The row number, NWBF, is stored in ROW, and a return is made to the calling routine.

Name: AEINIT

Purpose: To initialize standard areas of core for commands which allow arithmetic expressions in their syntax.

Linkage:

- Calling sequence: CALL AEINIT(IND, SETNO, FMTNO, ERR)
- Common blocks used: SYSCOM, SY3COM, CLTBL
- Subroutines or functions used: SQZE, INPARM
- Files used: Command file (logical unit 13), Message file (logical unit 7)

Input Description: IND = integer variable; indicator to cause special processing for certain commands. Set 0 for SN, JN, and CF commands. Set 1 for DF and JF commands. Set 2 for RP and JP commands.

Output Description: SETNO = integer variable; contains the set number converted from the input command line.  
FMTNO = integer variable; contains the format number converted from the input command line, if there is one.  
ERR = integer variable; returned zero if no errors found, non-zero if any error is found.

Process Description:

1. Zeroes are stored in all words of /SY3COM/.
2. Blanks are stored in all words of the Working Buffer, WBUF.
3. Word 6 of row 1 of the BY Processing Table, EPT, is initialized to '+\$-&' to cause the first record read in an RP or JP command to create a top level BY change.

4. The first word of array COMMAS in /CLTBL/ is initialized to 1. Variable FC, used to point to the next available character in array CMD of /SYSCOM/ (where the input command will be packed), is initialized to 1. The logical unit number for the command file, U13, is retrieved from U(13) in /SYSCOM/. The logical unit number for the message file, U7, is retrieved from U(7).
5. A call is made to SQZE (STR, 1, 80, CMD, FC, NCS, COMMAS) to transfer and compact the input command string (in STR of /SYSCOM/) into array CMD. NCS is returned as the number of characters stored by SQZE into CMD, and pointers to syntactically meaningful commas are returned in COMMAS.
6. FC is incremented by NCS.
7. If  $FC \leq 401$ , go to step 9.
8. Set ERR = 1 and return to the calling routine.
9. If  $COMMAS(1) < 0$ , meaning the last input card has been processed for an RP or JP command, then negate COMMAS(1) and go to step 13.
10. If IND = 2, go to step 12.
11. If the last character stored in CMD is a comma, then go to step 13, otherwise increment COMMAS(1) by 1, store FC into COMMAS(COMMAS(1)), and go to step 13.
12. Read, from unit U13, 80 characters into the beginning of array STR. If the actual unit number for the message file = 7, then go to step 5, otherwise echo the input string by writing, to unit U7, 80 characters from the beginning of STR, and go to step 5.



13. Calculate NC, the number of characters in the input set number, =  $\text{COMMAS}(2) - 3$ . Then convert  $\text{SETNO} = \text{INPARM}(\text{CMD}, 3, \text{NC})$ .
14. If  $\text{SETNO} \leq 0$  or  $\text{SETNO} \geq \text{TABNO}$ , then go to step 8.
15. If  $\text{IND} \neq 1$ , go to step 18.
16. Calculate NC, the number of characters in the input format number, =  $\text{COMMAS}(3) - \text{COMMAS}(2) - 1$ . Then convert  $\text{FMTNO} = \text{INPARM}(\text{CMD}, \text{COMMAS}(2) + 1, \text{NC})$ .
17. If  $\text{FMTNO} \leq 0$ , go to step 8.
18. Return to the calling program.

Name: AEPR

Purpose: To parse an arithmetic expression made up of arithmetic operators and operands (Field names, dates, or integer constants), entering operands (or pointers to them) into the working buffer format and building a sequence of internal commands to evaluate the expression and store the value into a specified result variable.

Linkage:

- Calling sequence: CALL AEPR (FC, NC, PTR, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: ADDFN, COMSTR, DTEINT, INDEX, INPARM, STAEPR, VERIFY
- Files used: None

Input Description:

FC = integer variable; character number in command line at which to begin processing.

NC = integer variable; number of characters to process.

PTR = integer variable; location into which the results are to be stored.

Output Description:

ERR = integer variable; processing or syntax error indicator. Normal command table containing internal commands to evaluate the expression and store the results.

Process Description:

Error flag set to zero and all internal variables set to appropriate value. The expression is scanned for paired

brackets and valid alphanumeric characters. If brackets not paired or any invalid character found ERR set to 2 and return executed. Otherwise, the number of paired brackets saved for later use, character pointers and counters set as needed and the expression is scanned and PASS 1 executed as follows:

1. (a) If the next character encountered is not an open bracket do step 2. Otherwise an open bracket is stored in the next location of VTAB and a -99 is stored in the next location of OPC (b). If this is the last character the error exit is taken otherwise update pointers and counters and redo step 1(a).
2. If the next character encountered is a closed bracket ERR is set to 2 and return executed. Otherwise do step 3.
3. The next 13 characters are searched for either the end of the scan or an arithmetic operator. If the end of the scan is found the pointer is set to the end of the scan +1 location otherwise the location of the next operator or bracket will be found. Then the current character will be checked to see if it is a pound sign. If it is not a pound sign step 4 is executed. Otherwise the next four characters past the pound sign are checked to see if they are numeric digits. If they are not digits, ERR is set to 2 and return executed. Otherwise DTEINT is called to convert the date to an integer, the appropriate

data is stored in the normal command buffer and step 6 executed.

4. The current character is checked to see if it is a literal value. If it is not a literal value step 5 is executed. Otherwise the literal is converted to an integer, the appropriate data is stored in the normal command buffer and step 6 executed.
5. If none of the above were executed then the next operand is a field name. In this case ADDFN is called to store the data in the normal command buffer and step 6 executed.
- 6(a) If the last operand or operator has been processed a-999 is placed in the current location of OPC and the pass 2 is executed as shown starting at step 8. Otherwise the next operator is checked to see if it is either an open or close bracket. If it is an open bracket the error exit is taken. If it is a closed bracket step 7 is taken. (b) Otherwise the operator is stored in the next location of OPC and step 1(b) executed.
7. The character after the closed bracket is checked. If it is an open bracket the error exit is taken. Otherwise a-88 is stored in the next location of VTAB & OPC and the pointer and counters updated. If the character after the closed bracket was itself a closed bracket step 6 is taken. If the operator after the closed bracket was not a

closed bracket and at the same time located at the end point of the expression to be processed, the error exit is taken otherwise step 6b is taken.

8. The count of data in VTAB & OPC is saved for later use. If the number of paired brackets is zero, PASS 3 is executed as shown starting at step 10. Otherwise step 9 is executed.
9. The OPC table is searched and the innermost paired brackets, as indicated by a -99 and a -88 respectively, is found along with their index location. Then STAEPR is called to store the data into the normal command table. Then the remainder of the VTAB & OPC tables is written over the area where the paired brackets were stored, the count of data in VTAB & OPC is decreased by the amount of data processed by STAEPR, the number of paired brackets is decreased by 1 and step 8 taken.
10. If more than one line of data is left in VTAB & OPC STAEPR is called to store the data into the normal command table. Otherwise the normal command table is updated with the calling argument PTR and column 5 of all used areas of the normal command table is updated to point to the next expression area to be processed.

Name: BLDTBF

Purpose: To convert a data base format into a form suitable for standardized processing.

Linkage:

- Calling sequence: CALL BLDTBF(P)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: COMSTR
- Files used: None

Input Description: P = integer variable; page number of array FMT in /SY2COM/ where the data base format is currently stored.

Output Description: None

Process Description:

1. Initialize K to zero.
2. Initialize I to two, the first row of the data base format which contains a field name.
3. If NWBF, the pointer to the last used row of WBF, the Working Buffer Format in /SY3COM/, is = 0, go to step 8.
4. Initialize J to one, the first row of WBF.
5. If WBF (2, J) = \$R, \$T, or \$L, go to step 7.
6. Use COMSTR to compare the field name in column 2 of WBF to the field name in column 2 of FMT at row I. If a match is found, go to step 9.
7. Increment J by one. If J  $\leq$  NWBF, go back to step 5.
8. Increment NWBF by one. Set J = NWBF. Transfer the field name from column 2 of row I of FMT to column 2 of row J of WBF.
9. Increment NTBF, the pointer to the last used row of TBF, the Target Buffer Format in /SY3COM/, by one. Store a one in

TBF (1, NTBF) and store J in TBF (2, NTBF).  
Transfer columns 3, 4, and 5 of row I of  
FMT to columns 3, 4 and 5 of row NTBF of  
TBF.

10. If the target field type, TBF (7, NTBF),  
= 4, set K = 1.
11. Increment I by one. If  $I \leq \text{FMT}(6, 1, P) + 1$ ,  
then go back to step 3.
12. If K = 0, return to the calling routine.
13. Search column 2 of WBF for the field names  
"UNLOAD" and "LSD". If "UNLOAD" is found  
at row I, set TBF (4, 1) = -I. If "LSD"  
is found at row I, set TBF(4, 2) = -I.
14. If "UNLOAD" is not found in WBF, increment  
NWBF by one, store "UNLOAD" as a field  
name in column 2 of WBF at row NWBF, and  
set TBF(4,1) = -NWBF.
15. If "LSD" is not found in WBF, increment  
NWBF by one, store "LSD" in column 2 of  
WBF at row NWBF, and set TBF (4,2)  
=-NWBF.
16. Return to the calling routine.

Name: CFIND

Purpose: To locate any single character of one string within another string.

Linkage:

- Calling sequence: CALL CFIND (STRA, STA, NCA, STRB, STB, NCB, LOCA, LOCB)
- Common blocks used: None
- Subroutines or functions used: INDEX
- Files used: None

Input Description: STRA = integer array name; start of string to be searched  
STA = integer variable; character number of STRA at which to begin search  
NCA = integer variable; number of characters in STRA to be searched  
STRB = integer array name; start of string containing characters for which to search  
STB = integer variable; character number of STRB where search characters start  
NCB = integer variable; number of search characters in STRB to be used.

Output Description: LOCA = integer variable; character number in STRA where the first find was made. Zero if no characters in STRB were found in STRA.  
LOCB = integer variable; character number in STRB of the character found. Zero if none found.

Process Description: Successive characters of STRA, starting at STA and continuing for NCA characters, are individually compared to the characters in STRB (via the INDEX function) until a match is found or STRA is exhausted.



Name: CICFDF

Purpose: To direct the command interpretation phase of the CF, DF, and JF commands.

Linkage:

- Calling sequence: CALL CICFDF(IND, ERR)
- Common blocks used: SY3COM, CLTBL
- Subroutines or functions used: CFIND, RPCLPR, RLCLPR
- Files used: None

Input Description: IND = integer variable; indicator for which command is being processed. Set zero for CF command. Set non-zero for DF or JF commands.

Output Description: ERR = integer variable; error indicator. Returned zero if no errors found. Returned non-zero if any error is found.

Process Description:

1. The error indicator, ERR, is initialized to 0. The replacement expression found indicator, REF, is initialized to 0. The commas pointer, CP, is initialized to 2.
2. If IND  $\neq$  0, CP is changed to equal 3.
3. The first character pointer, FC, is calculated = COMMAS(CP)+1. The number of characters, NC, between commas is calculated = COMMAS(CP+1)-FC.
4. Use CFIND to check the character at FC for a single quote mark, a number sign, or a numeric character. If any of these are found, go to step 9.
5. Use CFIND to check all NC characters for an arithmetic operator, parenthesis, equal sign or period. If one of these is found, go to step 7.

6. Set ERR = 1 and return to the calling routine.
7. If an equal sign was found, go to step 14.
8. Since no equal sign was found, the clause must be a relational clause. Since all relational clauses are to be before replacement clauses, check REF to see if a replacement clause has been found. If REF  $\neq$  0, go to step 6.
9. Use RLCLPR(FC, NC, ERR) to process the relational clause and return ERR non-zero if errors were found.
10. If ERR  $\neq$  0, go to step 6.
11. Increment CP by 1. If CP  $\neq$  COMMAS (1), go to step 3.
12. If IND  $\neq$  0, return to the calling routine.
13. The CF command must have a replacement clause, so if REF = 0, go to step 6, otherwise return to the calling routine.
14. Set REF = 1 and use RPCLPR (FC, NC, I, IND, ERR) to process the replacement clause. I was returned previously from CFIND as the location of the equal sign, and ERR will be returned non-zero from RPCLPR if any errors are found.
15. If ERR  $\neq$  0, then go to step 6, otherwise go to step 11.

Name: CIRP

Purpose: To direct the activities of parsing the command line, building tables, and building buffer formats for the RP and JP commands.

Linkage:

- Calling sequence: CALL CIRP(CIND, RECID, ERR)
- Common blocks used: SY3COM, CLTBL
- Subroutines or functions used: COMSTR, CFIND, ADDFN, AEPR, ADDLT, INDEX, INPARM, FTFMT, FTCMP
- Files used: None

Input Description: CIND = integer variable; command indicator. Set 0 for RP command or non-zero for JP command.  
RECID = integer variable; contains the record ID (accession number ) of the first record of the input set.

Output Description: ERR = integer variable; error indicator. Returned zero if no errors are found and non-zero if any error is found.

Process Description:

1. Counters and pointers are initialized.
2. If there are some clauses in the command, go to step 4.
3. Set ERR = 1 and return to the calling routine.
4. If there are no characters in the clause, go to step 3.
5. If this clause is the first one and it is not a BY clause, then go to step 3.
6. If this clause is not a BY clause and not the first clause, then go to step 16.

7. If this BY clause is occurring after an E&E BY clause has occurred, then go to step 3.
8. If this is the sixth BY clause, then go to step 3.
9. If there is no grouping field name for this BY clause, then go to step 3.
10. If this is an E&E BY clause, then if there are no report expressions, then go to step 3, otherwise store zeroes in the first 3 columns of the BY Processing Table, BPT in /SY3COM/, for this BY clause and go to step 15.
11. Use CFIND to check the grouping field name for arithmetic operators or parentheses. If any are found, go to step 13.
12. Set columns 2 & 3 of BPT = 0 for this BY clause, use ADDFN to store the grouping field name in the Working Buffer Format, WBF in /SY3COM/, store the row number returned by ADDFN into column 1 of BPT for this BY clause, and go to step 15.
13. Store the negative of the next available row number of WBF in column 1 of BPT for this BY clause. Store \$R in column 2 of that row of WBF. Store the next available row number of the Normal Command Table, CTBL in /SY3COM/, into column 2 of BPT for this BY clause. Store 4 and -1 into columns 4 & 5 of the \$R row of WBF.
14. Use AEPR to generate the commands which evaluate the arithmetic expression. If AEPR found any syntax errors, go to step 3. Calculate the number

of commands generated by AEPR and store this number into column 3 of BPT for this BY clause.

15. Move to the next pair of commas. If there is none, go to step 29, otherwise go back to step 4.
16. If this report expression is not a text type, go to step 18.
17. Use ADDLT to create an entry in WBF for this text literal and store values in the Target Buffer Format, TBF in /SY3COM/, to cause this text to be printed at the start or conclusion of this BY clause (depending on whether single or double quote mark characters were used). Go to step 15.
18. Compare the beginning characters of the report expression with an internal table of function names. If no match is found, go to step 21.
19. If this function reference is found in an E&E BY clause, then go to step 3.
20. Use ADDFN to create the field name reference in WBF for the field name specified in the function. Create a \$R row in WBF for the results of the function to be carried and create a reference in TBF to get the results printed upon the conclusion of this BY clause. Based on which function was specified, store an initialization value in column 1 of the \$R row of WBF. Then go to step 15.
21. Use INDEX to check for an equal sign in the report expression. If there is one, go to step 24.
22. This report expression is only a field name. If it has greater than 12 characters, go to step 3.

23. Use ADDFN to create a reference to the field name in WBF, and create an entry in TBF to cause the value of this field to be printed at the beginning of this BY clause. Then go to step 15.
24. Use CFIND to determine if this report expression begins with an I or a D. If it begins with neither, then go to step 3.
25. Use INPARM to convert the input field width to a binary integer. If it is greater than 99, then go to step 3.
26. Create a results field (\$R) in WBF and a target field for printing in TBF. Store the target field type in TBF as a 1 or a 2 based on whether the report expression began with an I or a D, respectively.
27. Initialize column 4 of BPT for this BY clause if it has not already been done.
28. Use AEPR to process the arithmetic expression to the right of the equal sign, and accumulate the number of commands generated into column 5 of BPT for this BY clause. Then go to step 15.
29. If CIND = 0, then set NLVLS, the number of data base levels, = 1, otherwise set NLVLS = 2.
30. Use FTFMT (RECID, NLVLS, ERR) to search the data base formats, collecting field information for WBF and SBF, the Source Buffer Format in /SY3COM/. If ERR is returned non-zero, go to step 3.
31. Now that the type and length of data base fields are known, this information is used to complete needed portions of WBF

and TBF where just field names and functions with field names are the report expressions.

32. Where function results are called for in WBF, a command is entered in the Function Command Table, FCTBL in /SY3COM/.
33. Use FTCMP to generate starting character positions in WBF and initial values in WBUF.
34. Use FTCMP to generate starting character positions with two spaces between fields in TBF.
35. Return to the calling routine.

Name: CFCR

Purpose: To direct the overall processing sequence for the CF command.

Linkage:

- Calling sequence: CALL CFCR
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, CICDFD, APSINT, SETIN1 XXIN1, FTFMT, FTCMP, GETREC, TFORMW, EXCMD5, APSTUP, TFORMZ, REPR, APSCNT, AUPOST
- Files used: Message file (logical unit 7), Deleted keys file (logical unit 10), New keys file (logical unit 9)

Input Description: None

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/ and use AEINIT to initialize /SY3COM/, returning input set number in SETNO and a non-zero in ERR if any errors were found.
2. If ERR = 0, go to step 4.
3. Write on U7, "Command terminated due to syntax error." and return to the calling routine.
4. If  $COMMAS(1) \leq 2$ , meaning there are no replacement clauses, then go to step 3.
5. Use CICDFD(0, ERR) to complete the command interpretation, returning ERR non-zero if any errors were found.
6. If  $ERR \neq 0$ , go to step 3.
7. Use APSINT to initialize for storing deleted keys on file U(10) and new



keys on file U(9). Use SETIN1 to initialize for returning record ID's via XXIN1. Use XXIN1 (RID) to return the first record ID in RID.

8. If RID = 0, return to the calling routine.
9. Use FTFMT (RID, 1, ERR) to complete the buffer formats with information about fields whose names occurred in the input command, returning ERR non-zero if an error occurred.
10. If ERR ≠ 0 go to step 3.
11. Use FTCMP to complete starting location information in the Working Buffer Format, WBF in /SY3COM/.
12. Loop through the Target Buffer Format, TBF in /SY3COM/, comparing its column 2 contents with the values found in column 2 of the Multilevel Move Table, MLMT in /SY3COM/. When a match is found at row I of TBF and row K of MLMT, extract L = column 1 of row K of MLMT, so that L points to the matching row of the Source Buffer Format, SBF in /SY3COM/. Then transfer the key field indicator from SBF(1,L) to TBF(3,I), the data base format row number from SBF(2,L) to TBF(4,I), the starting character position in the data base record from SBF(5,L) to TBF(5,I), the length of the field from SBF(6,L) to TBF(6,I), and the type of the field from SBF(7,L) to TBF(7,I).
13. Use GETREC(1, RID, STAT) to retrieve record RID into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.

14. If STAI  $\neq$  0, bypass this record by going to step 23.
15. Use TFORMW(1,1) to transfer needed fields from row 1 of BUF to WBUF, directed by row 1 of the Move Table Control Table, MTCT in /SY3COM/.
16. Set EF = 0 and use EXCMD5 to execute the commands in the Normal Command Table, CTBL in /SY3COM/, returning CFLAG as false if there was a failure of a relational clause, and returning EF non-zero if a command could not be executed for some reason.
17. If CFLAG is false, bypass this record by going to step 23.
18. If EF  $\neq$  0, bypass this record by going to step 23.
19. Loop through TBF looking for key fields. When one is found (TBF(3,I)  $\neq$  0), then use APSTUP to store the key to be deleted on file U(10).
20. Use TFORMZ(1,1) to transfer changed fields from WBUF to row 1 of BUF.
21. Loop through TBF, and for each key field found, use APSTUP to store the new key on file U(9).
22. Use REPR to replace the old record in the data base with the revised one in row 1 of BUF.
23. Use XXIN1(RID) to get the next record ID into RID.
24. If RID  $\neq$  0, meaning there was a next record, then go back to step 13.
25. Use APSCNT to retrieve the number of keys to be changed in the data base.
26. Use AUPOST to delete the keys stored on file U(10).

27. Use AUPOST to add the keys stored on file U(9).
28. Return to the calling routine.

~~4-21~~

39

Name: DBPRO

Purpose: To prevent accidental alteration of the data base for certain commands.

Linkage:

- Calling Sequence: CALL DBPRO (FLAG)
- Common block used: SYSCOM
- Subroutines or functions used: INDEX
- File used: None

Input Description: A command line containing the input command line plus a "YES" or "NO" after the command or a command line containing the input line then another input line containing "YES" or "NO" in response to an output query.

Output Description: FLAG = integer variable; where FLAG = 0 means do not allow data base to be altered and FLAG = 1 means do allow data base to be altered.

Process Description: The input command line is searched for a "YES" or "NO" on the command line past the command. If a "YES" is found, flag is set to "1" and a return performed. If a "NO" is found flag is set to "0" and a return is performed. If neither is found, the command line followed by "YES or NO ?" is output to the appropriate device. Then the response is accepted and flag set to "1" for "YES" or "0" for "NO" as above then a return is performed.

Name: DISFMT

Modification Purpose: To display characters from the input string without transferring them to another array.

Linkage Modification:

- Calling sequence: No change
- Common blocks used: No change
- Subroutines or functions used: SUBSTR is no longer used
- Files used: No change

Input Description Modification: No change

Output Description Modification: No change

Process Description Modification: The call to SUBSTR to transfer the input string to array X is deleted. The WRITE statements are changed to write from array STR instead of X.

Name:

DTEINT

Purpose:

To convert date format to/from a binary integer.

Linkage:

- Calling sequence:  
CALL DTEINT (FUNC, INT, STR, ST, NC)
- Common blocks used: None
- Subroutines or functions used:  
SUBSTR, CHAR, INPARM, MOD
- Files used: None

Input Description:

FUNC = Indicator, if FUNC = 0, converts character string (STR) to an integer (INT).  
If FUNC  $\neq$  0, converts an integer to a character string (STR).  
INT = Integer input.  
STR = character string input.  
ST = starting position of character string.  
NC = number of characters of STR to be converted.

Output Description:

INT = Integer output from converted character string.  
STR = character output from input integer.

Process Description:

In addition to the input variables, this routine contains an internal Julian day conversion table DTAB. DTAB is a one-dimensional array with each element representing the total number of days from the base year to year 'N', where N is the relative position of the array element representing an offset from the base year. If FUNC indicates an integer is to be converted to a character string the input integer date is tested for an invalid date. If this date is greater than the

greatest value of DTAB, the input date is replaced with that particular DTAB element and the conversion process continued. However, if the integer date is less than or equal to zero, blanks are moved to the output string (STR). Assuming the integer date is greater than zero, DTAB is searched until a value that is greater than or equal to the input integer is found. The input integer minus the previous table value gives the day segment of the Julian date. The year segment is then calculated by adding the base year to the DTAB index minus a constant of two. Having converted the integer date to a Julian date format, the results are then converted to an alphanumeric character string by use of the CHAR subroutine. To convert from an alpha Julian date format the year and day segments are calculated. The year portion is subtracted from the base year to serve as an index to pick up the appropriate DTAB element. Once this element is obtained this value is added to the day segment to produce the output integer.

Name: EXCMD

Purpose: To perform the operations specified in one row of a command table.

Linkage:

- Calling sequence: CALL EXCMD (TBL, ROW, ERR, CFLAG)
- Common blocks used: SYSCOM
- Subroutines or functions used: SUBSTR, DTEINT, INPARM, CHAR, COMSTR
- Files used: None

Input Description: TBL = integer array name; starting location of the table containing the command to be executed. ROW = integer variable; contains the row number of TBL where the command to be executed is stored.

Output Description: ERR = integer variable; contains zero on normal command execution or non-zero when command cannot be executed. CFLAG = logical variable; contains .TRUE. normally, but is set to .FALSE. when the command is a logical comparison and the comparison fails.

Process Description: Refer to the Command Table and Command Operations Table layouts in section 4.4 as a supplement to this description. The value retrieved from the references in columns 1-4 of the command table will be referred to as OPND(1), OP, OPND(2), and RESULT, respectively, in this description.

1. ERR and CFLAG are initialized to zero and true, OP is retrieved from TBL (2, ROW), OPND(1)'s pointer, P, is retrieved from TBL (1, ROW), and N is initialized to one.



2. If  $OP > 0$  and  $< 17$ , go to step 4.
3. Set  $ERR = 1$  and return to calling routine.
4. If  $OP > 10$ , go to step 24.
5. If  $P = 0$ , go to step 3.
6. If  $P < 0$ , negate  $P$ , retrieve  $OPND(N)$  from  $REG(P)$ , an array in  $/SY3COM/$ , and go to step 10.
7. Retrieve  $OPND(N)$ 's type from column 5 at row  $P$  of the Working Buffer Format,  $WBF$ , in  $/SY3COM/$ .
8. If  $OPND(N)$ 's type = 0, then if  $OP < 5$ , then go to step 3, otherwise if  $N = 2$ , then go to step 3, otherwise go to step 20.
9. Convert the value of  $OPND(N)$  from the Working Buffer,  $WBUF$  in  $/SY3COM/$ , based on the type, using either subprogram  $SUBSTR$  (type  $< 0$ , a binary integer),  $INPARM$  (type = 1, a numeric character string), or  $DTEINT$  (type = 2, a date character string).
10. If  $N = 1$ , then set  $N = 2$ , retrieve a new  $P$  from  $TBL(3, ROW)$ , and go to step 5.
11. If  $OP > 4$ , go to step 22.
12. Perform the arithmetic operation specified by  $OP$ , using  $OPND(1)$  and  $OPND(2)$  and storing the result in  $RESULT$ . If  $OPND(2)$  of a divide operation = 0, then go to step 3.
13. Retrieve a new  $P$  for  $RESULT$  from  $TBL(4, ROW)$ .
14. If  $P = 0$ , go to step 3.
15. If  $P < 0$ , negate  $P$ , store  $RESULT$  in  $REG(P)$ , and go to step 19.
16. Retrieve  $RESULT$ 's type from column 5 at row  $P$  of  $WBF$ .
17. If type = 0, go to step 3.
18. Convert the value in  $RESULT$  into  $WBUF$  based on the type, and using subprogram  $SUBSTR$  (type  $< 0$ ),  $CHAR$  (type = 1), or  $DTEINT$  (type = 2).

19. Return to the calling routine.
20. If OPND(2)'s pointer in TBL (3, ROW) = 0, or if OPND(2)'s type  $\neq$  0, then go to step 3.
21. Perform an alphanumeric comparison between OPND(1) and OPND(2) and set I to be negative, zero, or positive according to whether OPND(1) < OPND(2), OPND(1) = OPND(2), or OPND(1) > OPND(2), respectively. Go to step 23.
22. Perform arithmetic comparison by setting I = OPND(1)-OPND(2).
23. Leave CFLAG = true or change CFLAG = false based on the following table and then return to calling routine:
 

|         | I < 0 | I = 0 | I > 0 |
|---------|-------|-------|-------|
| OP = 5  | true  | false | false |
| OP = 6  | true  | true  | false |
| OP = 7  | false | true  | false |
| OP = 8  | true  | false | true  |
| OP = 9  | false | true  | true  |
| OP = 10 | false | false | true  |
24. If OP = 16, go to step 40.
25. Retrieve OPND(2)'s pointer, P2, from TBL(3, ROW). If P2 = 0, go to step 3.
26. If OP > 13, go to step 35.
27. If OPND(2) in WBUF is blanks, go to step 39.
28. Retrieve OPND(1) from WBUF using SUBSTR.
29. If OP = 11, set OPND(2) = OPND(1) +1, and go to step 34.
30. Retrieve OPND(2) from WBUF, converting based on its type and using INPARM or DTEINT.
31. Perform a numeric comparison between OPND(1) and OPND(2). If OPND(1) = OPND(2), go to step 39.

32. If  $OPND(1) > OPND(2)$ , then if  $OP = 13$ , then go to step 39, otherwise go to step 34.
33. If  $OP = 12$ , then go to step 39.
34. Use SUBSTR to store  $OPND(2)$  into  $OPND(1)$ 's place in WBUF and go to step 39.
35. Perform an alphanumeric comparison between  $OPND(1)$  and  $OPND(2)$ . If  $OPND(1) = OPND(2)$ , to to step 39.
36. If  $OPND(1) > OPND(2)$ , then if  $OP = 15$ , then go to step 39, otherwise go to step 38.
37. If  $OP = 14$ , go to step 39.
38. Use SUBSTR to store  $OPND(2)$ 's character string in WBUF into  $OPND(1)$ 's character string in WBUF.
39. Return to calling routine.
40. If  $P = 0$ , go to step 3.
41. If  $P < 0$ , negate  $P$ , retrieve RESULT from  $REG(P)$ , and go to step 13.
42. Retrieve  $OPND(1)$ 's type from column 5 at row  $P$  of WBF. If type = 0, go to step 44.
43. Convert  $OPND(1)$  from WBUF into RESULT based on type using subprogram SUBSTR (type < 0), INPARM (type = 1), or DTEINF (type = 2). Go to step 13.
44. Retrieve RESULT's pointer,  $P2$ , from TBL (4, ROW). If  $P2 = 0$ , go to step 3.
45. Transfer  $OPND(1)$ 's character string in WBUF to RESULT's location in WBUF, using SUBSTR, and then to go step 39.

Name:

EXCMDS

Purpose:

To execute a sequence of related command rows in a command table.

Linkage:

- Calling sequence: CALL EXCMDS (TBL, SR, NR, ERRFNC, CFLAG)
- Common blocks used: SY3COM
- Subroutines or functions used: EXCMD, SUBSTR
- Files used: None

Input Description:

TBL = integer array name; starting location of the table which contains the commands to be executed.

SR = integer variable; starting row number within the command table.

NR = integer variable; number of rows to be executed.

ERRFNC = integer variable; indicator for what procedure is to be followed if an error occurs: Zero means do nothing to the results field; non-zero means store blanks or zero in the results field (depending on field type).

Output Description:

ERRFNC = integer variable; set to zero if no errors were encountered. Set to one if an error was encountered.

CFLAG = logical variable; contains .TRUE. except when a relational comparison command has failed, then it contains .FALSE..

Process Description:

1. The last row to be processed is calculated into LR, ERRFNC is saved in EF and set = 0, and I is initialized to SR.
2. A call is made to subroutine EXCMD to execute the command at row I.

3. If CFLAG from EXCMD is returned with a value of false, return immediately to the calling routine.
4. If the error indicator from EXCMD is returned non-zero, go to step 7.
5. Increment I by 1.
6. If  $I > LR$ , return to the calling routine.
7. Set  $ERRFNC = 1$  and retrieve P from column 5 of the current row of the command table. P is the row number to which a jump should be made.
8. If  $EF = 0$ , go to step 13.
9. Retrieve the result pointer, P1, from column 4 of row P-1 of the command table.
10. If  $P1 = 0$ , go to step 13.
11. Retrieve the result type from column 5 of row P1 of the Working Buffer Format.
12. Based on type, store binary zeroes (type < 0), alpha zeroes (type > 0), or blanks (type = 0) into the result location in the Working Buffer.
13. Set  $I = P$  and go to step 6.

Name: FTCMP

Purpose: To calculate starting character positions for fields in generated formats.

Linkage:

- Calling sequence: CALL FTCMP (A, NS)
- Common blocks used: SY3COM
- Subroutines or functions used: SUBSTR
- Files used: None

Input Description: A = integer array name; starting location of the array which contains the format to be completed.  
NS = integer variable; the number of spaces to be inserted between fields.

Output Description: A = integer array name; starting location of the array which contains the completed format.

Process Description:

1. The start character counter, SC, is initialized to one, as is the row counter, ROW.
2. The length of the field at ROW is transferred from A(6, ROW) to L.
3. If L = 0, then processing is finished, so return to the calling routine.
4. If A(2, ROW) contains \$T then bypass start character calculations for this row (since text remains in the command line instead of being transferred to the Working Buffer, WBUF in /SY3COM/) and go to step 7.
5. If A(2, ROW) contains \$L or \$R, then use SUBSTR to initialize WBUF from A(1, ROW).
6. Store SC into A(5, ROW) and calculate the next SC = SC+L+NS.
7. Increment ROW by one and go to step 2.

Name: FTFMT

Purpose: To retrieve information from formats associated with records in the same family tree.

Linkage:

- Calling sequence: CALL FTFMT (RECID, NLVLS, ERR)
- Common blocks used: SYSCOM, SY2COM, SY3COM
- Subroutines or functions used: LOCREC, GET, INPARG, LODFMT, COMSTR, PRNTID
- Files used: Message file

Input Description: RECID = integer variable; record ID of a record at the lowest level of the data base where format information collection is to begin. NLVLS = integer variable; the number of levels of the data base to be used in tracing the family tree for format information.

Output Description: ERR = integer variable; returned non-zero if any errors were encountered.

Process Description:

1. The input record ID, RECID, is moved to variable ID. U7 is initialized to whatever unit has been designated as the message file, and the data base top level indicator, TLF, is set to zero.
2. The pointer, NMTCT, to the last used row of the Move Table Control Table, MTCT in /SY3COM/, is incremented by one.
3. The next available row number of the Multilevel Move Table, MLMT in /SY3COM/, is stored in column one of row NMTCT of MTCT.
4. Subroutine LOCREC is used to locate the pointer to the record with accession number = ID. If the pointer is found, go to step 7.

5. Write "Record not in data base" on unit U7.
6. Set ERR = 1 and return to calling routine.
7. Use GET to retrieve the pointer to the record, and use GET again to retrieve the format number of the record from the second word of the record (characters 5-8). Convert the format number character string to a binary integer via INPARM, and store the format number in FMTID(1) of /SY2COM/ and in column 3 of row FMTCT of MTCT.
8. Use LODFMT to retrieve the format whose number is in FMTID(1) and store the format in page 1 of FMT in /SY2COM/. If the format was not found in the data base, go to step 6.
9. Calculate the last used row, N, of page 1 of FMT as FMT(6, 1, 1) +1.
10. For each field name in column 2 of the Working Buffer Format, WBF in /SY3COM/, which is not \$R, \$T, or \$L, compare that name against the names up to row N in column 2 of page 1 of FMT, and if a match is found, do steps 11-21, otherwise just move to the next name in WBF until they are all processed, and then go to step 22.
11. If the row number, I, of WBF where the match was found, = -TBF(4,1), then negate TBF(4,1) and go to step 13. TBF is the Target Buffer Format in /SY3COM/, and TBF(4,1) contains the negative of the row number of the field whose name is "UNLOAD". This value was placed in TBF (4,1) by subroutine BLDTBF when a target field type of 4 was encountered, meaning a special output conversion was desired which depended on the contents of the "UNLOAD" field.
12. If I = -TBF(4,2), then negate TBF(4,2). This is the row number of the "LSD"



- field which is similar to the "UNLOAD" field in step 11.
13. The length of the field is transferred from column 4 of FMT to WBF(6,I).
  14. The type of the field is transferred from column 5 of FMT to WBF(7,I).
  15. The pointer, NSBF, to the last used row of the Source Buffer Format, SBF of /SY3COM/, is incremented by one.
  16. The key field indicator is transferred from column 6 of FMT to SBF(1, NSBF).
  17. The starting character number is transferred from column 3 of FMT to SBF(5, NSBF).
  18. The length of the field is transferred from column 4 of FMT to SBF(6, NSBF).
  19. The type of field is transferred from column 5 of FMT to SBF(7, NSBF).
  20. The row number within FMT of the field is stored in SBF(2, NSBF) for later use in the Change Field command.
  21. The pointer, NMLMT, to the last used row of MLMT is incremented by one. The value of NSBF is stored in MLMT(1, NMLMT), and the value of I is stored in MLMT(2, NMLMT).
  22. After processing all fields of WBF that were found in the format for records at this data base level, calculate the number of rows of MLMT which were generated ( $=NMLMT - MTCT(1, NMTCT) + 1$ ) and store it in MTCT(2, NMTCT).
  23. Check column 4 of all fields of WBF. If any lengths are still = 0, then more formats need to be examined if possible, so go to step 24, otherwise set TBF(4,1) and TBF(4,2) to zero if they are still negative and return to the calling routine.
  24. If  $NMTCT \neq NLVLS$ , go to step 26.

25. Write "Unidentified field(s)" on unit U7, and go to step 6.
26. If TLF  $\neq$  0, then go to step 25, otherwise use PRNTID to get the next level record ID and go to step 2.

Name: JFDFCR

Purpose: To direct the overall processing sequence for the JF and DF commands.

Linkage:

- Calling sequence: CALL JFDFCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, LODFMT, CICPDF, BLDTBF, SETIN1, XXIN1, FTFMT, FTCMP, GETREC, TFORMW, PRNTID, EXCMDS, TFORMZ, DISFMT.
- Files used: Message file (logical unit 7)

Input Description: CIND = integer variable; command indicator. Set zero for DF and non-zero for JF.

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/.
2. Use AEINIT to initialize /SY3COM/, returning the input set number in SETNO, the input format number in FMTNO, and error indication of non-zero in ERR.
3. If ERR = 0, go to step 5.
4. Write on U7, "Command terminated due to syntax error.", and return to the calling routine.
5. Store FMTNO in FMTID(2) and use LODFMT (2,HIT) to load the format into page 2 of array FMT in /SY2COM/, returning HIT as zero if the format could not be found in the data base.
6. If HIT = 0, then write on U7, "Format not found. ", and return to the calling routine.

7. If `COMMAS(1) = 3`, meaning there were no clauses in the input command, then go to step 10.
8. Use `CICFDF` to process the clauses in the input command, returning `ERR` non-zero if any errors were found.
9. If `ERR ≠ 0`, go to step 4.
10. Use `BLDTBF(2)` to convert the data base format in page 2 of `FMT` to a standard format in the Target Buffer Format, `TBF` in `/SY3COM/`.
11. Use `SETIN1` to initialize the input set for record ID's to be returned by `XXIN1`. Use `XXIN1(RID)` to return the first record ID in `RID`.
12. If `RID = 0`, meaning there were no records in the input set, return to the calling routine.
13. Set `NL`, the number of levels in the data base to be used, to 1 or 2, depending on whether `CIND = 0` or `≠ 0`, respectively. Then use `FTFMT(RID, NL, ERR)` to complete the buffer formats with information about fields whose names occurred in the input command, returning `ERR` non-zero if an error occurred.
14. If `ERR ≠ 0`, then go to step 4.
15. Use `FTCMP` to complete starting location information in the Working Buffer Format, `WBF` in `/SY3COM/`.
16. Initialize `I`, the pointer to the desired row of `MTCT`, the Move Table Control Table in `/SY3COM/`, to 1.
17. Transfer the format number for records at this data base level from `MTCT(3,I)` to `FMTID(1)`. This prevents an unnecessary retrieval of the format record by `GETREC`.

18. Use GETREC(1, RID, STAT) to retrieve record RID into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.
19. If STAT  $\neq$  0, ignore this record by going to step 27.
20. Use TFORMW(1,I) to transfer data from row 1 of BUF to WBUF, the Working Buffer in /SY3COM/, as directed by row I of MTCT.
21. If  $I \geq$  NMTCT, the last used row of MTCT, then go to step 23.
22. Increment I by 1, use PRNTID to get record RID's parent record, PID, set RID to PID, and go back to step 17.
23. Set EF = 1 and use EXCMDS to execute the commands in CTBL, the Normal Command Table in /SY3COM/, returning CFLAG as false if any of the relational clauses failed to be true.
24. If CFLAG is false, go to step 27.
25. Use TFORMZ(2,1) to transfer data from WBUF to row 2 of BUF.
26. Use DISFMT to display the record in row 2 of BUF according to the format in FMTID(2).
27. Use XXIN1 (RID) to retrieve the next record in RID.
28. If RID  $\neq$  0, meaning there was a next record, go back to step 16.
29. Return to the calling routine.

Name: JNSNCR

Purpose: To direct the overall processing sequence for the JN and SN commands.

Linkage:

- Calling sequence: CALL JNSNCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, RLCLPR, SETIN1, SETOUT, XXIN1, FTFMT, FTCMP, GETREC, TFORMW, PRNTID, EXCMDS, XXOUT, ENDSET
- Files used: Message file (logical unit 7), pointer lists file (logical unit 5)

Input Description: CIND = integer variable; command indicator. Set zero for SN and non-zero for JN.

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/. Initialize the comma array pointer, CP, to 2. Initialize the number of records selected, HITS, to 0.
2. Use AEINIT to initialize /SY3COM/, returning the input set number in SETNO, and returning ERR non-zero if any errors were found.
3. If ERR = 0, go to step 5.
4. Write on U7, "Command terminated due to syntax error.", and return to the calling routine.
5. If COMMAS(1) < 3, meaning that there were no relational clauses input, go to step 4.

6. Calculate FC, the first character of the relational clause, = COMMAS(CP)+1. Calculate NC, the number of characters in the relational clause, = COMMAS(CP+1)-FC. Then use RLCLPR(FC,NC,ERR) to process the relational clause, building buffer formats and commands to be executed, and returning ERR non-zero if any errors were found.
7. If ERR  $\neq$  0, go to step 4.
8. Increment CP by 1. If CP  $\neq$  COMMAS(1), go back to step 6.
9. Use SETIN1 to initialize the input set for record ID's to be returned by XXIN1. Use SETOUT to initialize file U(5) to receive selected record ID's.
10. Use XXIN1 (RID) to return the first record ID in RID.
11. If RID = 0, meaning there were no records in the input set, go to step 27.
12. Set NL, the number of levels of the data base to be used, to 1 or 2, depending on whether CIND = 0 or  $\neq$  0, respectively. Then use FTFMT(RID, NL, ERR) to complete the buffer formats with information about fields whose names occurred in the input command, returning ERR non-zero if an error occurred.
13. If ERR  $\neq$  0, go to step 4.
14. Use FTCMP to complete starting location information in the Working Buffer Format, WBF in /SY3COM/.
15. Initialize I, pointer to the desired row of MTCT, the Move Table Control Table in /SY3COM/, to 1. Set R = RID.
16. Transfer the format number for records at this data base level from MTCT (3,I) to FMTID(1). This prevents an unnecessary retrieval of the format record by GETREC.

17. Use GETREC(1, R, STAT) to retrieve record R into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.
18. If STAT  $\neq$  0, ignore this record by going to step 25.
19. Use TFORMW(1,I) to transfer data from row 1 of BUF to WBUF, the Working Buffer in /SY3COM/, as directed by row I of MTCT.
20. If  $I \geq$  NMTCT, the last used row of MTCT, then go to step 22.
21. Use PRNTID to get record R's parent record, PR. Then set R = PR, increment I by 1, and go back to step 16.
22. Set EF = 1 and use EXCMDS to execute the commands in CTBL, the Normal Command Table in /SY3COM/, returning CFLAG as false if any of the relational clauses failed to be true.
23. If CFLAG is false, go to step 25.
24. Increment HITS by 1, and use XXOUT (RID) to store the selected record ID on file U(5).
25. Use XXIN1(RID) to return the next record ID in RID.
26. If RID  $\neq$  0, meaning there is a next record, go back to step 15.
27. Use ENDSET (HITS, U(5)) to create and display an entry in the status table of sets.
28. Return to the calling routine.



Name: JPRPCR

Purpose: To direct the overall processing sequence for the JP and RP commands.

Linkage:

- Calling sequence: CALL JPRPCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM
- Subroutines or functions used: AEINIT, SETIN1, XXIN1, CIRP, GETREC, TFORMW, PRNTID, EXCMDS, COMSTR, SUBSTR, TFORMZ
- Files used: Message file (logical unit 7), Report file (logical unit 12).

Input Description: CIND = integer variable; command indicator. Set zero for RP command and non-zero for JP command.

Output Description: None

Process Description:

1. Initialize file pointers U7 and U12 to the values stored in U(7) and U(12) of /SYSCOM/. Initialize first and last record indicator, FLREC, to zero.
2. Use AEINIT to initialize values in common, return the input set number in SETNO, and return an error indicator, ERR, non-zero if any errors were found.
3. If ERR = 0, go to step 6.
4. Write on U7, "Command terminated due to syntax error."
5. Return to the calling routine.
6. Use SETIN1 to initialize set number SETNO for returning record ID's via XXIN1.
7. Use XXIN1 to return the first record ID, RECID, from the input set.

8. If  $RECID = 0$ , then write on U7, "Null Input Set." and go to step 5.
9. Use  $CIRP(CIND, RECID, ERR)$  to interpret the command, build tables and buffer formats, and return  $ERR$  non-zero if any errors were found.
10. If  $ERR \neq 0$ , go to step 4.
11. Initialize the Move Table Control Table pointer,  $MTCTP$ , to one.
12. Transfer the format number for this level of the data base from  $MTCT(3, MTCTP)$  to  $FMTID(1)$ . This prevents actual retrieval of the format record by  $GETREC$ , since it is not needed.
13. Use  $GETREC(1, RECID, STAT)$  to get record  $RECID$  into row 1 of  $BUF$  in  $/SY2COM/$ , returning  $STAT$  non-zero if a problem occurred.
14. If  $STAT \neq 0$ , ignore this  $RECID$  by going to step 28.
15. Use  $TFORMW(1, MTCTP)$  to transfer data from row 1 of  $BUF$  to  $WBUF$ , the Working Buffer in  $/SY3COM/$ , based on the directions provided by row  $MTCTP$  of  $MTCT$ .
16. If  $MTCTP \geq$  last used row of  $MTCT$ ,  $NMTCT$ , then go to step 18.
17. Use  $PRNTID$  to get the record ID,  $PID$ , of the parent of  $RECID$ . Store  $PID$  into  $RECID$ , increment  $MTCTP$  by one, and go back to step 12.
18. Initialize the BY Processing Table pointer,  $BPTP$ , to one.
19. Get the Grouping Field Name pointer,  $GFN$ , from column 1 of row  $BPTP$  of the BY Processing Table,  $BPT$  of  $/SY3COM/$ .
20. If  $GFN = 0$ , go to step 33.
21. If  $GFN > 0$ , go to step 24.

22. Negate GFN, set ERR = 0, and use EXCMD5 to execute the commands in CTBL, the Normal Command Table of /SY3COM/, as specified by columns 2 and 3 of row BPTP of BPT, returning ERR non-zero if the commands could not be executed for some reason.
23. If ERR  $\neq$  0, assume no change in this GFN, and go to step 25.
24. Use COMSTR to compare the new GFN in WBUF to the current GFN in column 6 of BPT. If they are different, go to step 49.
25. If BPTP  $\geq$  last used row of BPT, NBPT, then go to step 27.
26. Increment BPTP by 1 and go back to step 19.
27. Set ERR = 0 and use EXCMD5 to execute all the commands stored in the Function Command Table, FCTBL in /SY3COM/.
28. Use XXIN1 to get the next record ID into RECID.
29. If RECID  $\neq$  0, meaning there is another record to be processed, then go back to step 11.
30. Set FLREC = 2 to mean that the last record is being processed.
31. Set BPTP = 1 and get GFN from column 1 of row 1 of BPT.
32. If GFN < 0, negate GFN.
33. If FLREC = 0, meaning we are processing the first record, then set FLREC = 1 and go to step 39.
34. Initialize the local BPT pointer, LBPTP, to the current value of BPTP.
35. If BPT(1, LBPTP) = 0, go to step 38.
36. Transfer all function results and concluding text from this BY level by setting PFLAG = 2 \* LBPTP and

- calling TFORMZ(2, PFLAG) to transfer from WBUF to row 2 of BUF.
37. If LBPTP < last used row of BPT, NBPT, then increment LBPTP by 1 and go back to step 35.
  38. Write to U12 the first 120 characters of row 2 of BUF. If FLREC = 2, meaning we were processing the last record, then return to the calling routine.
  39. Blank out the first 120 characters of row 2 of BUF.
  40. Initialize LBPTP = BPTP.
  41. Set ERR = 1 and use EXCMDS to execute the commands of CTBL specified by columns 4 and 5 of row LBPTP of BPT.
  42. Transfer from WBUF to row 2 of BUF field values, calculations, and beginning text by setting PFLAG = 2 \*LBPTP-1 and calling TFORMZ(2, PFLAG).
  43. If LBPTP  $\geq$  NBPT, then go to step 27.
  44. Increment LBPTP by 1 and get GFN from BPT(1, LBPTP).
  45. If GFN = 0, go to step 41.
  46. If GFN > 0, go to step 48.
  47. Negate GFN, set ERR = 0, and use EXCMDS to execute the commands of CTBL specified by columns 2 and 3 of row LBPTP of BPT. If ERR is returned non-zero, go to step 41.
  48. Use SUBSTR to store the new value of GFN from WBUF to column 6 of row LBPTP of BPT, and then go to step 41.
  49. Use SUBSTR to store the new value of GFN from WBUF to column 6 of row BPTP of BPT, and then go to step 33.

Name: PRNTID

Purpose: To return the record ID of the next higher level record in the same family tree of an inverted tree logically structured data base.

Linkage:

- Calling sequence: CALL PRNTID (CID, PID, TLFLAG)
- Common blocks used: None
- Subroutines or functions used: None
- Files used: None

Input Description: CID = integer variable; child record ID.

Output Description: PID = integer variable; parent record ID.  
TLFLAG = integer variable; set zero or non-zero depending on whether output parent ID is not or is at the top level of the data base, respectively.

Process Description: Coded specifically for ASATS, the child record ID consists of the segment number concatenated with the acquisition date. To get the parent record ID, the acquisition date portion (lower 16 bits) is set to zero. TLFLAG is set to 1 since ASATS parent records are at the top level of the data base.

Name: RLCLPR

Purpose: To parse a relational clause of the form AE.OP.AE (where AE is an arithmetic expression, and OP is a comparison operator) and build a table of commands to evaluate the clause.

Linkage:

- Calling sequence CALL RLCLPR (FC, NC, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: COMSTR, INDEX, ADDLT, CFIND, AEPR, ADDNM, ADDDT, ADDFN
- Files used: None

Input Description: FC = integer variable; first character number of the string to be processed in array CMD of /SY3COM/.  
NC = integer variable; number of characters in the string to be processed.

Output Description: ERR = integer variable; returned zero if no errors are found, non-zero if an error is found.

Process Description:

1. Initialize ERR = 0, F = FC, N = NC, and K = 0.
2. If the character at F is not a single quote mark, go to step 11.
3. Use INDEX to find the next quote mark at J.
4. If  $J \neq 0$ , go to step 6.
5. Set ERR = 1 and return to the calling routine.
6. If  $J \leq F+1$ , go to step 5.
7. Use ADDLT (F, J-F+1), V(1)) to add the literal to the Working Buffer Format, WBF in /SY3COM/, getting the row number of WBF returned in V(1).
8. Set K = 1 to indicate that the left hand side of the relational clause has been processed.
9. Recalculate the number of characters remaining,  $N, = N - (J-F+1)$ .

10. Reset the first character pointer, F,  
= J+1.
11. Use INDEX to find the first period in N  
characters beginning at F and store  
the location in I.
12. If I = 0, go to step 5.
13. Use COMSTR to compare the four characters  
that start at I with an internal table of  
legal operators. If a match is found, then  
J is set to the row number of the internal  
table, otherwise go to step 5.
14. The actual operator number, OP, to eventually  
be stored in the command table is calculated  
by adding 4 to J.
15. J is initialized to 1.
16. If K = 0, go to step 19.
17. If I  $\neq$  F, go to step 5.
18. J is reset to 2, F is incremented by 4 to  
set it past the operator, N is decremented  
by 4 to account for the operator characters,  
and a jump to step 20 is made.
19. N is set to the number of characters to the  
left of the operator by setting it equal to  
I-F.
20. If  $N \leq 0$ , go to step 5.
21. If the character at F is not a single quote  
mark, go to step 28.
22. If  $N \leq 2$ , go to step 5.
23. If the character at F+N-1 is not a single  
quote mark, go to step 5.
24. Use ADDLT (F, N, V(J)) to add the literal to  
WBF and receive the row number in V(J).
25. If J = 2, go to step 27.
26. Set J = 2, F = I+4, and N = NC-N-4 to adjust  
to the right hand side of the operator, and  
go back to step 20.
27. Increment NCTBL, the pointer to the last  
used row of the Normal Command Table, CTBL in

- /SY3COM/, by 1. Store V(1) in CTBL (1, NCTBL), OP in CTBL(2, NCTBL), V(2) in CTBL (3, NCTBL), NCTBL+1 in CTBL (5, NCTBL), and return to the calling routine.
28. Use CFIND to locate any arithmetic operator, storing its location in K.
  29. If K = 0, go to step 33.
  30. Increment NWBF, the pointer to the last used row of WBF, by 1. Store NWBF in V(J), 0 in WBF(1,NWBF), \$R in WBF(2,NWBF), 4 in WBF (6, NWBF), and -1 in WBF (7, NWBF).
  31. Use AEPR (F, N, V(J), ERR) to process the arithmetic expression, building commands in CTBL which store a result at V(J) of WBF, and returning ERR non-zero if any errors were found.
  32. If ERR  $\neq$  0, go to step 5, otherwise go to step 25.
  33. Use CFIND to determine if the character at F is the number sign (K will be returned = 1 and L will be returned = 8) or a numeric character (K will be returned = 1 and L will be  $>8$ ).
  34. If K = 0 (implying a field name), go to step 38.
  35. If L  $\neq$  8 (implying a numeric literal), use ADDNM (F, N, V(J)) to add the number to WBF, receiving the row number back in V(J), and go to step 25.
  36. If N  $\neq$  5, go to step 5.
  37. Use ADDDT (F, N, V(J)) to add the date to WBF, receiving the row number in V(J), and go to step 25.
  38. If N  $>12$ , go to step 5.
  39. Use ADDFN (F, N, V(J)) to add the field name to WBF if necessary, receiving the row number back in V(J), and go to step 25.



Name: RPCLPR

Purpose: To parse a replacement clause of the form FN = AE (where FN is a field name and AE is an arithmetic expression) and build a table of commands to perform the replacement.

Linkage:

- Calling sequence: CALL RPCLPR (FCS, TNC, LOCEQL, IND, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: INDEX, ADDFN, COMSTR, ADDLT, CFIND, ADDDT, AEPR, ADDNM
- Files used: None

Input Description:

FCS = integer variable; first character number of the string to be processed in array CMD of /SY3COM/.

TNC = integer variable; total number of characters in the string to be processed.

LOCEQL = integer variable; the character number which is the location of the equal sign in the input string. If zero, the equal sign will be searched for internally.

IND = integer variable; command indicator to allow special processing for different commands. A value of zero means the Change Field command is being processed. A non-zero value means the Display Formatted or Joint Display Formatted command is being processed.

Output Description: ERR = integer variable; set to zero if no errors are found and set to non-zero if an error is found.

Process Description:

1. If  $LOCEQL \leq 0$ , then INDEX is used to find the equal sign and its character position is stored in I, otherwise I is set = LOCEQL.
2. If the equal sign is located, go to step 4.

3. Set ERR = 1 and return to the calling routine.
4. Initialize the first character pointer, FC, to FCS.
5. Calculate the number of characters, NC, in the field name = I-FC.
6. If  $NC \leq 0$ , go to step 3.
7. If  $NC > 12$ , go to step 3.
8. Use ADDFN (FC, NC, K) to add the field name to the Working Buffer Format, WBF in /SY3COM/, if it is not already there, and receive back the row number of WBF in K.
9. If IND = 0, then increment NTBF, the pointer to the last used row of TBF, the Target Buffer Format in /SY3COM/, by one, store a one in TBF (1, NTBF) and store K in TBF (2, NTBF).
10. Move the first character pointer, FC, to the first character past the equal sign by setting  $FC = I+1$ . Calculate NC, the number of characters to the right of the equal sign, by  $NC = TNC-I$ .
11. If  $NC \leq 0$ , go to step 3.
12. If the character at FC is not a single quote mark, go to step 17.
13. If  $NC < 3$ , go to step 3.
14. Use ADDLT (FC, NC, L) to add the text literal to WBF and receive the row number back in L.
15. Increment NCTBL, the pointer to the last used row in CTBL, the Normal Command Table in /SY3COM/, by one. Store L in CTBL (1, NCTBL), 16 in CTBL (2, NCTBL), K in CTBL (4, NCTBL), and NCTBL+1 in CTBL (5, NCTBL).
16. Return to the calling routine.
17. Use CFIND to determine if the character string to the right of the equal sign is an arithmetic expression by locating any +, -, \*, /, (, or), and pointing to it with I.

18. If  $I \neq 0$ , then call AEPR (FC, NC, K, ERR) to process the arithmetic expression, creating commands in CTBL to evaluate the expression and store the result in K, returning ERR as non-zero if any errors were found, otherwise go to step 21.
19. If  $ERR \neq 0$ , go to step 3.
20. Return to the calling routine.
21. Check the character at FC for the number sign or a numeric character via CFIND.
22. If neither was found, the right side of the equal is assumed to be a field name. If  $NC > 12$ , then go to step 3, otherwise call ADDFN (FC, NC, L) and go to step 15.
23. If character FC is a number sign, then a date literal is expected. If  $NC \neq 5$ , then go to step 3, otherwise call ADDDT (FC, NC, L) to add the date literal to WBF and go to step 15.
24. If character FC is a numeric character, then a numeric literal is expected, and ADDNM (FC, NC, L) is called to add it to WBF. Then go to step 15.

Name: STAEPR

Purpose: To store arithmetic processing data into the normal command table in mathematic hierarchical order.

Linkage:

- Calling sequence: Call STAEPR (VTAB, OPC, FIRST, LAST, TREG)
- Common blocks used: SY3COM
- Subroutines or functions used: None
- Files used: None

Input Description:

VTAB = integer array; contains pointers to variables or literals in the working buffer format table, intermediate storage registers or special integers representing close or open brackets.

OPC = integer array; contains either mathematical operator indicators or special integers representing close or open brackets.

FIRST = integer variable; pointer to first variable in VTAB and OPC to be used for processing.

LAST = integer variable; pointer to last variable in VTAB & OPC to be used in processing.

TREG = integer variable; index pointer into intermediate storage register buffer used for intermediate data storage.

Output Description: Normal command table filled with appropriate arithmetic processing data. TREG updated as intermediate storage registers are needed.

Process Description:

A loop is set up to search the entries in the OPC table. Steps 1 thru 3 performed for all entries.

1. The intermediate storage register pointer (TREG) is updated. The OPC entry for the next two adjacent locations is checked for mathematical hierarchy. If they are of equal hierarchy or if the first is of a lesser hierarchy, step 2 is performed, otherwise step 3 is performed.
2. The next normal command table entry is loaded with values from the current and next entry of VTAB, the current value of OPC and the intermediate storage register pointer (TREG). Then the next entry of VTAB is loaded with the intermediate storage register pointer (TREG) and return to step 1.
3. The next normal command table entry is loaded with values from the next and next +1 entry of VTAB, the next +1 value of OPC and the intermediate storage register pointer (TREG). The next +1 entry of VTAB & OPC is loaded with the current value of VTAB & OPC respectively. The next +2 entry of VTAB is loaded with the intermediate storage register pointer (TREG) and return to step 1.

Name:

SQZE

Purpose:

To delete extraneous blanks from a character string and build an array of pointers to the commas in the character string.

Linkage:

- Calling sequence: CALL SQZE (INARY, INST, INLEN, OUTARY, OUTST, OUTLEN, COMMAS)
- Common blocks used: None
- Subroutines or functions used: None
- Files used: None

Input Description:

INARY = integer array name; starting location of the array containing the input string  
INST = integer variable; character number of INARY at which to begin processing  
INLEN = integer variable; number of characters in INARY to be processed  
OUTST = integer variable; character number of OUTARY at which to begin storing output  
COMMAS = integer array name; contains the Comma Location Table. The first word contains the number of the last used word in the array and should be input containing the value one upon the first call within any one command.

Output Description:

OUTARY = integer array name; starting location of the array to contain the output

OUTLEN = integer variable; number of characters stored in OUTARY

COMMAS = integer array name; contains the Comma Location Table. The first word contains the number of the last used word in the array. The other words contain the character number of OUTARY where commas occur (exclusive of those commas occurring between

pairs of quote marks). The first word will be output as a negative value when an exclamation mark has been encountered and stored as a terminating comma for the command.

Process Description:

1. Counters and pointers are initialized.
2. If the last character of INARY has been passed, go to step 10.
3. If processing is between quote marks (QSET = 2 or 3), check this character for the terminating quote mark, reset QSET to 1 if it is, and go to step 8.
4. If the character is a blank, go to step 9.
5. If the character is a quote mark, set QSET (= 2 for single quote, = 3 for double quote) and go to step 8.
6. If the character is a comma, store the OUTARY pointer in the next available location in COMMAS, increment the pointer to the last used word of COMMAS, and go to step 8.
7. If the character is an exclamation mark, store the OUTARY pointer in the next available location in COMMAS, increment and negate the pointer to the last used word of COMMAS, store a comma in OUTARY, increment the OUTARY pointer, and go to step 10.
8. Transfer the character to OUTARY and increment the OUTARY pointer.
9. Increment the INARY pointer and go to step 2.
10. Store the pointer to the last used word of COMMAS into word one of COMMAS, calculate OUTLEN as the OUTARY pointer minus OUTST, and return to the calling program.

Name: TFORMW

Purpose: To transfer data from a source buffer to the Working Buffer.

Linkage:

- Calling sequence: CALL TFORMW(ROW, MTCTRW)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: SUBSTR
- Files used: None

Input Description:

ROW = integer variable; the row number of the source buffer, BUF in /SY2COM/, where the input data is stored.

MTCTRW = integer variable; the row number of the Move Table Control Table, MTCT in /SY3COM/, to be used for control.

Output Description: None

Process Description: The starting row of the Multilevel Move Table, MLMT in /SY3COM/, is retrieved from MTCT (1, MTCTRW). The number of rows of the MLMT to use is retrieved from MTCT (2, MTCTRW) and used to calculate the final row number. Then for each of these rows, (1) a pointer for the Source Buffer Format, SBF in /SY3COM/, is retrieved from the first word of the row of MLMT, (2) a pointer for the Working Buffer Format, WBF in /SY3COM/, is retrieved from the second word of the row of MLMT, and (3) SUBSTR is used to transfer the data from BUF to WBUF, the Working Buffer in /SY3COM/. After the specified number of rows have been processed, a return is made to the calling routine.



Name: TFORMZ

Purpose: To transfer data from the Working Buffer to a target buffer, converting the data representation when needed.

Linkage:

- Calling sequence: CALL TFORMZ(ROW, PF)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: SUBSTR, DTEINT, CHAR, COMSTR
- Files used: None

Input Description: ROW = integer variable; the row number of the target buffer, BUF in /SY2COM/, where the data is to be stored.  
PF = integer variable; indicator for which fields of the target buffer are to be filled from the Working Buffer, WBUF in /SY3COM/. A field is filled if word one of its Target Buffer Format, TBF in /SY3COM/, is equal to PF.

Output Description: None

Process Description: For each row of TBF, the following process is done, and then a return is made to the calling routine:

1. If column 1 of TBF is not equal to PF, ignore this row and go to step 25.
2. Retrieve F, the pointer to the WBF row number, from column 2 of TBF.
3. If the output field type, column 5 of TBF, is > 3, go to step 9.
4. If column 2 of row F of WBF indicates text type (by \$T), then use SUBSTR to transfer the text from the command line array, CMD in /SY3COM/, to BUF and then go to step 25.

5. If column 2 of row F of WBF does not indicate a results field (by \$R), then go to step 7.
6. If column 5 of row F of WBF does not indicate text type (contains non-zero), then go to step 8.
7. Use SUBSTR to transfer the data from WBUF to BUF and if WBUF was a results field, then use SUBSTR to reinitialize WBUF from the first word of row F of WBF and go to step 25, otherwise just go to step 25.
8. Use SUBSTR to transfer the data in WBUF to an integer variable named RESULT and based on the target type (column 5 of TBF), use CHAR (type = 1) or DTEINT (type = 2) to convert RESULT to a character string in BUF. Then reinitialize WBUF from the first word of row F of WBF and go to step 25.
9. Extract the first character of the field in WBUF and store this character in CRDTYP.
10. If the target field type  $\neq$  5, go to step 12.
11. Search the array FLMTYP until a match with CRDTYP is found at element L. If no match is found, set L = 8. Store the 12 characters of row L of table FLMTAB into the field in the target buffer and go to step 25.
12. If the target field type  $\neq$  4, go to step 23.
13. Search the array CMPTYP until a match with CRDTYP is found at element L. If no match is found, set L = 11.
14. If L < 5 or = 11, go to step 22.
15. If L > 7, go to step 19.
16. Extract the pointer to the "UNLOAD" field from TBF(4,1) and store it in K.

17. If  $K = 0$ , go to step 22.
18. If the "UNLOAD" field is non-blank, then increment  $L$  by 3 and go to step 22, otherwise go to step 22.
19. Extract the pointer to the "LSD" field from  $TBF(4,2)$  and store it in  $K$ .
20. If  $K = 0$ , then set  $L = 11$  and go to step 22.
21. Use SUBSTR to transfer the contents of the "LSD" field from WBUF to the target field and go to step 25.
22. Use SUBSTR to store the 12 characters of row  $L$  of table CMPTAB into the field in the target buffer and go to step 25.
23. If the target field type  $\neq 9$ , go to step 7.
24. Search the array GCMTYP until a match with CRDTYP is found at element  $L$ . If no match is found, set  $L = 1$ . Use SUBSTR to store the 12 characters of row  $L$  of table GCMTAB into the field in the target buffer.
25. Move to the next row of TBF and start over at step 1.

Name: TJUMP

Purpose: To eliminate headers and other data associated with a null set.

Linkage:

- Calling sequence: CALL TJUMP
- Common blocks used: SYSCOM, SY2COM
- Subroutines or functions used: INDEX, SUBSTR, INPARM
- Files used: None

Input Description: A command line containing the set to be checked.

Output Description: None

Process Description: The set in question is checked to see if it contains data. If it does contain data no action is required and the routine exits. If there is no data in the set, the label from the command line is saved for later use. Next the input file is read until a label card containing "LA" followed by the label saved from the JT command line is found. The routine then exits.

### 4.3 DESCRIPTION OF SUBROUTINE MODIFICATIONS

This section contains all subroutines requiring minor modifications. Those requiring major redesign are in 4.2

Name: CHAR

Modification Purpose: To allow conversion of negative numbers.

Linkage Modification:

- Calling sequence: No change
- Common blocks used: No change
- Subroutines or functions used: The Fortran function IABS is now used.
- Files used: No change

Input Description Modification: V = integer variable; may now contain values less than zero.

Output Description Modification: STR = integer array name; leftmost position will contain the minus character if the input value in V was negative.

Process Description Modification: Convert the absolute value of V to a character string by the original process. Then, if V is negative, store the minus character as the leftmost character of the output field.

Name:

JLASYS

Modification Purpose:

To allow the following new capabilities:

1. Data base protection for Delete Set, Delete Record, Delete Key and No Key commands.
2. Null set detection and control transfer.
3. Joint Sort, Joint Select Non-Key, Report and Joint Report commands.

Linkages Modification:

No change

Input Description Modification:

No change

Output Description Modification:

No change

Process Description Modification:

1. When the input command JT is encountered call TJUMP subroutine.
2. When the input command LA is encountered treat it as a NOP and process next input command.
3. When either of the commands DK, DR, DS or NK is encountered, call DBPRO. If DBPRO returns a flag = 0 do not execute the command. If DBPRO returns a flag not = 0 process the command as before.
4. When the JS command is encountered set SFLAG = 2 and call the SORTP subroutine.
5. When the SO command is encountered set SFLAG = 1 and call the SORTP subroutine.
6. When the "JN" or "SN" command is encountered call the JNSNCR subroutine.

7. When the "RP" or "JP" command is encountered call the JPRPCR subroutine.
8. When the "JF" or "DF" command is encountered call the JFDFCR subroutine.



Name: SEL (Main program)

Modification Purpose: To echo 80 characters of the input command instead of 40 to the message file (unless it is the terminal).

Linkage Modification:

- Calling sequence: No change
- Common blocks used: No change
- Subroutines or functions used: No change
- Files used: No change

Input Description Modification: No change

Output Description Modification: No change

Process Description Modification: When the WRITE statement with array X in it is executed, the entire array (20 words) is written instead of just the first 10 words.

Name:

SORTS

Modification Purpose:

To allow the user to order a set of FLOCON records based upon the contents of fields in either the FLOCON or DAPTS records for that set.

Linkage Modification:

- Calling Sequence: CALL SORTS (SET, NF, LIST, SF)
- Common Blocks used: No change
- Subroutines or functions used: No change
- Files used: No change

Input Description Modification:

SF: integer value, where SF =1 means sort on FLOCON data, SF =2 means sort on either FLOCON or DAPTS data.

Output Description Modification:

No change

Process Description Modification:

For SF = 2 only, the DAPTS record for each appropriate FLOCON record is retrieved. Next the formats for DAPTS or FLOCON records are loaded as needed, a table of sorting names is loaded in proper hierarchical order and a buffer pointer table is also built to point to the proper buffer for data retrieval. Lastly, the data retrieval section is altered to get data from the appropriate buffer by use of the buffer pointer table as an index.

Name: SORTP

Modification Purpose: To pass an argument to SORTS to indicate the type of sort to perform.

Linkage Modification:

- Calling sequence: CALL SORTP(SF)
- Common blocks used: No change
- Subroutines or functions used: No change
- Files used: No change

Input Description Modification: SF = integer variable, indicates which type of sort SORTS is to perform. 1 = normal sort, 2 = joint sort.

Output Description Modification: None

Process Description Modification: The argument SF received from JLASYS is passed to SORTS to allow SORTS to perform the appropriate type of sort.

Name:

SPCSET

Modification Purpose:

To stop the input process when an end-of-file is read as well as a zero record ID.

Linkage Modification:

- Calling sequence: No change
- Common blocks used: No change
- Subroutines or functions used:  
No change
- Files used: No change

Input Description Modification:

No change

Output Description Modification:

No change

Process Description Modification:

Insert an end-of-file branch to statement number 3 into the statement that reads from the data file.

#### 4.4 DESCRIPTION OF NEW SYSTEM TABLES

This section describes the common blocks and tables used in conjunction with implementing arithmetic operators.

## BUFFER FORMATS

General layout for Source Buffer Format (SBF), Working Buffer Format (WBF), and Target Buffer Format (TBF).

Column 1 - 1 word - SBF: temporary key field indicator for CF command

WBF: the value to be used for initialization after printing of a results field

TBF: print flag to associate printing of this field with a change of a BY field (BY processing table row number N)

Print flag =  $2*N-1$  means print this field at top of BY number N

Print flag =  $2*N$  means print this field at bottom of BY number N

Column 2 - 3 words - SBF: first word is data base format row number of key field for CF command.

WBF: four types of data: (1) alphanumeric characters representing field names, (2) \$Lbb in first word for integer literal in command line, (3) \$Tbb in first word for alphanumeric literal in command line, (4) \$Rbb in first word for calculation results

TBF: First word is row number of WBF of desired output field. Second word is key field indicator for CF command. Third word is data base format row number of key field for CF command.

Column 3 - 1 word - SBF, WBF, TBF: starting character for actual value in buffer being used. (In WBF, the data is actually in the command line instead of the working buffer if column 2 = \$Tbb)

Column 4 - 1 word - SBF, WBF, TBF: length of field (in characters)

Column 5 - 1 word - SBF, WBF, TRF: type of data in the field:

-1 means a binary integer contained in 4 characters

0 means an alphanumeric character string

1 means an integer in a numeric character string

2 means a date in YDDD numeric character string format

## BY PROCESSING TABLE

Each successive row of this table defines a successively lower level subgroup of the input data and the processing associated with a change at that subgroup level.

Column 1 - 1 word - binary integer; index to Working Buffer Format (i.e., row number) pointing to the Grouping Field Name (GFN). If 0, it means the GFN was E&E. If  $<0$ , then a calculation must be performed before a test for the BY change can be made.

Column 2 - 1 word - binary integer; starting row number of normal command table when column 1 is  $<0$ .

Column 3 - 1 word - binary integer; number of rows of normal command table to be processed when column 1 is  $<0$ .

Column 4 - 1 word - binary integer; starting row number of normal command table for use when the value of this BY field or calculation changes.

Column 5 - 1 word - binary integer; number of rows of normal command table to process when the value of this BY field or calculation changes.

Column 6 - 5 words - current value of the GFN for this subgroup level. An integer or calculation result is stored in the first word, whereas a text field may be all 20 characters.



## COMMAND OPERATIONS TABLE

This table does not exist as an identifiable entity in the software. It is an explanation of what is meant by a row of a command table and a definition of what operation is performed for each operator by the subroutine EXCMD. In the description below, columns 1-4 of a command table are referenced by the terms OPND(1), OPERATOR, OPND(2), and RESULT, respectively, and CFLAG is a logical argument in the call to EXCMD.

| <u>OPERATOR</u> | <u>OPERATION PERFORMED</u>   |
|-----------------|--|
| 1               | ADDITION: OPND(1) + OPND(2)→RESULT   |
| 2               | SUBTRACTION: OPND(1) - OPND(2)→RESULT                                      |
| 3               | MULTIPLICATION: OPND(1) * OPND(2)→RESULT                                   |
| 4               | DIVISION: OPND(1)/OPND(2)→RESULT   |
| 5               | .LT.: IF OPND(1) < OPND(2), THEN .TRUE.→CFLAG,<br>OTHERWISE .FALSE.→CFLAG  |
| 6               | .LE.: IF OPND(1) ≤ OPND(2), THEN .TRUE.→CFLAG,<br>OTHERWISE .FALSE.→CFLAG  |
| 7               | .EQ.: IF OPND(1) = OPND(2), THEN .TRUE.→CFLAG,<br>OTHERWISE .FALSE.→CFLAG  |
| 8               | .NE.: IF OPND(1) ≠ OPND(2), THEN .TRUE.→CFLAG,<br>OTHERWISE .FALSE.→CFLAG  |
| 9               | .GE.: IF OPND(1) ≥ OPND(2), THEN .TRUE.→ CFLAG,<br>OTHERWISE .FALSE.→CFLAG |
| 10              | .GT.: IF OPND(1) > OPND(2), THEN .TRUE.→ CFLAG,<br>OTHERWISE .FALSE.→CFLAG |
| 11              | COUNT: IF OPND(2) IS NOT BLANK, THEN OPND(1)<br>+1 → OPND(1)               |
| 12              | NUM. MIN.: THE NUMERICAL MINIMUM OF OPND(1) AND<br>OPND(2) → OPND(1)       |
| 13              | NUM. MAX.: THE NUMERICAL MAXIMUM OF OPND(1) AND<br>OPND(2) → OPND(1)       |

OPERATOR

OPERATION PERFORMED

- |    |   |
|----|---|
| 14 | ALPHA MIN.: THE ALPHA MINIMUM OF OPND(1) AND<br>OPND(2) → OPND(1) |
| 15 | ALPHA MAX.: THE ALPHA MAXIMUM OF OPND(1) AND<br>OPND(2) → OPND(1) |
| 16 | TRANSFER: OPND(1) → RESULT  |

C-2

## COMMAND TABLE

General layout for the normal command table and the function command table. Each row of this five-column table represents an operation to be performed by the execute command subroutine, EXCMD.

Column 1 - 1 word - binary integer pointing to the first operand. A positive number is the row number of the Working Buffer Format. A negative number means an intermediate storage register, and its absolute value tells which register.

Column 2 - 1 word - positive binary integer representing the operation to be performed. See Command Operations Table.

Column 3 - 1 word - binary integer pointing to the second operand for binary operations. Same type pointer as column 1.

Column 4 - 1 word - binary integer pointing to the location where the result of the operation is to be stored. Same type pointer as column 1.

Column 5 - 1 word - binary integer whose value is the row number of this command table to which a jump is made when the current operation cannot be performed due to absence of data in an operand.



COMMON BLOCK CLTBL

This common block contains the 50 word array named COMMAS,  
which is the Comma Location Table.

COMMON BLOCK SY3COM

| <u>Variable or Array</u> | <u>Usage</u>                   |
|--------------------------|--------------------------------|
| CTBL(5,50)               | Normal Command Table           |
| FCTBL(5,10)              | Function Command Table         |
| WBUF(50)                 | Working Buffer                 |
| WBF(7,50)                | Working Buffer Format          |
| SBF(7,20)                | Source Buffer Format           |
| TBF(7,30)                | Target Buffer Format           |
| MLMT(2,20)               | Multilevel Move Table          |
| MTCT(3,2)                | Move Table Control Table       |
| BPT(10,5)                | BY Processing Table            |
| REG(16)                  | Intermediate storage registers |
| CMD(100)                 | Packed input command           |
| NCTBL                    | last used row number in CTBL   |
| NFCTBL                   | last used row number in FCTBL  |
| NWBF                     | last used row number in WBF    |
| NSBF                     | last used row number in SBF    |
| NTBF                     | last used row number in TBF    |
| NMLMT                    | last used row number in MLMT   |
| NMTCT                    | last used row number in MTCT   |
| NBPT                     | last used row number in BPT    |

## COMMA LOCATION TABLE

Table that points to the commas which surround relational, replacement, and BY clauses, and report expressions in the packed input string for the command. It is a one-dimensional array of four-byte integers.

Word 1 - binary integer whose absolute value is the last used word of the Comma Location Table. A negative value is used for the JP & RP commands to indicate that the command terminating exclamation point was encountered by subroutine SQZE.

Words 2-n - binary integers whose values are the character numbers in array CMD of /SY3COM/ where significant syntactical commas occur.

### MULTILEVEL MOVE TABLE

Each row of this two-column table represents a move of data to be made by TFORMW subroutine.

Column 1 - 1 word - binary integer whose value is the row number of the Source Buffer Format array where information about the field in the Source Buffer is located.

Column 2 - 1 word - binary integer whose value is the row number of the Working Buffer Format array where information about the field in the Working Buffer is located.

### MOVE TABLE CONTROL TABLE

Each row of this three-column table defines which moves in the Multilevel Move Table are to be performed for the record in the Source Buffer from a particular data base level.

Column 1 - 1 word - binary integer whose value is the starting row number in the Multilevel Move Table.

Column 2 - 1 word - binary integer whose value is the number of rows in the Multilevel Move Table to be processed via TFORMW subroutine to get all the needed data transferred from the Source Buffer to the Working Buffer at a particular data base level.

Column 3 - 1 word - binary integer whose value is the format number for records at this data base level.