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.
Flight Design System-1
System Design Document

Executive Logic Flow - Program
Design Language

(NASA-TM-80841) FLIGHT DESIGN SYSTEM-1
SYSTEM DESIGN DOCUMENT. VOLUME 9:
EXECUTIVE LOGIC FLOW, PROGRAM DESIGN
LANGUAGE (NASA) 449 p HC A19/MF A01

Mission Planning and Analysis Division
December 1979

NASA
National Aeronautics and
Space Administration
Lyndon B. Johnson Space Center
Houston, Texas
SHUTTLE PROGRAM

FLIGHT DESIGN SYSTEM-1
SYSTEM DESIGN DOCUMENT

EXECUTIVE LOGIC FLOW - PROGRAM DESIGN LANGUAGE

By Mission Analysis and Engineering
Federal Systems Division - Houston
IBM Corporation

JSC Task Monitor: Software Development Branch

Approved:
Elric N. McHenry, Chief
Software Development Branch

Approved:
Ronald L. Berry, Chief
Mission Planning and Analysis Division

Mission Planning and Analysis Division
National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas
December 1979
The Flight Design System-1 (FDS-1) is a pilot project to evaluate current concepts and to determine the hardware/software capability that will be required for the operational era to support Shuttle flight planning. This software system is being implemented on a Hewlett-Packard 21MX computer with a Daconics documentation system and will provide terminal-based interactive flight planning capability.

The System Design Document (SDD) for FDS-1 is the specification for and description of this hardware/software facility. The SDD is logically organized into 10 published volumes. This organization is presented in the accompanying table. The material in the early volumes is primarily presented from the user's point of view, whereas the latter material is software-developer oriented. The SDD will be published by volumes over a period of time, and various volumes will be updated and republished during the development of FDS-1.
FDS-1 SYSTEM DESIGN DOCUMENT

Volume I  Introduction, Overview, and User Interface
Volume II  Utility Processor Library
Volume III Processor Library
Volume IV  System Architecture and Executive
Volume V  Data Management and Data Base Documentation Support System
Volume VI  Standards
Volume VII Utility Support Software
Volume VIII Build and Delivery Procedures, Software Development, Debug, and System Build Aids
Volume IX  Executive Logic Flow - Program Design Language
Volume X  Document Change Request Procedure and Submittal Form

*Combined as one volume with title: Volume III FDS-1 Processor Library
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2.0</td>
<td>COMMON FOR FDS EXECUTIVE</td>
<td>2-1</td>
</tr>
<tr>
<td>3.0</td>
<td>FDS EXECUTIVE MESSAGES</td>
<td>3-1</td>
</tr>
<tr>
<td>4.0</td>
<td>PDL LISTING PROGRAM</td>
<td>4-1</td>
</tr>
<tr>
<td>5.0</td>
<td>FDS EXECUTIVE DETAILED LOGIC FLOW</td>
<td>5-1</td>
</tr>
<tr>
<td>6.0</td>
<td>DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION</td>
<td>6-1</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

The flight design system can be divided functionally into two major areas: the FDS Executive and the application processor. The requirements for and the architecture of the FDS Executive is presented in volume I and volume IV, respectively, of this document. Volume IX presents the detailed logic flow for the FDS Executive.
2.0 COMMON FOR FDS EXECUTIVE

Three commons (XE, XB, and XS) presented in this section are used throughout the FDS Executive.
```
COMMON XE(400), XB(1400), XS(200)

1 10 EXECUTIVE FIXED COMMON (GLOBAL)
2 11 EXECUTIVE DYNAMIC BLOCK (SUBSTA LEVEL DEPENDENT)
3 12 EXECUTIVE SCRATCH SPACE (VOLATILE ACROSS ALL CALLS
4 13 TO FOS ROUTINES EXCEPT XR,...)
5
6

---

7
8
9

---

10 10 XE CONTENTS
11
12
13 10 INTEGER
14 11 # CARTPG
15 12 # CLASMD
16 13 # COMPR
17 14 # FLAGS
18 15 # PRCMNAM(3)
19 16 # REFBUF(64)
20 17 # SEREND
21 18 # SEPR
22 19 # SUBSTA
23 20 # TKMLNG
24 21 # TOCS(32)
25 22 # XE
26 23
27 20 DIMENSION
28 21 # INTNA(3)
29 22
30 20 EQUIVALENCE
31 21 # XE(1) LU
32 22 # XE(3) SUBSTA
33 23 # XE(5) MASSTA
34 24 # XE(7) SEGNAM(1)
35 25 # XE(11) SEREND
36 26 # XE(13) INTNA(1)
37 27 # XE(19) SEPR
38 28 # XE(80) TKMLNG
39 29 # XE(85) TOCS(1)
40 30 # XE(140) XE
41 31 # XE(142) CARTG
42 32 # XE(144) COMPR
43 33
44 40 CARTG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES
45 41 CLASMD - EXECUTIVE/MANAGER REQUEST BLOCK CLASS I/O NUMBER
46 42 COMBGF - TERMINAL COMMUNICATIONS OUTPUT BUFFER
47 43 (1) - NUMBER OF TOKENS IN BUFFER
48 44 (2) - NUMBER OF USED WORDS IN BUFFER
49 45 (5-256) - TOKENS REPRESENTING USER'S RESPONSE
50 46 COMPR - POINTER TO TOKEN CURRENTLY BEING PROCESSED FROM COMBSF
51 47 EXEEND - SEQUENCE # WHERE EXECUTION IS TO END (RETURN TO X)
52 48 FLASE - EXECUTIVE FLAG WORD (0-OFF, 1-ON)
53 49 BITS 0-10 UNUSED
54 50 PROCESSOR ON-LINE DEBUG
55 51 MANAGER ON-LINE DEBUG
56 52 EXECUTIVE ON-LINE DEBUG
57 53 PRODUCE A DUMP ON ALL TERMINATIONS
58 54 MANAGER REQUEST TRANSACTION TRACE FLAG
59 55 INTNA - NAME OF INTERFACE TABLE INPUT TO INTERFACE TABLE EDITOR
```
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR ASSOCIATED WITH PROCESSOR EXECUTED IN MANUAL, SEMI OR AUTO-WITH-TRACE MODE (FIRST WORD = 0 IF DEFAULT INTERFACE TABLE)</td>
<td>LUI - LOGICAL UNIT NUMBER OF TERMINAL BEING SUPPORTED BY THIS EXECUTIVE</td>
<td>HASSA - EXECUTIVE MASTER STATE FLAG (LEVEL LAST PASSED CONTROL BY XECE). RESET TO ZERO BY LEVELS RETURNING TO DIRECTIVE LEVEL.</td>
</tr>
<tr>
<td>BITS 0-9 - NOT USED</td>
<td>BITS 10-13 - DIRECTIVE CONTROL MODE IF BITS 14-15 = 0</td>
<td>BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR</td>
</tr>
<tr>
<td>0 - LIST</td>
<td>0 - INITIALIZATION FROM DIRECTIVE</td>
<td>IF BITS 14-15 = 1</td>
</tr>
<tr>
<td>1 - TORC</td>
<td>1 - INITIALIZATION FOR REENTRY</td>
<td></td>
</tr>
<tr>
<td>2 - SAVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 - RECALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 - DELETE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 - RENAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 - COPY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 - CLEAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 - OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 - STORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 - STOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 - UNLOAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 - LOAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 - BATCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT 14-15 - EXECUTIVE STATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 - DIRECTIVE LEVEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - EXECUTION CONTROL LEVEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - SEQUENCE TABLE EDIT LEVEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 - INTERFACE TABLE EDIT LEVEL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPROC - NUMBER OF PROCESSORS IN LIBRARY</td>
<td>OOLDIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE</td>
<td>PCHNAM - NAME OF PROCESSOR FOR WHICH INTERFACE TABLE EDITOR WAS INVOKED OR BEING EXECUTED IN MANUAL, SEMI OR AUTO-WITH-TRACE MODE</td>
</tr>
<tr>
<td>QUAL - USE: UNIQUE FILE NAME QUALIFIER (SIXTH CHARACTER OF NAME)</td>
<td>REBUF - BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SDR 6.2.7)</td>
<td>REBUF - POINTER TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUF (0 INDICATES REBUF EMPTY) OR RETURN CODE FOR XERR</td>
</tr>
<tr>
<td>RED - READ - NAME OF PROCESSOR FOR WHICH INTERFACE TABLE EDITOR WAS INVOKED OR BEING EXECUTED IN MANUAL, SEMI OR AUTO-WITH-TRACE MODE</td>
<td>SECUENT- TERMINATING SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN SEMI OR AUTO MODE AS PASSED TO THE MANAGER</td>
<td>SECUENT - NAME OF SEQUENCE TABLE INPUT TO SEQUENCE TABLE EDITOR OR EXECUTED IN SEMI OR AUTO MODE</td>
</tr>
<tr>
<td>SYP - SYMP - POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SEMI</td>
<td>SYMP - INITIAL SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN SEMI OR AUTO MODE</td>
<td>SUBS - EXECUTIVE SUB-STATE FLAG (LEVEL IN COMMUNICATION WITH USER TERMINAL). SET TO LEVEL TO BE INITIALIZED NEXT OR ZERO IF LEVEL INITIALIZATION FAILS.</td>
</tr>
</tbody>
</table>
0 - DIRECTIVE LEVEL
1 - EXECUTION CONTROL LEVEL
2 - SEQUENCE TABLE EDIT LEVEL
3 - INTERFACE TABLE EDIT LEVEL

TABEND - SEQUENCE # OF LAST ENTRY IN SEQUENCE TABLE
TOKENS - NUMBER OF CONSECUTIVE ENTRIES PRESENTLY ALLOCATED IN

TOKENS (29)

TOKENS - IDENTIFYING NUMBERS OF TOKENS RETURNABLE IN COMBUF
(1)= 0= END OF BUFFER
(2)= 1= INTEGER NUMBER
(3)= 2= REAL NUMBER
(4)= 3= DOUBLE PRECISION NUMBER
(5)= 4= SIX CHARACTER NAME
(6)= 5= CHARACTER STRING
(7)= 6= +
(8)= 7= -
(9)= 8= *
(10)= 9= /
(11)= 10= <
(12)= 11= >
(13)= 12= =
(14)= 13= &
(15)= 14= 
(16)= 15= ;
(17)= 16= (;
(18)= 17= )
(19)= 18= )
(20)= 19= BEGIN SYMBOLIC STRING
(21)= 20= END SYMBOLIC STRING
(22)= 21= =
(23)= 22= /
(24)= 23= =
(25)= 24= =
(26)= 25= \ LEFT BRACKET
(27)= 26= \ RIGHT BRACKET
(28)= 27= \ REPEAT
(29)= 28= \ END
(30)= 29= \ UNDF
(31)= 30= \ INCLUDE
(32)= 31= \ INCLUDE
147  1 CO  XB CONTENTS (DIRECTIVE LEVEL)
148  1 CO  INTEGER
149  1 CO  = DIRECT(50)
150  1 CO  EQUIVALENCE
151  1 CO  = (XB(1), NUBDIR)  , (XB(2), DIRECT(1))
152  1 CO  = (XB(151), BEGINNING OF DIRECTIVE DEPENDENT ALLOCATION)
153  1 CO  DIRECT - FIXED ORDER LIST OF FDS DIRECTIVES (FOUR CHARACTERS PER
154  1 CO  DIRECTIVE)
155  1 CO  NUBDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
156  1 CO  ******************
162      CD      XB CONTENTS (EXECUTION CONTROL LEVEL)
163      CD      INTEGER
164      CD      ASCLNT
165      CD      RESIND
166      CD      SERIND
167      CD      SERNO
168      CD      SERLEN
169      CD      SERTAB
170      CD      DIMENSION
171      CD      LIBB(150)
172      CD      ASCLNT(10)
173      CD      SERTAB(1150)
174      CD      EQUIVALENCE
175      CD      (XB(1),NP2C2)
176      CD      (XB(235),RESIND)
177      CD      (XB(249),SERNO)
178      CD      (XB(250),SERLEN)
179      CD      (XB(251),SERTAB)
180      CD      ASCLNT - SEQUENCE TABLE ENTRY IN ASCII TO PROMPT USER
181      CD      IN SEMI MODE ONLY
182      CD      CURIND - CURRENT INDEX TO EXECUTING SEQUENCE ENTRY
183      CD      LIBB - LIBRARY DIRECTORY PROCESSOR NAME TABLE
184      CD      NPROC2 - NUMBER OF PROCESSORS IN LIBB (SAME AS XE(143))
185      CD      RESIND - INDEX OF RESET ENTRY WHEN RESET SEQUENCE 0 IS REQUESTED
186      CD      SERNO - NUMBER OF ENTRIES IN SEQUENCE TABLE
187      CD      SERLEN - LENGTH OF SEQUENCE TABLE
188      CD      SERTAB - SEQUENCE TABLE CURRENTLY BEING EXECUTED
189      CD     
JSUB F - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
LENFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
LISTLU - LU TO WHICH PRINT SHOULD GO (USED WHEN
LST DIRECTIVE CALLS XILSD OR XICHRT)
LITDSP - DISPL. TO LITERAL DATA FOR THIS ARGUMENT
LITDNL - INDEX IN WKBUF TO END OF LITERAL DATA
LITLEN - LENGTH OF LITERAL DATA AREA OF WKBUF
LITPTR - INDEX IN WKBUF TO START OF LITERAL DATA
LISTZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
LISTFLG - FLAG USED TO DETERMINE ORIGIN OF A
CALL TO XILSD OR XICHRT.

= 0, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
AN INTERFACE TABLE
= 3, CALLED FROM INTERFACE TABLE EDITOR TO LIST ARGUMENT
DATA, PARAMETER OR INCOMPLETE INDICATORS
= 4, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
A DATA ELEMENT'S VALUE(S)

MODAV - PREVIOUS VALUE OF PRMTMD WHILE PRMTMD = 4 (CONTINUE)
MARG - INDEX IN WKBUF TO START OF SHORT PROMPTS
MDSSM - INDEX IN WKBUF TO BIT MASK WORD(S) FOR THIS ARGUMENT
WETATB - ASCII NAME OF TABLE BEING GENERATED
NOBSTM - NO. OF BIT MASTS ASSOCIATED WITH THIS ARGUMENT
NMA R-NO. OF ARGUMENTS IN THIS INTE. TABLE
NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
PRMLN - LENGTH (IN WORDS) OF PROMPT BUILT
PRMTMD - CURRENT PROMPTING MODE

PROMPT - ASCII ARRAY FOR PROMPT BUILT
SFLAG - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
VERSION - VERSION NO. OF THIS INTERFACE TABLE
WKBLNG - LENGTH OF WKBUF
WKBUFF - WORKING BUFFER FOR INTERFACE TABLE BEING EDITED
ORGANIZED AS :

HEADER (7 WORDS)
ARGUMENT CHARACTERISTICS (NUMARG*7 WORDS)
ARGUMENT PROMPTS (NUMARG*3 WORDS)
LITERAL DATA AND PARAMETER SUBSCRIPTS (LITLEN WORDS)
3.0 FDS EXECUTIVE MESSAGES

The list of messages generated by the Executive are presented in this section.
FD$ ERROR MESSAGES

* XA01 ATTENTION FUNCTION MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION
* XA02 USER INITIATED INTERRUPT ENTER REQUEST - KILL(S), STATUS(S), RETURN( )
* XA03 ERROR, LU IS NOT SIGNED ON TO FD$ FDS ATTN FUNCTION TERMINATING
* XA05 FDS MANAGER SIGNALED TO TERMINATE BOTTOM ASSOC TASK
* XA06 FDS EXECUTIVE ACTIVE - NO ACTION TAKEN
* XA07 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT
* XA08 MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN
* XB BATH JOB CREATION
* XC CONFIGURATION PROGRAMS
  * XD1 'NN' SIGNED ON TO FD$ D103 'ID' SIGNED ON REJECTED
  * XD1 'LU 'LU' IS AN INVALID LU D104 CANNOT SIGN ON
  * XD2 FD$ CURRENTLY AT MAX USER'S. CANNOT SIGN ON
  * XD4 'ID' SIGN ON TERMINATED
* XD6 'LU' IS ALREADY SIGNED ON TO FD$ CANNOT SIGN ON
* XD7 CANNOT FIND 'NAME' ID- SIGN ON TERMINATED
* XD8 ENTER VALID USER ID (A-Z)

* XM MANAGER
  * XM01 INVALID REQUEST II FROM 'NAME'
  * XM02 'LU' IS CURRENTLY SIGN ON
  * XM03 'LU' IS CURRENTLY USING ID 'ID' D104 'NAME'
* XM05 'ID' SIGN ON REJECTED
  * XM06 'ID' SIGNED ON TO DWA
  * XM07 I/O ERROR FOR DWA, STATUS = 'NNNNNN'
  * XM08 TOO MANY NESTED REPEATS
  * XM09 TOO MANY NESTED SUBREPEATS
  * XM10 INO responders TO REQUEST
  * XM11 AWA ACCESS FAILURE FOR ...
  * XM12 PROCESSOR INITIALIZATION ERROR

* XP PROCESSOR SERVICES
  * XP01 TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED
  * XP02 DATA AREA OVERFLOW
  * XP03 SUBREPEAT OUT OF RANGE
  * XP04 TOO MANY NESTED SUBREPEATS
  * XP05 INO RESPONSE TO REQUEST
  * XP06 INO RESPONSE TO REQUEST II
  * XP07 INVALID SUBREPEAT
  * XP08 RESPONSE IS TOO LONG FOR BUFFER
  * XP09 'NAME' SIGNED ON TO DWA
  * XP10 'NAME' SIGNED ON TO DWA

* XV SYSTEM SERVICES
  * XV01 'ID-N0M' NAMED PROGRAM MADE A PAN REQUEST RESERVED FOR MANAGER
  * XV02 'ID-N0M' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER REPLY
  * XV03 'ID-N0M' NAMLO PROGRAM HAS INVALID BACK CHAIN TO MANAGER
4.0 PDL LISTING PROGRAM

The detailed logic flow of the program that generates PDL listing is presented as follows.
PDL STRUCTURED LISTING PROGRAM

**INPUT**
80 COLUMN PDL IMAGES SUBJECT TO THE FOLLOWING CONVENTIONS:
* IN COLUMN 1 INDICATES PAGE EJECT AND THE FIRST TOKEN (6 OR LESS
CHARACTERS) IS REPRODUCED IN COLUMNS 127-132 OF OUTPUT LISTING
UNTIL NEXT RECORD IS DETECTED. IF RECORD CONTAINS ONLY THE
TOKEN FROM THE PREVIOUS RECORD CONTINUES TO APPEAR IN THE
IDENTIFICATION COLUMNS OF THE OUTPUT
* IN COLUMN 1 INDICATES A COMMENT TO BE COPIED TO THE OUTPUT LISTING.
* OTHER IN COLUMN 1 INDICATES PDL RECORD TO BE STRUCTURED BASED ON KEY
WORDS APPEARING AS FIRST NON-BLANK CHARACTERS (SEE LOGIC).

**OUTPUT**
132 COLUMN LISTING AS FOLLOWS
2-6 SEQUENCE NUMBER
8-11 STRUCTURE LEVEL NUMBER
13-125 STRUCTURED LISTING OF 80 COLUMN INPUT RECORDS
127-132 IDENTIFICATION
FOLLOWING THE LISTING IS A SYMBOL DEFINITION TABLE INDICATING THE
SEQUENCE NUMBER OF THE LINE CONTAINING EACH 'BEGIN NAME' AND
"LABEL:":

**NOTES**
USES FSTWRD & SORT1
33 1 BEGIN PDLIST
34 2 INITIALIZE SEQUENCE NUMBER, LEVEL AND DEFINITION TABLE COUNTER
35 3 DO UNTIL END-OF-FILE INPUT
36 4 READ RECORD
37 5 INCREMENT SEQUENCE NUMBER
38 6 IF COLUMN 1 = .
39 7 THEN
40 8 SET PAGE EJECT IN IMAGE
41 9 IF REMAINDER OF IMAGE IS NOT BLANK
42 10 THEN
43 11 SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
44 12 ENDIF
45 13 ELSE
46 14 CLEAR LEVEL INCREMENT
47 15 IF COLUMN 1 NOT= *
48 16 THEN
49 17 IF FIRST CHARACTER = :  — INDICATES :LABEL:
50 18 THEN
51 19 SET LINE SKIP IN IMAGE
52 20 GENERATE DEFINITION TABLE ENTRY FOR LABEL
53 21 ELSE
54 22 CALL FSTD to GET FIRST WORD OF PDL
55 23 LOOKUP FIRST WORD IN KEY WORD TABLE
56 24 KEY WORD TABLE CONTAINS
57 25 1 BEGIN — BEGIN SECTION INDICATOR
58 26 2 IF — SECTION INDICATOR
59 27 3 DO — SECTION INDICATOR
60 28 4 DFOR — SECTION INDICATOR
61 29 5 DOUNTI — SECTION INDICATOR
62 30 6 DOWHIL — SECTION INDICATOR
63 31 7 CASE — SECTION INDICATOR
64 32 8 START — SECTION INDICATOR
65 33 9 STARTS — SECTION INDICATOR
66 34 10 ELSE — SECTION SEPARATOR
67 35 11 THEN — SECTION SEPARATOR
68 36 12 EXIT — SECTION SEPARATOR
69 37 13 EXITF — SECTION SEPARATOR
70 38 14 OR — SECTION SEPARATOR
71 39 15 DELSE — SECTION SEPARATOR
72 40 16 ENDOLO — SECTION SEPARATOR
73 41 17 END — END OR END LOOP?
74 42 18 ENDF — SECTION TERMINATOR
75 43 19 ENDDO — SECTION TERMINATOR
76 44 20 ENDCAS — SECTION TERMINATOR
77 45 21 ENDOSEA — SECTION TERMINATOR
78 46 IF KEY WORD LOCATED
79 47 THEN
82 50 (:SEP:, :TERM:, :TERM:, :TERM:, :TERM:)
83 51 :BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
84 52 SET LEVEL INCREMENT = 1
85 53 :SECOND: SET LEVEL INCREMENT = 1
116 1 CD1       EXTRACT THE FIRST TOKEN FROM A PDL RECORD
117 1 CD1
118 1 C---------- INPUT
119 1 CD2    SINGLE CHARACTER PER WORD RECORD AND LENGTH
120 1 CD2
121 1 CD2
122 1 C---------- OUTPUT
123 1 CD3    FIRST (NEXT) TOKEN IN SIX CHARACTER WORD. BLANKS AND : ARE
124 1 CD3    DELIMITERS. THE DISPLACEMENT OF THE NEXT CHARACTER IN THE RECORD IS
125 1 CD3    ALSO OUTPUT
126 1 CD3
127 1 CD3
128 1 C----------
129 1 *
130 1 *
131 1 *
132 1 BEGIN FSTWRD
133 2  BLANK OUTPUT WORD
134 2  LOCATE FIRST NON-BLANK CHARACTER
135 2  DO UNTIL SIX CHARACTERS STORED OR END-OF-RECORD
136 3  IF CHARACTER IS NON-BLANK AND NON-:
137 3  THEN
138 4  STORE CHARACTER
139 4  ELSE
140 4  EXIT DO
141 4  EIDOIF
142 4  ENDDO
143 2  RETURN LOCATION
144 2  END FSTWRD
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DEFINITION</th>
<th>TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>46</td>
<td>323</td>
</tr>
<tr>
<td>b</td>
<td>133</td>
<td>88</td>
</tr>
<tr>
<td>c</td>
<td>87</td>
<td>47</td>
</tr>
</tbody>
</table>

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR.
5.0 FDS EXECUTIVE DETAILED LOGIC FLOW

A directory listing the major programs and subroutines in alphabetical order is presented initially. The detailed logic flow of each then follows in alphabetical order.
MULTIPLE WORD BIT STRING BIT CLEAR/SET

COMPARE ARRAYS

DOUBLE PRECISION TO ASCII CONVERSION

ASCII MANAGEMENT REQUEST ROUTINE

EXTRACTS A VARIABLE LENGTH FIELD FROM A WORD

CONVERT A WORD TO ASCII IN 16-BIT FORMAT

CONVERT A BINARY INTEGER TO ASCII CONVERSION ROUTINE

I/O BUS BLOCK RN LOCK

RETURN 16-BIT ADDRESS OF ARGUMENT

MOVES WORDS FROM ARRAY1 TO ARRAY2

FS EXECUTIVE MESSAGE ROUTINE

MULTIPLE WORD BIT STRING BIT SEARCH

SETS A VARIABLE LENGTH FIELD INTO A WORD

SHIFT A WORD LEFT LOGICALLY

SHIFT A WORD RIGHT LOGICALLY (XRSFL EP)

I/O BUS RN UNLOCK (XRLC K EP)

FILE NAME QUALIFICATION

FILE NAME DEQUALIFIED

REMOVES BLANKS AND UNPACKS FROM A2 TO R1 FORMAT

REMOVES DUPLICATE BLANKS FROM A2 STRING

SEQUENCE TABLE EDITOR

SEQUENCE TABLE EDITOR (STE) DIRECTIVE SCANNER

STE DELETE DIRECTIVE PROCESSOR

STE ENTITY PROCESSOR

SEQUENCE TABLE EDITOR MAIN ROUTINE

STE LIST DIRECTIVE PROCESSOR

SEQUENCE TABLE LIST ROUTINE

STE INPUT PROCESSOR

STE NUMBER DIRECTIVE PROCESSOR

STE TABLE COMPACTOR

STE PROMPT DIRECTIVE PROCESSOR

STE PROMPT CONSTRUCTOR

TERMINAL COMMUNICATIONS

PROMPTS USER, READS RESPONSE, CALLS XITLAN AND XTPRM

CONVERTS ASCII USER'S RESPONSE TO TOKENS

HANDLES EXTENDED PROMPTING REQUESTS

UTILITY (SOFTWARE AIDS)

ON-LINE SNAP AND MEMORY MODIFICATION ROUTINE

SYSTEM RESIDENT PARTITION DUMP (XVABN EP)

FILE MANAGER FILE DUMP PROGRAM

OCTAL AND ASCII DUMP LINE FORMAT

DUMP FORMATTER

SYSTEM SERVICES

FDS ABEND (SEE XUDMP)

FDS COMMUNICATIONS SERVICES (POST AND WAIT)

EXECUTION CONTROL

AUTOMATIC MODE

EXECUTION CONTROL MAIN PROGRAM

DECODES USER RESPONSE IN MANU AND SEMI

READS IN DEFAULT INTERFACE TABLE IF NEEDED

EXECUTES ASRTAB AND HANDLES ERROR CONDITIONS

MANUAL MODE
170 1: XXSEIM  SENI - AUTOMATIC NODE
180 1: XXSTO  STORE SEQUENCE TABLE IN ASERTAB
181 1: XXTMP  TEMPORARY EXECUTION OF ONE ENTRY WITH ASERTAB
182 1: **  UTILITY PROCESSORS
183 1: **  ASSGN  ASSIGN PROCESSOR
184 1: **  DBDSP  DATA BOX DISPLAY PROCESSOR
185 1: **  DEFIN  DEFINE PROCESSOR
186 1: **  DO  CONDITIONAL LOOP IN SEQUENCE TABLE
187 1: **  ELSE  EXECUTION POINT FOR FALSE IF CONDITION
188 1: **  ENDEIF  TERMINATES AN IF STRUCTURE
189 1: **  ENDOO  TERMINATES A DO LOOP STRUCTURE
190 1: **  ENDSCE  END SCAN PROCESSOR
191 1: **  IF  CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES
192 1: **  SCAN  SCAN PROCESSOR
193 1: **  XICHR  CHARACTER OBJECT STORE FOR ASSGN
194 1: **  XZFTT  FIND ANY TOKEN IN A SYMBOLOGIC STRING
195 1: **  XZDIN  DATA BOX DISPLAY INPUT PROCESSOR
196 1: **  XZDINK  DATA BOX DISPLAY CONSTRAINT MASKER
197 1: **  XZDOT  DATA BOX DISPLAY OUTPUT ROUTINE
198 1: **  XZDOP  DATA BOX DISPLAY PASS 1 PROCESSOR
199 1: **  XZDOP2  DATA BOX DISPLAY PASS 2 PROCESSOR
200 1: **  XZEVF  PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS
201 1: **  XZFCL  FIND PROCESSOR CLASS NUMBER
202 1: **  XZFRM  FUNCTIONAL OPERATIONS FOR ASSGN
203 1: **  XZFRM  FREE OBJECT STORE FOR ASSGN
204 1: **  XZFRD  FIXED OBJECT STORE FOR ASSGN
205 1: **  XZISIP  REMOVE DUPL. BLANKS & BLANK FILL
206 1: **  XZLSS  SYMBOLOGIC STRING SYMBAK ERROR LISTER
207 1: **  XZPGTG  FRS PROCESSOR MESSAGE ROUTINE
208 1: **  XZFRG  MATH OPERATIONS FOR ASSGN
209 1: **  XZPS  DATA CONVERSION AND STORAGE FOR ASSGN
210 1: **  XZPS1  PASS 1 SUBROUTINE FOR ASSGN PROCESSOR
211 1: **  XZPS2  PASS 2 SUBROUTINE FOR ASSGN PROCESSOR
212 1: **  XZRET  DATA RETRIEVAL FOR ASSGN
213 1: **  XZSCH  SEARCHES SEQUENCE TABLE FOR IF STRUCTURES
214 1: **  XZSYM  SYMBOL TABLE INTERFACE FOR ASSGN
215 1: **  XZT  SYMBOL TABLE MAINTENANCE
SAVE EOT ADDRESS(IN BREG ON ENTRY)
CALL ERLU(BREG) GET LU IN ASCII & BINARY
STARTUP UNTIL LAST STATUS TABLE ENTRY
EXITIF STBG EN LU
SET STBG ENTRY ADDRESS
ENDLOOP
SET STBG ENTRY TO ZERO
ENDSEARCH
IF STBG ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STBG)
IF MANAGER IS DORMANT, THEN
WRITE "***XAO- MANAGER HAS TERMINATED;"
READ(LU) ** WAIT FOR REPLY **
LOCK ON THE FDS TABLE RESOURCE
CALL SLBRM DISABLE
IF STBEXECUTIVE ADDRESS .NE. 0, THEN
IF STAT(CURRENT) .NE. STBEX, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XCHR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STBG(MANGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARN ONE(P1)
CALL SLBRM ENABLE
CALL MESS 'OFF,BOTTOM'
CALL SLBRM DISABLE
SET BOTTOM TO NEXT-AT
ENDO
ENDIF
ENDIF
CALL SLIST MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBG
ENDIF
DECREMENT NUMBER ACTIVE(STBAC)
GET EOT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBEX)
CLEAR STBER
CLEAR MANAGER'S ID, STBG, & STLU
ENABLE... (VIA A JMP TO EXEC(DISPATCHER))
RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS
CLEAR LOCK ON FDS TABLE
ENDIF
ELSE ** MANAGER IS STILL ALIVE **
WRITE "***XAO- USER INITIATED INTERRUPT?"
WRITE 'ENTER REQUEST- KILL(S), STATUS(S), OR RETURN(BLANK)' READ (LU) REQUEST
IF REQUEST IS KILL OR 1, THEN
PERFORM XAINT
ELSE
IF REQUEST IS STATUS OR S, THEN
PERFORM XAST
ENDIF
ENDIF
ELSE
WRITE '***XAO- ERROR LU IS NOT SIGNED-ON TO FDS'
ENDIF
WRITE '***XAO- FDS ATTENTION FUNCTION TERMINATING'
END
77 1 IF: X STAT PRODUCE A FDS STATUS REPORT
78 2 CALL SLIBR - DISABLE
79 2 GET CURRENT-TIME FROM STIME
80 2 GET MANAGER'S ADDRESS FROM STBNG
81 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
82 2 GET EXECUTIVE'S ADDRESS FROM STBEX
83 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
84 2 GET CURRENT AT FROM STBAT
85 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
86 2 PERFORM XABTM(CURRENT) FIND BOTTOM AT
87 2 SET BOTTOM TO CURRENT
88 2 GO WHILE FATHER-ID NE ZERO
89 3 SET FATHER-ID FROM CURRENT
90 3 CALCULATE NEXT
91 3 IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
92 4 MOVE NAME, STATUS, PARTITION, & PRIORITY
93 3 ENDF
94 3 IF NEXT IS THE MANAGER, THEN
95 4 SET CURRENT AS TOP
96 3 ENDF
97 3 SET CURRENT TO NEXT
98 2 ENDDO
99 1 IF TOP IS ZERO, THEN
100 2 USE TOP TO MOVE NAME, PARTITION, & PRIORITY
101 2 USE STATUS TO 'IN USE' OCTAL 17
102 2 ENDF
103 2 CALL SLIBX ENABLE
104 2 WRITE FIRST SET OF HEADERS
105 2 SET TOP AS REPORT DATA
106 2 WRITE REPORT LINE
107 2 SET MANAGER AS REPORT DATA
108 2 WRITE REPORT LINE
109 2 WRITE REPORT LINE
110 2 SET EXECUTIVE AS REPORT DATA
111 2 WRITE REPORT LINE
112 2 SET CURRENT AS REPORT DATA
113 2 WRITE REPORT LINE
114 2 WRITE INTERMEDIATE HEADERS
115 2 DO UNTIL MAX ENTRIES OR NO MORE DATA
116 3 WRITE REPORT LINE
117 3 SET NEXT REPORT DATA
118 2 ENDDO
119 1 END XSTAT
120 1 SAMPLE REPORT
121 2 FDS STATUS FOR LU 10 HH:MN:SS 360
122 2 # NAME PRIOR PART# STATUS
123 2 # TOP AT- PROC 926 3 GENERAL WAIT
124 1 # MANAGER- XGMNN 40 5 GENERAL WAIT
125 1 # EXECUTIVE- XEXNN 60 3 GENERAL WAIT
126 1 # CURRENT AT- PROC 1131 1 4 GENERAL WAIT
127 1 # BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
128 1 # PROC 32767 6 DISC ALLOCATE SUSPEND
129 1 PROC 1131 4 GENERAL WAIT
130 1 PROC 2040 3 GENERAL WAIT
131 1 PROC 845 6 GENERAL WAIT
132 1
140 1 BEGIN XAKILL
141 2 * TERMINATE CURRENT FDS FUNCTION
142 3 * IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
143 4 * ON NEXT RETURN VIA A PAN,
144 5 * IF THE EXEC IS ACTIVE-DO NOTHING
145 6 * IF A PROCESSOR IS ACTIVE- USE RTE MESS TO OFF THE PROCESSOR
146 7 * SET MANAGER'S ID ADDRESS(STBMG)
147 8 * IF STATUS OF MANAGER IS NOT WAIT, THEN
148 9 * SET TERMINATE FLAG IN STD-ENTRY
149 10 * WRITE "****XA05 FDS MANAGER SIGNALED TO TERMINATE SEQUENCE"
150 11 ELSE
151 12 IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
152 13 WRITE "****XA06 FDS EXECUTIVE ACTIVE; NO ACTION TAKEN"
153 14 ELSE
154 15 PERFORM XATHM(CURRENT) FIND BOTTOM AT
155 16 IF BOTTOM AT IS D.RTR OR SNP THEN
156 17 WRITE "****XA08 MANAGER IS WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
157 18 EXIT XAKIL
158 19 ELSE
160 20 IF RETURNED BOTTOM IS MANAGER THEN
161 21 IF MANAGER IS NOT WAITING ON A PROGRAM THEN
162 22 WRITE "****XA09 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
163 23 EXIT XAKIL
164 24 ELSE
165 25 SET RETURN PARAMETER TO PROCESSOR ABENDED
166 26 INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
167 27 CALL BLIST TO REACTIVATE MANAGER
168 28 ENDF
169 29 ELSE
170 30 WRITE "****XA07 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT."
171 31 IF RETURNED BOTTOM WAS NOT MANAGER THEN
172 32 SET NAME IN 'OFF' COMMAND
173 33 CALL MESS TO 'OFF' THE PROCESSOR
174 34 ENDF
175 35 ENDF
176 36 ENDF
177 1 END XAKILL
179 1 BEGIN XABTM  FIND BOTTOM AT
180 2  DO WHILE CURRENT IS IN GENERAL WAIT,
181 3 AND WAIT POINTER(P) HAS A SON ADDRESS,
182 3 AND SONS FATHER ID POINTS TO CURRENT
183 3 SET SON AS CURRENT
184 2 ENDDO
185 2 SET BOTTOM AS CURRENT
186 1 END XABTM
29 2 * PARMS=LU,IO,DWA SIZE,PARM(ON OR OFF),OPTIONS
30 2 * SWITCH INPUT PARNs AROUND SO THAT
31 2 * NOW PARM=LU,P2(ON OR OFF),ID,OPTS,DWA SIZE
32 2 * FOR COMPATABILITY TO BUILD 1.
33 2 CALL RMPAR(PARMS)
34 2 IF LU IS .LT. 0, OR
35 3 .GT. LUNAY(1653), OR
36 3 .EQ. 6(PRINTER), OR
37 3 THE DRIVER IS .NE. 0 OR 5, THEN
38 3 ISSUE MESSAGE "***XCO4 'LU' IS AN INVALID LU"
39 ELSE
40 IF PARM P2 IS OFF THEN
41 4 PERFORM XCOFF SIGN OFF
42 ELSE
43 4 PERFORM XCON SIGN ON
44 ENDIF
45 2 ENDIF
46 2 :EXEC
47 2 CALL EXEC PROGRAM TERMINATION
48 1 END XCONF
1 BEGIN XCON
2   * SIGN ON A USER TO FDS
3     LO UNTIL VALID USER ID (P3)
4     IF ID NOT A - 2, THEN
5     WRITE 'XCOB ENTER VALID ID (A - Z)'  
6     READ RESPONSE
7     ENDF
8     ENDDO
9     IF USER ID IS BEING USED, THEN
10    WRITE '***XCOB LU 'LU' IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED'
11     EXIT :XCETA
12     ENDF
13     IF FDS RESOURCE NUMBER NOT DEFINED, THEN
14    CALL RMAG (GLOBAL ALLOCATE, LOCAL SET)
15     ELSE
16     CALL RMAG (LOCAL SET)
17     ENDF
18     IF NUMBER SIGNED ON (STBNK,) .EQ. MAXIMUM USERS(STBNK), THEN
19     ISSUE MESSAGE '***XCOB FDS CURRENTLY AT MAX USERS.'
20     ELSE
21     DO FOR STBNK(NUMBER OF FDS ENTRIES)
22        IF ENTRY'S LUCSTBLU (LU) REQUESTING LU(P1) THEN
23           ISSUE MESSAGE '***XCOB 'LU' IS CURRENTLY SIGNED ON TO FDS'
24           EXIT :XCETA
25         ELSE
26           IF THIS ENTRY IS AVAILABLE, THEN
27              SET CURRENT-ENTRY-ADDRESS
28              ENDF
29         ENDIF
30     ENDDO
31     ENDDO
32     IF XMGR NOT FOUND AND THIS ID .EQ. XMGR, THEN
33        SET ID ADDRESS OF XMGR
34        INCREMENT NUMBER-FOUND
35     ELSE
36        IF XATM NOT FOUND AND THIS ID .EQ. XATM, THEN
37           SET ID ADDRESS OF XATM
38           INCREMENT NUMBER-FOUND
39        ELSE
40           IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
41              SET ID ADDRESS OF FIRST-BLANK
42              INCREMENT NUMBER-FOUND
43           ELSE
44              IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
45                  SET ID ADDRESS OF SECOND-BLANK
46                  INCREMENT COUNT
47              ENDIF
48        ENDIF
49        ENDF
50     ENDIF
51     ENDF
52     ENDF
53     ENDF
54     ENDF
55     ENDIF
56     ENDF
57     ENDF
58     ENDF
59     ENDF
60     ENDF
61     ENDF
62     ENDF
63     ENDF
64     ENDF
65     ENDF
66     ENDF
67     ENDF
68     ENDF
69     ENDF
70     ENDF
71     ENDF
72     ENDF
73     ENDF
74     ENDF
75     ENDF
76     ENDF
77     ENDF
78     ENDF
79     ENDF
80     ENDF
81     ENDF
82     ENDF
83     ENDF
84     ENDF
85     ENDF
86     ENDF
87     ENDF
88     ENDF
89     ENDF
90     ENDF
91     ENDF
92     ENDF
93     ENDF
94     ENDF
95     ENDF
96     ENDF
97     ENDF
98     ENDF
99     ENDF
100    ENDF
101    ENDF
102    ENDF
103    ENDF
104    ENDF
105    ENDF
106    ENDF
107    ENDF
108    EXIT IF THERE ARE NO MORE IDS
CALL BLINK ENABLE
ISSUE MESSAGES "**XCO/ CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP
BUILD ENTRY IN XINST

SET LU INQ STBLU
SET LU IN ASCII INTO STBLA
SET USER'S ID INTO STIB
SET ADDRESS OF FIRST_BLANK INTO STBLR
SET ADDRESS OF SECOND_BLANK INTO STBER
INCREMENT ACTIVE COUNT(STBAC)
BUILD XENVN & XENVN
MOVE PRIORITY THRU DISC ADDRESS FROM XMGR TO FIRST_BLANK
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND_BLANK
TURN ON TH BIT
SET NAME TO XENVN
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND_BLANK
TURN ON TH BIT
SET NAME TO XENVN
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE ERT OVERLAY
SAVE ERT VALUE IN STBER
SET ID ADDRESS OF EATTN INTO ERT
ENDIF
SET INPUT PARRS INTO ID OF XMGN
SCHEDULE XENVN VIA BLIST
CALL BLIST
CALL BLINK ENABLE
IF FDS HAS A FATHER, THEN
CALL MESS 'OFF,FATHER'
ENDIF
ISSUE MESSAGE "**XCO1 LU "MM" SIGNED ON TO FDS"
148 1 BEG: XCOFF
149 2 * SIGN OFF A USER / FDS
150 3 " WRITE 'FDS SIGN OFF FUNCTION NOT SUPPORTED'
151 4 END XCOFF
152
FORTRAN CALLING PROCEDURE
CALL XDLD (XDCLD)

XDCLD DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
LIST

INPUT
XE COMMON - MASSTA (BITS 10-13 CONTAIN A 0 INDEX INTO A LIST OF
DIRECTIVES)

INTERNAL VARIABLES
LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

NOTES
USES .ENTRY, XDLIST, XENTRY

XDCLD IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
CONTAINING THE REFERENCED DIRECTIVES

BEGIN XDCLD
EXTRACT DIRECTIVE INDEX FROM MASSTA
CASE (LIST:) INDEX
LIST: CALL XDLIST
ENDCASE
CALL XENTRY TO RETURN FROM SEGMENT
END XDCLD
40 1 #00  FORTRAN CALLING PROCEDURE
41 1 #00  CALL XELBS (XCLF)
43 1 #00  *********
44 1 #01  XCLF DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
45 1 #01  REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
47 1 #01  TO:
48 1 #01  SAVE
49 1 #01  RECALL
50 1 #01  DELETE
51 1 #01  RENAME
52 1 #01  COPY
53 1 #01  CLEAR
54 1 #01  OFF
55 1 #01  *********
57 1 #02  INPUT
58 1 #02  XE COMMON - MASSTA (BIT 10-13 CONTAIN A 1-8 INDEX INTO A LIST OF
59 1 #02  DIRECTIVES)
60 1 #02  *********
61 1 #04  INTERNAL VARIABLES
63 1 #04  LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
64 1 #04  *********
66 1 #05  NOTES
67 1 #05  USES X fools, XDEC, XDCOP, XDELE, XDOFF, XDREC, XDREM, XDSEY,
68 1 #05  XTOC, XETTM
69 1 #05  XCLF IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
71 1 #05  CONTAINING THE REFERENCED DIRECTIVES
72 1 #05  *********
74 1 1 *
75 1 1 *
76 1 1 *
77 1 BEGIN XCLF
78 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
80 3 :TOC: CALL XTOC
81 3 :SAVE: CALL XDSEY
82 3 :RECA: CALL XDREY
83 3 :DELE: CALL XDELE
84 3 :REMA: CALL XDREM
85 3 :COPY: CALL XDCOP
86 3 :CLEA: CALL XCLF
87 3 :OFF: CALL XDSEY
88 2 ENDCASE
89 2 CALL XETTM TO RETURN FROM SEGMENT
90 1 END XCLF
92 1 DO  
93 1 DO  
94 1 DO  
95 1 ***  
96 1 DO  
97 1 DO  
98 1 XDCLU DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS  
99 1 DO  
100 1 DO  
101 1 DO  
102 1 DO  
103 1 DO  
104 1 DO  
105 1 ***  
106 1 DO  
107 1 DO  
108 1 ***  
109 1 DO  
110 1 ***  
111 1 DO  
112 1 ***  
113 1 DO  
114 1 ***  
115 1 DO  
116 1 DO  
117 1 DO  
118 1 DO  
119 1 DO  
120 1 DO  
121 1 ***  
122 1 ***  
123 1 ***  
124 1 ***  
125 1 ***  
126 1 BEGIN XDCLU  
127 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7  
128 2 CASE (:STOR :,REST :,UML0 :,LOAD :,BATCH:) INDEX  
129 3 :STOR: CALL XDSTO  
130 3 :REST: CALL XDRES  
131 3 :UML0: CALL XDUNL  
132 3 :LOAD: CALL XDLOAD  
133 3 :BATCH: CALL XD BAT  
134 2 ENDCASE  
135 2 CALL XERN TO RETURN FROM SEGMENT  
136 1 END XDCLU
FORTRAN CALLING PROCEDURE

CALL XDCLE

XDCLE PURGES ALL DATA FROM THE AWA. THE ONLY ELEMENTS REMAINING ARE PERMANENT SYSTEM TABLES AND DATA BASE FILES (CHAINS 1 & 8).

INPUT

XC COMMON - CARTAG, FLASS, LU
MANAGER - AWA HEADER AND TOC (SEE XMANA)

OUTPUT

XC COMMON - REBUFF, REOPTN, COMBUF-SCRATCH
XB COMMON - SCRATCH BEYOND XB(200)
XS COMMON - SCRATCH
MANAGER - COMMANDS TO PURGE AWA AND RESTORE CHAINS 0 AND 8

LOCAL VARIABLES

AVA - AWA HEADER AND TOC RECEIVED FROM MANAGER
HD1 - TOC CHAIN 1 HEAD
HD3 - TOC CHAIN 3 HEAD
HD5 - TOC CHAIN 5 HEAD
NEXT - INDEX TO NEXT ALLOCATABLE EIGHT WORD ENTRY IN QUEUE
NMAX - MAXIMUM SIZE OF QUEUE
ORG - BASE REFERENCE ADDRESS FOR AWA ADDRESSES
QUEUE - BUFFER FOR CONSTRUCTING AWA RESTORE REQUESTS FOR NON-DELETED ITEMS
TOCMAX - MAXIMUM SIZE OF TOC ACCOMMODABLE BY XDCLE

NOTES

IN THE EVENT THAT A COMPACTED AWA TOC WILL NOT FIT IN THE ALLOCATED BUFFER SPACE AND ELEMENTS OF CHAIN 1 OR 8 EXTEND BEYOND IT, THE CLEAR FUNCTION WILL NOT BE PERFORMED.
SHOULD THE RESTORATION OF CHAIN 1 OR 8 FAIL AFTER THE CLEAR REQUEST HAS BEEN COMPLETED BY THE MANAGER, FB5 WILL BE TERMINATED IN ORDER TO PROTECT USER DISK FILES.
188 1 BEGIN XDCL
189 2 RETRIEVE TOC
190 2 BUILD REQUEST TO CLEAR AWS
191 2 DO UNTIL END OF PERMANENT SYSTEM TABLES CHAIN (CHAIN 1)
192 2 EXIT TO :ERR24; IF CHAIN POINTS BEYOND TOC BUFFER
193 2 EXIT TO :ERR46; IF RESTORATION REQUEST QUEUE IS FULL
194 2 BUILD REQUEST TO REALLOCATE TABLE
195 2 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
196 2 BUILD REQUEST TO STORE TABLE INTO AWS FROM SAM
197 2 ENDDO
198 2 DO UNTIL END OF DATA BASE FILES CHAIN (CHAIN 8)
199 2 EXIT TO :ERR24; IF CHAIN POINTS BEYOND TOC BUFFER
200 2 EXIT TO :ERR46; IF RESTORATION REQUEST QUEUE IS FULL
201 2 BUILD REQUEST TO REALLOCATE TOC ENTRY
202 2 ENDDO
203 2 BUILD REQUEST TO TERMINATE LIST
204 2 DO UNTIL END OF DRDE CHAIN (CHAIN 3)
205 3 IF CHAIN POINTS BEYOND TOC BUFFER
206 4 THEN
207 5 OUTPUT X013 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
208 5 EXIT PURGE LOOP
209 5 ENDF
210 5 PURGE FILE
211 2 ENDDO
212 2 DO FOR EACH BLOCK OF EIGHT REQUESTS
213 3 TRANSMIT BLOCK TO MANAGER
214 2 EXIT TO :ERR23; IF REQUESTS FAILED
215 2 ENDDO
216 2 EXIT XDCL
217 2 :ERR23;
218 3 DO FROM FAILING REQUEST TO END OF LIST
219 3 IF REQUEST TO STORE
220 4 THEN
221 5 READ SAM TO FREE BUFFER AND CLASS NUMBER
222 5 ENDF
223 2 ENDDO
224 2 DO UNTIL END OF DATA BASE FILE CHAIN (CHAIN 8)
225 3 IF FILE IS UTDB (TYPE 1)
226 4 THEN
227 5 CALL PURGE TO DELETE FILE
228 5 ENDF
229 2 ENDDO
230 2 PURGE ALL UTDB FILES
231 2 TERMINATE FDS WITH CLEAR FAILURE MESSAGE
232 2 :ERR46;
233 2 :ERR24;
234 2 DO FOR ALL STORE REQUESTS BUILT
235 2 READ SAM TO FREE BUFFER AND CLASS NUMBER
236 2 ENDDO
237 1 EXIT XDCL WITH CLEAR FAILURE MESSAGE
238 1 END XDCL
240 1C*********
241 1C00  FORTRAN CALLING PROCEDURE
242 1C00  CALL XDCOP
243 1C00  
244 1C00  
245 1C00  
246 1C00  
247 1C00  XDCOP PROCESSES A USER REQUEST TO COPY A SEQUENCE TABLE,
248 1C01  INTERFACE TABLE, DATA ELEMENT, DRE, HITOB, OR FOR. ONLY
249 1C01  THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S AUA CAN BE COPIED.
250 1C01  
251 1C01  
252 1C01  
253 1C01  
254 1C01  INPUT
255 1C01  COMMON XE - CARTAG, COMBUF, QUAL, REQPTR, TOKENS
256 1C01  
257 1C01  
258 1C01  
259 1C01  
260 1C01  OUTPUT
261 1C01  COMMON XE - COMPTR, RERBUF
262 1C01  
263 1C01  
264 1C01  
265 1C01  INTERNAL VARIABLES
266 1C01  
267 1C01  
268 1C01  COMMON XS - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE)
269 1C01  (2) DBTYPE: TYPE CODE FOR DATA BASE FILES
270 1C01  (3) I: INDEX
271 1C01  (4) IDC: EXEC BUFFER
272 1C01  (148) IERR: FILE MANAGER ERROR RETURN
273 1C01  (149) IMSG: ERROR MESSAGE NUMBERS
274 1C01  (150) FNNAME: NEW FMGR FILE NAME
275 1C01  (153) WMNAME: OLD FMGR FILE NAME
276 1C01  (154) OMNAME: OLD NAME IN COMBUF
277 1C01  (157) ONMNAME: NEW NAME IN COMBUF
278 1C01  (158) TOCENT: TOC ENTRY FOR DATA BASE RENAME
279 1C01  (166) EOF: EOF RETURN FROM FMGR
280 1C01  
281 1C01  COMMON XB - (204) IPBUFF: USER BUFFER FOR EXEC I/O (1024)
282 1C01  (124A) FNTYPE: FMGR FILE TYPE
283 1C01  (1245) PARM1: FIRST BYTE OF REQPTR
284 1C01  (1246) PARM2: SECOND BYTE OF REQPTR
285 1C01  (1247) PREFIX: PREFIX FOR FILE NAME
286 1C01  (1248) SCRTH: FILE SECURITY CODE (Q OR 88)
287 1C01  (1249) SIZE: FILE SIZE IN BLOCKS
288 1C01  (1251) IDCBY: EXEC BUFFER
289 1C01  
290 1C01  
291 1C01  Routines Used -
292 1C01  
293 1C01  CLOSE, CREAT, EXEC, OPEN, PURGE, READF, WRITF, XDOBA, XDDOD,
294 1C01  XREQ, XRHOV, XRMIG, XRFIN, XRSFE
1 BEGIN XDCOP
2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
3 SAVE POINTER TO CURRENT NAME
4 IF NEXT TOKEN IS HYPHEN, THEN
5 DECODE CLASS NAME
6 EXIT TO :CLASSR: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,B)
7 ELSE
8)
9 ENDIF
10 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
11 SAVE POINTER TO NEW NAME
12 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT END-OF-MESSAGE
13 IF CLASS IS DATA BASE OR ORDE, THEN
14 EXIT IF NEW NAME IS MORE THAN 4 CHARACTERS :NAME:
15 IF CLASS IS DATA BASE, THEN
16 EXIT TO :NAME: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
17 ENDIF
18 BUILD ADA MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
19 BUILD ADA MANAGER REQUEST TO VERIFY NEW NAME
20 CALL XREN
21 EXIT TO :TOCRR: IF RETURN INDICATES ERROR ON FIRST REQUEST
22 EXIT TO :TOCRR: IF NO ERROR RETURNED ON SECOND REQUEST
23 CALL EXEC TO GET TOC ENTRY
24 IF CLASS IS DATA BASE THEN
25 IF TYPE IS POD, THEN
26 CALL XDDBA TO ADD NEW POD TO LOG FILE (XPDB)
27 EXIT TO :FILERR: IF FMGR ERROR RETURNED
28 EXIT TO :TOCRR: IF NEW NAME IS DUPLICATE
29 EXIT TO :MAXERR: IF POD MAX IS EXCEEDED
30 SET FILE PREFIX TO RIGHT BRACKET
31 ELSE
32 EXIT TO :INVLD: IF FILE IS NOD
33 SET FILE PREFIX TO *
34 ENDIF
35 SET FMGR FILE TYPE TO 1
36 SET SECURITY CODE TO 88
37 ELSE
38 SET FILE PREFIX TO /
39 SET SECURITY CODE = 0
40 SET FMGR FILE TYPE FROM TOC ENTRY
41 IF FILE IS TYPE 2, THEN
42 STORE RECORD LENGTH FROM TOC ENTRY
43 ENDIF
44 CALL XREFM TO FORMAT NEW FILE NAME
45 CALL CREATE FOR NEW FILE
46 EXIT TO :FILERR: IF FMGR ERROR RETURNED
47 CALL XREFM TO FORMAT OLD FILE NAME
48 CALL OPEN FOR OLD FILE
49 EXIT TO :FILERR: IF FMGR ERROR RETURNED
50 EXIT TO :TPERR: IF TYPE IS NOT SAME AS IN TOC
51 DO UNTIL END-OF-FILE IS READ ON OLD FILE
52 CALL READF TO READ RECORD FROM OLD FILE
53 EXIT IF ERROR DETECTED TO :FILERR:
54 CALL WRITE TO WRITE RECORD TO NEW FILE
55 EXIT IF ERROR DETECTED TO :FILERR:
56 END DO
57 CALL CLOSE FOR OLD FILE
58 CALL CLOSE FOR NEW FILE
59
358 BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY
359 EXIT TO :TOCERR: IF ERROR IS INDICATED
360 ELSE
361 BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
362 BUILD MANAGER REQUEST FOR DATA RETRIEVAL
363 CALL XREQ
364 EXIT TO :TOCERR: IF ERROR IS INDICATED
365 CALL EXEC TO GET TOC ENTRY
366 BUILD MANAGER REQUEST TO ALLOCATE NEW TABLE
367 BUILD MANAGER REQUEST TO STORE TABLE
368 IF CLASS IS INTERFACE TABLE, THEN
369 CALL EXEC TO WRITE NEW TABLE NAME TO SAM
370 BUILD MANAGER REQUEST TO STORE NEW NAME IN TABLE
371 ENDIF
372 CALL XREQ
373 EXIT TO :TOCERR: IF ERROR IS INDICATED
374 ENDIF
375 1 EXIT XDCOP
376 2 :SYNTAX: CALL XRMG -"SYNTAX ERROR ..." AND EXIT
377 2 :CLASER: CALL XRMG -"INVALID CLASS DESIGNATOR ..." AND EXIT
378 2 :NAMEERR: CALL XRMG -"NEW NAME IS INVALID ..." AND EXIT
379 2 :MAXERR: CALL XRMG -"AUTHORIZED LIMIT ..." AND EXIT
380 2 :INVAL: CALL XRMG -"NOB CANNOT BE ..." AND EXIT
381 2 :FILERR: CALL XRMG -"FILE ACCESS ERROR ..." ON ........ AND EXIT TO :END:
382 2 :TOCERR: CALL XRMG TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
383 2 :TYPEERR: CALL XRMG -"INCONSISTENT FILE TYPE ..."
384 2 :END:
385 2 IF POD HAS BEEN LOGGED IN XPDG, THEN
386 3 CALL XDOD TO DELETE POD FROM XPDG
387 2 ENDIF
388 2 IF A NEW FILE HAS BEEN BUILT, THEN
389 3 PURGE NEW FILE
390 3 CLOSE OLD FILE
391 2 ENDIF
392 1 END XDCOP
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>394</td>
<td>1 *00</td>
</tr>
<tr>
<td>395</td>
<td>1 *00</td>
</tr>
<tr>
<td>396</td>
<td>1 *00</td>
</tr>
<tr>
<td>397</td>
<td>1 *00</td>
</tr>
<tr>
<td>400</td>
<td>1 *00</td>
</tr>
<tr>
<td>401</td>
<td>1 *01</td>
</tr>
<tr>
<td>404</td>
<td>1 *01</td>
</tr>
<tr>
<td>405</td>
<td>1 *01</td>
</tr>
<tr>
<td>410</td>
<td>1 *02</td>
</tr>
<tr>
<td>411</td>
<td>1 *02</td>
</tr>
<tr>
<td>412</td>
<td>1 *02</td>
</tr>
<tr>
<td>413</td>
<td>1 *02</td>
</tr>
<tr>
<td>414</td>
<td>1 *02</td>
</tr>
<tr>
<td>415</td>
<td>1 *02</td>
</tr>
<tr>
<td>416</td>
<td>1 *02</td>
</tr>
<tr>
<td>417</td>
<td>1 *03</td>
</tr>
<tr>
<td>418</td>
<td>1 *03</td>
</tr>
<tr>
<td>419</td>
<td>1 *03</td>
</tr>
<tr>
<td>420</td>
<td>1 *03</td>
</tr>
<tr>
<td>421</td>
<td>1 *03</td>
</tr>
<tr>
<td>422</td>
<td>1 *03</td>
</tr>
<tr>
<td>423</td>
<td>1 *03</td>
</tr>
<tr>
<td>424</td>
<td>1 *03</td>
</tr>
<tr>
<td>426</td>
<td>1 *04</td>
</tr>
<tr>
<td>429</td>
<td>1 *04</td>
</tr>
<tr>
<td>430</td>
<td>1 *04</td>
</tr>
<tr>
<td>431</td>
<td>1 *04</td>
</tr>
<tr>
<td>433</td>
<td>1 *04</td>
</tr>
<tr>
<td>436</td>
<td>1 *05</td>
</tr>
<tr>
<td>437</td>
<td>1 *05</td>
</tr>
<tr>
<td>438</td>
<td>1 *05</td>
</tr>
<tr>
<td>439</td>
<td>1 *05</td>
</tr>
<tr>
<td>440</td>
<td>1 *05</td>
</tr>
<tr>
<td>442</td>
<td>1 *05</td>
</tr>
<tr>
<td>443</td>
<td>1 **********</td>
</tr>
</tbody>
</table>
1 BEGIN XDPA
2 STORE RETURN ADDRESS
3 CALL .ENTRY TO SET UP CALLING ARGUMENTS
4 CALL IRC FOR EXCLUSIVE USE OF XPDB
5 CALL OPEN FOR EXCLUSIVE USE OF XPDB
6 EXIT TO :FILERR: IF ERROR RETURNED
7 COMPUTE RECORD NUMBER FOR USER'S DIRECTORY (QVAL-77B)/2+1
8 CALL READ FOR RECORD COMPUTED
9 EXIT TO :FILERR: IF ERROR RETURNED
10 DETERMINE PART OF RECORD TO BE USED
11 IF REQUEST IS FOR ADD, THEN
12 EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
13 START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
14 EXIT TO :NAMERR: IF ENTRY NAME MATCHES PARAMETER NAME
15 ENDLOOP
16 ENDSERCH
17 STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
18 INCREMENT # OF CURRENT ENTRIES
19 ELSE
20 START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
21 EXIT IF ENTRY NAME MATCHES PARAMETER NAME
22 ENDLOOP
23 EXIT TO :NAMERR:
24 ENDSERCH
25 IF REQUEST IS FOR VERIFY, THEN
26 IF ERROR RETURNED, THEN
27 REPLACE ENTRY WITH LAST ENTRY
28 STORE ERRORS IN LAST ENTRY
29 DECRENENT # OF CURRENT ENTRIES
30 ENDIF
31 ENDTF
32 CALL WRIT TO WRITE RECORD TO XPDB
33 EXIT TO :RETURN:
34 :NAMERR: SET IERR = 1 AND EXIT TO :RETURN:
35 :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
36 :FILERR: SET IERR = FNRG ERROR CODE
37 :RETURN: CALL CLOSE FOR XPDB
38 :CALL XHRLK TO RETURN RESOURCE #
39 END XDPA
486 1 C**********
487 1 CRO            FORTRAN CALLING PROCEDURE
488 1 CRO
489 1 CRO          CALL XDELE
490 1 CRO
491 1 CRO
492 1 C**********
493 1 CDO          XDELE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT
494 1 C01           SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE ANA.
495 1 C01          IF THE ELEMENT IS A DATA BASE, THE ASSOCIATED FILE
496 1 C01          MANAGER FILE IS PURGED AND FOR A PDB THE PDB DIRECTORY
497 1 C01          IS UPDATED
498 1 C01
499 1 C01
500 1 C**********
501 1 C02            INPUT
502 1 C02
503 1 C02          COMMON XE = COMBUF, COMPTR, LU, QUAL, TOKENS
504 1 C02
505 1 C02
506 1 C**********
507 1 C03            OUTPUT
508 1 C03
509 1 C03
510 1 C03          COMMON XE = REGBUF
511 1 C03
512 1 C**********
513 1 C05
514 1 C05          NOTES
515 1 C05
516 1 C05          ROUTINES USED
517 1 C05
518 1 C05          EXEC
519 1 C05           IAND
520 1 C05           PURGE
521 1 C05           XDDDB
522 1 C05           XRIN
523 1 C05           XREST
524 1 C05           XREL
525 1 C05           XMOV
526 1 C05           XMSG
527 1 C05           XRCK
528 1 C05           XDEL
529 1 C05           XUPK
530 1 C05           XUDOG
531 1 C05
532 1 C**********
1 BEGIN XDELETE
2 DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED
3 EREREX IF COMMA IS NOT NEXT LEXICAL ELEMENT :ERROR:
4 IF CLASS DESIGNATOR IS SPECIFIED, THEN
5 SET REQUESTED CLASS APPROPRIATELY (B, S, I, D, OR F)
6 ELSE
7 SET REQUESTED CLASS TO BE (O)
8 ENDIF
9 IF DATA BASE TO BE DELETED, THEN
10 BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY
11 ENDIF
12 IF ELEMENT IS NOT A MASTER DATA BASE, THEN
13 IF ELEMENT IS A PERSONAL DATA BASE, THEN
14 CALL PROD TO DELETE THIS PD FROM XPDB
15 IF ERROR IS RETURNED, THEN
16 CALL XRMG - "FILE ACCESS ERROR @.. XPDB"
17 SET ERROR FLAG
18 ENDIF
19 ENDIF
20 BUILD AND ISSUE AWA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
21 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
22 CALL XRMG - "XXXX NOT FOUND"
23 SET ERROR FLAG
24 ENDIF
25 IF CLASS IS DATA BASE (C), OR
26 CLASS IS IDDE (F), THEN
27 IF ERROR FLAG IS NOT SET, THEN
28 CALL XRMG TO CONSTRUCT FILE NAME
29 ISSUE RTE PURGE FOR THE FILE
30 IF RETURN CODE FROM PURGE, THEN
31 CALL XRMG - "FILE ERROR WHEN XXXX"
32 ENDIF
33 ENDIF
34 ELSE
35 CALL XRMG - "... IS A MDO. NOT DELETED."
36 ENDIF
37 ENDDO
38 1 EXIT TO :RETURN:
39 2 :ERROR: CALL XRMG - "SYNTAX ERROR"
40 2 :RETURN:
41 1 END XDELETE
CALL XLIS (TOCLST)

XLIS PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECORATING EACH ELEMENT AND FLAGGING IT IN TOCLST AS AN ELEMENT
TO BE STORED/RESTORED.

INPUTS IN CALLING SEQUENCE:

TOCLST - (INTEGER, 1200 WORDS) ADDRESS OF FIRST TOC ENTRY
OF A LIST OF POSSIBLE ELEMENTS' TOC ENTRIES

INPUTS IN COMMON:

XB(151) ABLG, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

TOCLST - (INTEGER, 1200 WORDS) LIST OF TOC ENTRIES; THOSE
 THAT ARE TO BE STORED/RESTORE ARE FLAGGED.

OUTPUTS IN COMMON:

XB(151) ABLG, XB(157) TOTAL, XB(158) TOTWPD

INTERNAL XB COMMON USED:

XB(151) ABLG - (INTEGER, 1 WORD) ABORT FLAG
XB(152) EBLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TOTAL - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
BE STORED/RESTORED
XB(158) TOTWPD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
BE STORED/RESTORED
XB(198) RETC - (INTEGER, 1 WORD) XCOM RETURN CODE
XB(199) RCSTS - (INTEGER, 1 WORD) CLASS OF DATA BEING SEARCHED
XP IN LIST OF TOC ENTRIES
XB(200) HAMTPR - (INTEGER, 1 WORD) POINTER TO NAME IN COMBUF
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCLST

COMMON USED:

EQUIVALENCE

+ (RE(5), HASS), (RE(8), EOS )
+ (RE(85), NAME ), (RE(92), HTPN )
+ (RE(113), COMMA ), (RE(144), COMTRA )
+ (RE(145), COMBUF )
BEGIN XLIS

DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR

TURN ERFLAG OFF

:RTN1:

DO UNTIL EOS IS SENSED IN COMBUS

IF NEXT TOKEN IS NOT "NAME" TO :ERR1:

SAVE INDEX TO NAME FIELD

INCREMENT TO NEXT TOKEN

IF TOKEN IS A HYPHEN THEN

ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:

INCREMENT TO NEXT TOKEN

IF CLASS NAME (I, S, D, F)

ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:

SET CLASS TO CLASS SPECIFIED

ELSE

SET CLASS TO DATA ELEMENT

ENDIF

IF XLIS CALLED FROM STORE THEN

ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:

ENDIF

ERREXIT IF NAME/CLASS ENTRY NOT FOUND IN TOC TO :ERR2:

CALL ARSET TO TURN STORE/RESTORE BIT ON

INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT

ENDDO

:RTN2:

IF ERROR FLAG IS ON THEN

CALL XCOM TO REPROMPT USER TO CONTINUE

ERREXIT IF RESPONSE IS X TO :ERR3:

ENDIF

ENDDO

EXIT XLIS

:ERR1:

SET ERROR FLAG ON

CALL XMSG TO DISPLAY SYNTAX ERROR

GO TO :RTN2:

:ERR2:

IF ERROR FLAG IS OFF THEN

TURN ERROR FLAG ON

CALL XMSG TO DISPLAY NOT STORED/RESTORED MESSAGE

ENDIF

CALL XEMSG TO DISPLAY ELEMENT NAME

GO TO :RTM1:

:ERR3:

SET ABFLG TO ABORT STORE/RESTORE OPERATION

END XLIS
FORTRAN CALLING PROCEDURE

CALL XLST

PROCESS THE LIST DIRECTIVE TO LIST ALL TABLES AND/OR DATA ELEMENTS SPECIFIED.

COMMON XE - COMBUF, COMPTR, FLAGS, LU, TOKENS

PRINT TO LOGICAL UNIT 6 OR TO LOGICAL UNIT 'LU'

INTERNAL VARIABLES

COMMON XS - BUFFER = PRINT LINE TO BE OUTPUT
BUFFTR = INDEX INTO BUFFER FOR NEXT ASCII DATA

COMMON XD - POINTR = CHAIN POINTER TO NEXT (OR 1ST) TOC

USES ROUTINES - EXEC, XEINT, XILSO, XILST,
XREP, XREX, XRMV, XRMG,
XRPCK, XRPK, XSLST, OPEN,
CLOSE, READY

NOTES
783 1 CD**********
784 1 CD0      FORTRAN CALLING PROCEDURE
785 1 CD0      CALL XDOFF
786 1 CD0      CD**********
787 1 CD1      XDOFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
788 1 CD1      DELETES ALL ORDE AND UTOB FILES LOGGED IN THE ANA,
789 1 CD1      PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
790 1 CD1      RETURNS NORMAL PATHS TO THE FDS MANAGER AND TERMINATES
791 1 CD1      NORMALLY VIA RTE.
792 1 CD1      CD**********
793 1 CD2      INPUT
794 1 CD2      CD0      COMMON XE - LU, FLAGS, QUAL, RERBUF
795 1 CD2      CD0      COMMON XB - ORG = ORIGIN ADDRESS OF ANA. USED TO CALCULATE
796 1 CD2      CD0      INDICES INTO 'ANA' FROM ADDRESS POINTERS
797 1 CD2      CD0      OF TOC ENTRIES
798 1 CD2      CD0      ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND
799 1 CD2      CD0      TOC RETRIEVED VIA XREQ
800 1 CD2      CD**********
801 1 CD4      INTERNAL VARIABLES
802 1 CD4      CD**********
803 1 CD5      ROUTINE USED - EXEC, PURGE, XDSTA, XPIT, XREQ, XREXT, XRI6, XRNV, XRM5G, XRFM, XRPCK, XRSET, XRUPK, XICOM, XUDBG, XVABN
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL XREQ TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR DDRE FILES IS NOT NEGATIVE THEN
6 DO UNTIL DDRE CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
8 OUTPUT 'XDI3 TOC TOO LARGE, PURGE INCOMPLETE'
9 EXIT DO
10 ENDF
11 CALL XREQF TO CREATE FILE NAME '/XXXQ'
12 CALL PURGE TO SCRATCH FILE
13 SET DDRE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
14 ENDDO
15 ENDF
16 IF CHAIN HEAD FOR DATA BASES IS NOT NEGATIVE, THEN
17 DO UNTIL DATA BASE CHAIN HEAD IS NEGATIVE
18 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
19 OUTPUT 'XDI3 TOC TOO LARGE, PURGE INCOMPLETE'
20 EXIT DO
21 ENDF
22 IF TYPE OF DATA BASE IS UTOB, THEN
23 CALL XREQF TO CREATE FILE NAME '/XXXQ'
24 CALL PURGE TO SCRATCH FILE
25 ENDF
26 SET DATA BASE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
27 ENDDO
28 ENDF
29 CALL XDATA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XUBGC
32 ENDF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVABN - NO RETURN FROM THIS CALL
35 ENDF
36 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
37 CALL XEXIT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE EXEC
38 ENDF
39 RETURN
40 END XDOFF
```
871 1 C0***********
872 1 C00
873 1 C00      FORTRAN CALLING SEQUENCE:
874 1 C00
875 1 C00      CALL XDRE (DATBUF, DBDCB)
876 1 C00
877 1 C00***********
878 1 C01      XDRE HANDLES THE RESTORING OF DATA ELEMENTS, SEQUENCE TABLES
879 1 C01 AND INTERFACE TABLES
880 1 C01
881 1 C01***********
882 1 C02      INPUTS FROM CALLING SEQUENCE:
883 1 C02
884 1 C02
885 1 C02      DATBUF - (INTEGER, 1400 WORDS) BUFFER USED TO READ IN RECORDS
886 1 C02 OF DATA FROM DATA BASE FILE
887 1 C02
888 1 C02      DBDCB - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DCB
889 1 C02
890 1 C02
891 1 C02      INPUTS FROM XB COMMON:
892 1 C02
893 1 C02      XB(154) ALLFLG, XB(175) ENBLK, XB(182) TOTSIZ,
894 1 C02      XB(201) NOTOC
895 1 C02
896 1 C02***********
897 1 C03      OUTPUTS IN CALLING SEQUENCE:
898 1 C03
899 1 C03
900 1 C03      DATBUF, DBDCB
901 1 C03
902 1 C03      OUTPUTS IN XB COMMON
903 1 C03
904 1 C03      XB(151) ABLFLG, XB(152) ERRFLG, XB(165) TOCHDS,
905 1 C03      XB(169) TOPBLK, XB(170) ENBLK, XB(171) TOCHDX,
906 1 C03      XB(201) NOTOC
907 1 C03
908 1 C03***********
909 1 C04      INTERNAL XB COMMON USED:
910 1 C04
911 1 C04      XB(151) ABLFLG - ABORT FLAG
912 1 C04      XB(152) ERRFLG - ERROR MESSAGE FLAG
913 1 C04      XB(155) ALLFLG - RESTORE ALL UDBF FLAG
914 1 C04      XB(155) DEBUG - DEBUG FLAG
915 1 C04      XB(175) FILHNM - DB FILE NAME
916 1 C04
917 1 C04      XB(162) DATBLK - BLOCK # WHERE DATA ITEM BEGINS
918 1 C04      XB(163) DATHDX - WORD INDEX INTO DATBUF WHERE DATA BEGINS
919 1 C04      XB(164) IERR - ERROR FLAG FOR FMGR CALLS
920 1 C04
921 1 C04      XB(165) TOCHDX - WORDS OF TOC IN TOCBUF (NOTOC =8) + 1
922 1 C04
923 1 C04      XB(167) LEM - # WORDS OF DATA TO MOVE
924 1 C04      XB(168) DATEN - BLOCK # WHERE DATA ITEM ENDS
925 1 C04      XB(169) TOPBLK - BLOCK # OF FIRST BLOCK IN DATBUF
926 1 C04      XB(169) ENBLK - BLOCK # OF LAST BLOCK READ
927 1 C04      XB(171) TOCHDX - INDEX INTO TOC OF CURRENT DATA ITEM
928 1 C04      XB(175) IL - # WORDS OF DATA TO READ
929 1 C04      XB(176) TOCENT - DATA BASE TOC ENTRY
930 1 C04
```
1 BEGIN XDRDE
2 INITIALIZE FILE INDICES TO INDICATE NO DATA IN DATBUF
3 INITIALIZE REQUEST BUFFER TO SAY NO REQUESTS
4 DO WHILE THERE ARE NON-DREAD FILES TO PROCESS
5 IF ALLFLG IS ZERO OR IF STORE/RESTORE BIT IS ON THEN
6 IF DATA IS NOT CURRENTLY IN DATBUF THEN
7 CALL READ TO READ 1 BUFFER BEGINNING WITH DATBLK FOR THIS ELEMENT
8 ERREXIT IF READ ERROR TO :ERR1:
9 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
10 ELSE, DATA BEGINS IN DATBUF
11 IF DATA DOES NOT END IN DATBUF THEN
12 CALL XMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
13 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
14 CALL READY TO READ ENOUGH TO FILL DATBUF
15 ERREXIT IF READ ERROR TO :ERR1:
16 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
17 ENDF
18 ENDIF
19 BUILD AVA REQUEST TO ALLOCATE AND STORE DATA
20 CALL EXEC TO WRITE DATA TO SAN
21 ERREXIT IF ERROR FROM EXEC TO :ERREXIT:
22 IF AVA REQUEST BUFFER IS FULL THEN
23 CALL XDRDE TO ISSUE REQUEST
24 ENDF
25 ENDF
26 ENDDO
27 1 EXIT XDRDE
28 :ERR1:
29 CALL XR16 TO CONVERT ERROR CODE TO ASCII
30 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
31 GO TO :ERR3:
32 :ERR2:
33 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
34 :ERR3:
35 SET ABFLG TO SAY ABORT RESTORE
36 1 END XDRDE
FORTRAN CALLING SEQUENCE:

CALL XDRUF (DATBUF, DBDCB)

XDRUF HANDLES THE RESTORING OF DREDE FILES FROM A HDD/UTDB FILE

INPUTS IN CALLING SEQUENCE:

DATBUF - (INTEGER, 148 WORDS) BUFFER USED TO READ IN RECORDS OF DREDE FILES FROM DATA BASE FILE

DBDCB - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DBC

OUTPUTS IN CALLING SEQUENCE:

OUTPUTS IN XB COMMON:

XB(151) ABFLG

INTERNAL XB COMMON USED:

XB(151) ABFLG - ABORT FLAG

XB(152) ERFLE - ERROR MESSAGE FLAG

XB(154) ALLFLG - RESTORE ALL UTDB FLAG

XB(155) DEBU - DEBUG FLAG

XB(159) FIELMN - DB FILE NAME

XB(162) DATBLK - BLOCK # WHERE ORDE BEGINS

XB(163) DATNOX - WORD INDEX INTO DATBUF WHERE ORDE BEGINS

XB(164) IERR - ERROR FLAG FOR FNGR CALLS

XB(165) TOCH- # WORDS OF TOC IN TOCBUF (HTOC #) + 8

XB(166) NBLKS - # BLOCKS OF UTDB/DRDE TO READ/WRITE

XB(167) LEN - # WORDS OF DATA TO MOVE

XB(169) TOPBLK- BLOCK # OF FIRST BLOCK IN DATBUF

XB(170) ENDBLK- BLOCK # OF LAST BLOCK READ

XB(171) TOCHNOX- INDEX TO CURRENT TOC ENTRY

XB(172) DDFIL- DRDE FILE NAME

XB(175) IL - # WORDS OF DATA TO READ/WRITE

XB(201) TOCBUF- TOC BUFFER

COMMON USED:
1061  1 BEGIN XRDF
1062  2 DO WHILE THERE ARE TOC ENTRIES TO PROCESS
1063  3 IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
1064  4 CALL XRDFN TO CREATE THE DDE FILE NAME
1065  5 CALL CREAT TO CREATE THE DDE FILE
1066  6 IF THERE WAS A CREATE ERROR THEN
1067  7 IF ERROR FLAG IS OFF THEN
1068  8 SET ERROR FLAG ON
1069  9 CALL XRMSG TO DISPLAY MAIN MESSAGE
1070 10 ENDIF
1071 11 CALL XR16 TO CONVERT ERROR CODE TO ASCII
1072 12 CALL EXEC TO DISPLAY DDE NAME AND ERROR CODE
1073 13 ELSE
1074 14 CALL CLOSE TO CLOSE FILE
1075 15 ERREXIT IF CLOSE ERROR TO :ERR1:
1076 16 CALL OPEN TO OPEN DDE FILE AS TYPE 1
1077 17 ERREXIT IF OPEN ERROR TO :ERR1:
1078 18 DO UNTIL ALL BLOCKS OF DDE HAVE BEEN PROCESSED
1079 19 IF DATA FOR DDE IS IN DATBUF THEN
1080 20 CALL WRITF DATA TO DDE
1081 21 ERREXIT IF WRITF ERROR TO :ERR1:
1082 22 ELSE
1083 23 CALL READF TO READ NEXT BUFFER OF DATA
1084 24 ERREXIT IF READF ERROR TO :ERR2:
1085 25 SET INDICES INDICATING DATA IN BUFFER
1086 26 ENDF
1087 27 EENDO
1088 28 CALL CLOSE TO CLOSE DDE FILE
1089 29 ERREXIT IF CLOSE FAILED TO :ERR1:
1090 30 BUILD DMA REQUEST TO ALLOCATE DDE IN DMA
1091 31 IF DMA REQUEST BUFFER IS FULL THEN
1092 32 CALL XDREM TO MAKE REQUEST
1093 33 EXIT XRDF IF XDREM ERROR
1094 34 ENDF
1095 35 EENDF
1096 36 ENDR
1097 37 EENDO
1098 38 Exit XRDF
1099 39 :ERR1:
1100 40 CALL PURGE TO PURGE DDE FILE
1101 41 :ERR2:
1102 42 CALL XRMSG TO DISPLAY ERROR MESSAGE WITH FILE NAME
1103 43 SET ABFLG TO SAY ABORT RESTORE
1104 44 END XRDF
1106 1 C*********
1107 1 C  FORTRAN CALLING PROCEDURE
1108 1 C
1109 1 C
1110 1 C CALL XDREC
1111 1 C
1112 1 C*********
1113 1 C81
1114 1 C81 XDREC PROCESSES THE RECALL DIRECTIVE. A UTD0 IS CREATED AND
1115 1 C81 THE CONTENTS OF THE SPECIFIED PDD ARE COPIED TO IT.
1116 1 C81
1117 1 C*********
1118 1 C82 INPUT
1119 1 C82
1120 1 C82 COMMON XE - CARTAG, COMMUF, CUMTPR, FLGS, LV, TOKENS
1121 1 C82
1122 1 C82 FILES - )XXX (PDD FILE SPECIFIED)
1123 1 C82
1124 1 C82
1125 1 C*********
1126 1 C83 OUTPUT
1127 1 C83
1128 1 C83 COMMON XE - REMUF, REOPTR
1129 1 C83
1130 1 C83 FILES - )XXX (UTDO FILE SPECIFIED)
1131 1 C83
1132 1 C83
1133 1 C*********
1134 1 C84 INTERNAL VARIABLES
1135 1 C84
1136 1 C84
1137 1 C84 DCPPDD - DCB FOR THE PDD FILE; ALLOCATED IN XB COMMON;
1138 1 C84 CONTAINS 1152 WORD BUFFER USED TO READ THE PDD
1139 1 C84 AND TO WRITE THE UTD0
1140 1 C84 DCBBUD - DCB FOR THE UTD0 FILE; ALLOCATED IS XS COMMON
1141 1 C84
1142 1 C*********
1200 2 :ERROR7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERROR9: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERR16: ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERR18: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
1204 2 :ERR19: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERR21: ISSUE MESSAGE - "AVM OVERFLOW - XXXX NOT LOGGED"
1206 2 :ERR22: ISSUE MESSAGE - "XXXX ALREADY EXISTS"
1207 2 :ERR44: ISSUE MESSAGE - "FILE ACCESS ERROR &--- XPDB"
1208 2 :RETURN:
1209 2 IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
1210 3 PURGE UTDB
1211 2 ENDIF
1212 2 IF STATUS FLAG INDICATES UTDB IS LOGGED IN AWA, THEN
1213 3 CALL XER TO DELETE UTDB FROM AWA
1214 2 ENDIF
1215 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1216 3 CALL CLOSE FOR PDB FILE
1217 2 ENDIF
1218 1 "NO XDREC
1220 1 ******
1221 1 C00  FORTRAN CALLING PROCEDURE
1222 1 C00  CALL XOREN
1223 1 C00  ******
1224 1 C01  XOREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE,
1225 1 C01  INTERFACE table, DATA ELEMENT, ORDE, UTDB, OR PDB. ONLY
1226 1 C01  THOSE TABLES OR ELEMENTS LOGGED IN THE USERS AWA ARE RENAMED.
1227 1 C01  ******
1228 1 C02  INPUT
1229 1 C02  COMMON XE - CARTAG, COMBUF, COMPTR, QUAL, REPSTR, TOKENS
1230 1 C02  ******
1231 1 C03  OUTPUT
1232 1 C03  COMMON XE - REBUF
1233 1 C03  ******
1234 1 C04  INTERNAL VARIABLES
1235 1 C04  COMMON XS - (2) DTYPF: TYPE CODE FOR DATA BASE FILES
1236 1 C04  (3) J: INDEX
1237 1 C04  (148) IERR: FILE MANAGER ERROR RETURN
1238 1 C04  (149) JMSC: ERROR MESSAGE NUMBERS
1239 1 C04  (150) FNAMF: NEW FNRG FILE NAME
1240 1 C04  (153) LLCPTR: POINTER TO NEW NAME IN COMBUF
1241 1 C04  (154) OLCPTR: OLD FGRT FILE NAME
1242 1 C04  (157) OTCPTR: POINTER TO OLD NAME IN COMBUF
1243 1 C04  (158) TOTENT: TOC ENTRY FOR DATA BASE RENAME
1244 1 C04  (160) PATELS: DATA CLASS CODE (STORED IN LEFT BYTE)
1245 1 C04  COMMON XB - (201) IDC8: EXEC BUFFER
1246 1 C04  ******
1247 1 C05  EXTERNAL REFERENCES
1248 1 C05  ROUTINES USED -
1249 1 C05  EXEC
1250 1 C05  JAO
1251 1 C05  KEVT
1252 1 C05  MAPI
1253 1 C05  XDDBA
1254 1 C05  XNBD
1255 1 C05  XREP
1256 1 C05  XRMOV
1257 1 C05  XRMSG
1258 1 C05  XRGS
1259 1 C05  XRSCR
1260 1 C05  ******
1280 1 BEGIN XDREN
1281 2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
1282 3 SAVE POINTER TO CURRENT NAME
1283 4 INCREMENT TO NEXT TOKEN
1284 5 IF TOKEN IS A HYPHEN, THEN
1285 6 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
1286 7 DECODE CLASS NAME
1287 8 EXIT TO :CLASER: IF CLASS SPECIFIED IS NOT VALID (S, I, D, F, B)
1288 9 ELSE
1289 10 SET CLASS TO 60
1290 11 ENDIF
1291 12 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
1292 13 SAVE POINTER TO NEW NAME
1293 14 IF CLASS IS DATA BASE OR ORED, THEN
1294 15 EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS
1295 16 ENDIF
1296 17 IF CLASS IS DATA BASE, THEN
1297 18 EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
1298 19 ENDIF
1299 20 IF CLASS IS INTERFACE TABLE, THEN
1300 21 CALL EXEC TO WRITE/READ NEW NAME
1301 22 BUILD AWA MANAGER REQUEST TO CHANGE NAME IN TOC
1302 23 BUILD AWA MANAGER REQUEST TO STORE NEW NAME IN TABLE
1303 24 CALL XRER TO ISSUE REQUESTS
1304 25 EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR
1305 26 ELSE
1306 27 IF CLASS IS DATA BASE, THEN
1307 28 BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY
1308 29 EXIT TO :NAMERR: IF RETURN INDICATES ERROR
1309 30 EXIT TO :INVALID: IF DATA BASE IS NOT A DB
1310 31 ENDIF
1311 32 BUILD AND ISSUE AWA MANAGER REQUEST FOR NAME CHANGE
1312 33 EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR
1313 34 IF CLASS IS DATA BASE OR ORED, THEN
1314 35 CALL XRPNF TO FORMAT FILE NAME
1315 36 CALL FILE MANAGER TO CHANGE DISC FILE NAME
1316 37 EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR
1317 38 IF FILE IS A PDB, THEN
1318 39 CALL XDDDP TO DELETE OLD PDB FROM XPD
1319 40 EXIT TO :NAMGR: IF ERROR RETURNED
1320 41 CALL XDDPA TO ADD NEW PDB NAME TO XPD
1321 42 EXIT TO :TELUSER: IF ERROR RETURNED
1322 43 ENDIF
1323 44 ENDIF
1324 45 ENDIF
1325 46 EXIT XDREN
1326 7 :SYNTAX: CALL XRMSG TO DISPLAY SYNTAX ERROR AND EXIT
1327 7 :TOOLNG: CALL XRMSG ("NEW NAME IS TOO LONG") AND EXIT
1328 7 :CLASER: CALL XRMSG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT
1329 7 :INVALID: CALL XRMSG ("AN MDB CANNOT BE RENAMED") AND EXIT
1333 2 :NAMAGM:
1334 2 CALL FILE MANAGER TO CHANGE NAME BACK
1335 2 :UNDO:
1336 2 BUILD AND ISSUE AN AMA MANAGER REQUEST TO CHANGE NAME BACK
1337 2 CALL XRMSG ("FILE MANAGER ERROR #" ; RENAME UNSUCCESSFUL") AND EXIT
1338 2 :TELUER:
1339 2 CALL XRMSG ("PDB NOT LOGGED IN XPDD ; SYSTEM ERROR # ...") AND EXIT
1340 1 END XDREM
**FORTRAN CALLING PROCEDURE**

CALL XDREQ

**INPUT**

COMMON XE - LU, RECPTR, REBUF
COMMON XB - DEBUG, ERFLG
COMMON XS - DRRDCB

**OUTPUT**

COMMON XB - ABFLG

**EXTERNAL REFERENCES**

EXEC
IAND
PURGE
XREQ
XRET
XRIG
XRMOV
XRMSG
BEGIN XDREQ
2 CALL XDREQ TO PROCESS AWS REQUEST(S)
3 IF AN ERROR RETURNED BY AWS MANAGER, THEN
4 IF ERROR FLAG (ERFLG) IS ZERO, THEN
5 TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED
6 CALL XDMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
7 ENDIF
8 CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
9 IF CLASS OF ELEMENT IS DDE, THEN
10 CALL PURGE TO DELETE THE FILE
11 ELSE, ELEMENT RESIDES IN AWS
12 CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
13 ENDIF
14 IF AWS REQUESTS EXIST IN REQBUF BEYOND FAILING REQUEST, THEN
15 MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
16 ENDIF
17 ELSE
18 SET REQPTR TO 1 INDICATING NO REQUESTS PRESENT
19 ENDIF
20 EXIT XDREQ
21 END XDREQ
FDRES

1408 1 C00
1409 1 C00
1410 1 C00
1411 1 C00
1412 1 C00
1413 1 C00
1414 1 C00
1415 1 C00
1416 1 C00
1417 1 C00
1418 1 C00
1419 1 C00
1420 1 C00
1421 1 C00
1422 1 C00
1423 1 C00
1424 1 C00
1425 1 C00
1426 1 C00
1427 1 C00
1428 1 C00
1429 1 C00
1430 1 C00
1431 1 C00
1432 1 C00
1433 1 C00
1434 1 C00
1435 1 C00
1436 1 C00
1437 1 C00
1438 1 C00
1439 1 C00
1440 1 C00
1441 1 C00
1442 1 C00
1443 1 C00
1444 1 C00
1445 1 C00
1446 1 C00
1447 1 C00
1448 1 C00
1449 1 C00
1450 1 C00
1451 1 C00
1452 1 C00
1453 1 C05
1454 1 C05
1455 1 C05
1456 1 C05
1457 1 C05
1458 1 C05
1459 1 C05
1460 1 C05
1461 1 C05
1462 1 C05
1463 1 C05
1464 1 C05
1465 1 C05
1466 1 C05

**FORTRAN CALLING PROCEDURE**

**CALL FDRES (DATBUF)**

FDRES PROCESSES THE RESTORE DIRECTIVE. THE SPECIFIED UTDB OR MDB FILE IS OPENED AND ITS TOC IS READ. JDILS IS CALLED TO MARK TOC ENTRIES FOR RESTORE. JDREDF IS CALLED TO RESTORE DATA ELEMENTS AND TABLES. XRDFIS CALL TO RESTORE DREDF.

**INPUT**

- COMMON XE - LU, FLAGS, REPR, REBUF, TOKENS,
- CARTGR, COMPTR, CONBUF

**CALLING SEQUENCE**

**DATBUF - 1408 WORD BUFFER USED TO READ UTDB FILE**

**OUTPUT**

- COMMON XE - REBUF

**COMMON XB - ABFLG = ABORT FLAG, NON-ZERO VALUE FROM XRDEF,
- XRDF, JDILS OR JDREDF INDICATES ABORT OF RESTORE**

- ALLFLG = SET NON-ZERO IF LIST OF ELEMENTS SPECIFIED
- DEBUG = DEBUG BIT OF 'FLAGS' IN XE COMMON
- ENDBLK = BLOCK # OF LAST BLOCK READ
- ERFLG = SET NON-ZERO IF MESSAGE #5A ISSUED SO THAT IT IS ISSUED ONLY ONCE
- FILNAM = UTDB/MDB FILE NAME
- TOCBUF = UTDB TOC ENTRIES, 8 WORDS EACH, MAX 1200 WORDS
- TOCENT = ATA TOC ENTRY FOR THE UTDB
- TONXID = INDEX TO NEXT UTDB TOC ENTRY
- TOTSIZ = NO. OF BLOCKS REMAINING IN UTDB FILE

**NOTES**

**Routines Called**

- CLOSE
- EXEC
- OPEN
- READ
- XDBL
- XRDF
- XRDF
1476 1 BEGIN XDRES
1477 2 SET AFLGS TO ZERO
1478 3 ERREXIT IF NEXT TOKEN IS NOT A COMM :ERR09:
1479 4 ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERR09:
1480 5 RETAIN THIS NAME AS DATA BASE TO BE RESTORED
1481 6 INCREMENT TO NEXT TOKEN
1482 7 IF TOKEN IS NOT A COMM, THEN
1483 8 ERREXIT IF TOKEN IS NOT EOS :ERR04:
1484 9 ENDIF
1485 10 BUILD DATA REQUEST FOR TOC ENTRY RETRIEVE
1486 11 CALL XREX TO PROCESS DATA REQUEST
1487 12 ERREXIT IF DATA REQUEST FAILED :ERR10:
1488 13 ERREXIT IF DATA BASE FOUND IS A PDB :ERR33:
1489 14 IF DATA BASE IS A UDB, THEN
1490 15 CALL XDFM TO CONSTRUCT QUALIFIED FILE NAME
1491 16 ENDIF
1492 17 CALL OPEN TO OPEN SPECIFIED FILE
1493 18 ERREXIT IF OPEN FAILED :ERROR:
1494 19 CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1495 20 ERREXIT IF READ FAILED :ERROR:
1496 21 INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1497 22 IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1498 23 CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1499 24 ERREXIT IF READ FAILED :ERROR:
1500 2 ENDIF
1501 2 UPDATE TOTSIZ TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 2 CLEAR ERROR MESSAGE FLAG (ERFLG)
1503 2 IF TOKEN IS EOS (I.E., NO LIST OF ELEMENTS), THEN
1504 2 SET ALLFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 ELSE
1506 2 SET ALLFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 2 CALL XDLIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 2 EXIT XDRES IF ALLFLG SET BY XDLIS
1509 2 ENDIF
1510 2 CALL XDREX TO RESTORE AWS RESIDENT ELEMENTS
1511 2 EXIT XDRES IF AFLGS SET BY XDREX
1512 2 CALL ZDREX TO RESTORE OREX'S
1513 2 EXIT XDRES IF AFLGS SET BY ZDREX
1514 2 CALL CLOSE TO CLOSE DATA BASE FILE
1515 2 ERREXIT IF CLOSE FAILED :ERROR:
1516 2 DO WHILE AWS REQUESTS REMAIN IN RENBUF
1517 2 CALL XDREX TO PROCESS AWS REQUESTS
1518 2 EXIT XDRES IF AFLGS SET BY XDREX
1519 2 ENDOD
1520 1 EXIT XDRES
1521 2 :ERR04: CALL XRMG - "SYNTAX ERROR. EXTRMEOD DATA"
1522 2 :ERR06: CALL XRMG - "FILE MANAGER ERROR ......"
1523 2 :ERR09: CALL XRMG - "SYNTAX ERROR. MISSING OR ILLEGAL FIELD"
1524 2 :ERR10: CALL XRMG - "...... NOT FOUND"
1525 2 :ERR33: CALL XRMG - "CANNOT RESTORE A PDB"
1526 2 DO UNTIL ALL AWS REQUESTS IN RENBUF HAVE BEEN PROCESSED
1527 3 IF REQUEST IS TO STORE DATA, THEN
1528 4 CALL EXEC TO FREE THE SPECIFIED CLASS NO. AND SAM BUFFER
1529 2 ENDIF
**FORTRAN CALLING PROCEDURE**

**INPUT**

1 CD0 COMMON XE - CARTAG, COMBUF, COMPTR, FLAGS, LU, TOKENS

1 CD2 FILES - *XXXQ (UTDB FILE SPECIFIED)

**OUTPUT**

1 CD3 COMMON XE - REGBUF, RERPTR

1 CD3 FILES - *XXXQ (PDB FILE SPECIFIED)

**INTERNAL VARIABLES**

1 CD4 DCBPDB - DCB FOR THE PDB FILE; ALLOCATED IN X5 COMMON

1 CD4 DEBUTO - DCB FOR THE UTDB FILE; ALLOCATED IN X8 COMMON;

1 CD4 CONTAINS 1152 WORD BUFFER USED TO READ THE

1 CD4 UTDB AND TO WRITE THE PDB.
BEGIN YDSAV
SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
EXIT IF "*" IS NOT NEXT TOKEN :ERROR:
INCREMENT TO NEXT TOKEN
ERREX IF TOKEN IS NOT 'NAME' :ERROR:
ERREX IF THIS NAME IS MORE THAN 4 CHARACTERS :ERR06:
ERREX IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERR06:
RENAME THIS NAME AS UTDB
INCREMENT TO NEXT TOKEN
ERREX IF "*" IS NOT NEXT TOKEN :ERROR:
INCREMENT TO NEXT TOKEN
ERREX IF TOKEN IS NOT 'NAME' :ERROR:
ERREX IF THIS NAME IS MORE THAN 4 CHARACTERS :ERR16:
ERREX IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERR16:
RENAME THIS NAME AS PDB
INCREMENT TO NEXT TOKEN
ERREX IF TOKEN IS NOT 'E05' :ERR04:
BUILD AWA REQUEST TO RETRIEVE UTDB'S TOC ENTRY
CALL XENH TO PROCESS AWA REQUESTS
ERREX IF TOC RETRIEVE FAILED :ERR06:
ERREX IF TOC ENTRY DOES NOT INDICATE UTDB :ERR06:
CALL XODDA TO ADD PDB TO XOD
ERREX IF PDB LIMIT EXCEEDED :ERR20:
ERREX IF OPEN FAILED :ERR18:
SET STATUS FLAG INDICATING PDB LOGGED
CALL XRNEN TO BUILD UTDB FILE NAME
CALL OPEN TO OPEN UTDB FILE (SPECIFYING TYPE 1)
ERREX IF OPEN FAILED :ERR18:
SET STATUS FLAG INDICATING UTDB FILE OPEN
BUILD AWA REQUEST TO ALLOCATE PDB -- USE SIZE OF UT13
CALL XENH TO PROCESS AWA REQUEST
ERREX IF PDB IS DUPLICATE :ERR17:
ERREX IF AWA OVERFLOW :ERR21:
SET STATUS FLAG INDICATING PDB LOGGED IN AWA
CALL XRNEN TO BUILD PDB FILE NAME
CALL CREATE TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE
OF UTDB FILE FROM TOC ENTRY
ERREX IF CREATE FAILED :ERR18:
SET STATUS FLAG INDICATING PDB FILE OPEN
DO FOR ALL DATA IN UTDB FILE
   CALL ...ADF TO READ 1 BUFFER OF UTDB DATA
ERREX IF READ FAILED :ERR16:
CALL WRITTY TO WRITE 1 BUFFER TO PDB FILE
ERREX IF WRITE FAILED :ERROR:
ENDDO
CLOSE UTDB FILE
CLOSE PDB FILE
EXIT YDSAV
:ERROR: ISSUE MESSAGE "SYNTAX ERROR -- ILLEGAL OR MISSING FIELD"
:ERROR: ISSUE MESSAGE "INVALID UTDB FILE NAME ...."
:ERROR: ISSUE MESSAGE "INVALID PDB FILE NAME ...."
:ERROR: ISSUE MESSAGE "SYNTAX ERROR. EXTRANEOUS DATA"
:ERROR: ISSUE MESSAGE "PDB ..... ALREADY EXISTS"
1625 2  :ERR18: ISSUE MESSAGE "PRO FILE ACCESS ERROR ... "
1626 2  :ERR20: ISSUE MESSAGE "FILE MANAGER ERROR ... "
1627 2  :ERR20: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PRO's
1628 2  ALREADY REACHED"
1629 2  :ERR21: ISSUE MESSAGE "ANA OVERFLOW. NOT LOGGED"
1630 2  :RETURN:
1631 2  IF STATUS FLAG INDICATES PRO FILE IS OPEN, THEN
1632 2      PURGE PRO FILE
1633 2  ENDIF
1634 2  IF FLAG INDICATES PRO IS IN ANA, THEN
1635 2      CALL XODD TO DELETE PRO FROM ANA
1636 2  ENDIF
1637 2  IF FLAG INDICATES UTOD IS OPEN, THEN
1638 2      CLOSE UTOD
1639 2  ENDIF
1640 2  IF FLAG INDICATES PRO IS IN XPRD, THEN
1641 2      CALL XODD TO DELETE PRO FROM XPRD
1642 2  ENDIF
1643 1  END XDSAV
1737 1 BEGIN XSTO,
1738 2 IF ABFLG is ZERO (ABORT FLAG)
1739 3 ERREXIT IF UTDB NAME IS NOT VALID TO :ERR2:
1740 4 BUILD REQUEST FOR AMA TOC
1741 5 CALL XReq TO MAKE MANAGER REQUEST
1742 6 CALL EXEC TO SET AMA TOC
1743 7 ERREXIT IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1744 8 DO FOR ALL DATA BASE CLASS ENTRIES
1745 9 ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1746 10 ENDDO
1747 11 DO FOR IT, SFT, DE, ORDE AWA TOC ENTRIES
1748 12 IF WHOLE AWA IS TO BE STORED THEN
1749 13 DO FOR EACH ENTRY IN THIS CHAIN
1750 14 IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1751 15 IF STORE/RESTORE BIT IS ON THEN
1752 16 INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
1753 17 ENDF
1754 18 ENDDO
1755 19 ELSE
1756 20 ERREXIT IF ABLFLG IS NOT ZERO TO :ERR5:
1757 21 SET NOTOC = 0 (NUMBER OF UTDB TOC ENTRIES)
1758 22 DO FOR IT, SFT, DE, ORDE CHAINS
1759 23 IF STORE/RESTORE BIT IS ON THEN
1760 24 INCREMENT TOTAL BY 1
1761 25 ENDF
1762 26 ENDDO
1763 27 ERREXIT IF THERE ARE NO UTDB TOC ENTRIES (NOTOC=0) TO :ERR2:
1764 28 COMPUTE DATA AS FIRST RECORD AVAILABLE FOR DATA
1765 29 CALL XP ' TO CREATE FILE NAME
1766 30 CALL TO CREATE UTDB FOR TOTAL
1767 31 ERREXIT IF ERROR IN CREATE TO :ERR3:
1768 32 CALL XWRITE TO WRITE UTDB FILE
1769 33 ERREXIT IF ABFLG IS 4 (ORDE LARGER THAN SPECIFIED) TO :ERR1:
1770 34 ERREXIT IF ABFLG IS 3 (ORDE FILE ERROR) TO :ERR4:
1771 35 ERREXIT IF ABFLG IS 2 (UTDB FILE ERROR) TO :ERR3:
1772 36 ERREXIT IF ERROR IN WRITE TO :ERR3:
1773 37 CALL CLOSE TO CLOSE UTDB FILE
1774 38 ERREXIT IF ERROR IN CLOSE TO :ERR3:
1775 39 BUILD REQUEST TO ALLOCATE UTDB IN AWA
1776 40 CALL XReq TO MAKE REQUEST
1777 41 ERREXIT IF AWA OVERFLOW TO :ERR1:
1778 42 EXIT XSTO
1779 43 :ERR1:
1780 44 CALL XMSG TO DISPLAY MSGNO
1781 45 GOTO TO :ERR4:
1782 46 :ERR2:
1794 2 CALL XRMSTG TO DISPLAY MSGNO
1795 2 GO TO :ERR5:
1796 2 :ERR3:
1797 2 CALL XRMSTG TO DISPLAY MSGNO WITH UTDB FILE ERROR CODE
1798 2 :ERR4:
1799 2 CALL CLOSE TO CLOSE UTDB
1800 2 CALL PURGE TO PURGE UTDB
1801 2 :ERR5:
1802 2 CALL XRMSTG TO DISPLAY STORE ABORTED/ UTDB NOT CREATED MESSAGE
1803 1 END XDSTO
FORTRAN CALLING PROCEDURE

CALL XDOTC

**NOTE**

FDS AWA/DWA TABLE OF CONTENTS DIRECTIVE HANDLER. XDOTC INTER- 
PRETS THE TOC DIRECTIVE, RETRIEVES THE INDICATED TOC (AWA OR 
DATA BASE FILE) AND FORMATS AND OUTPUTS THE REQUESTED ENTRIES.

**LOCAL VARIABLES**

- ASZ - TOTAL ALLOCATABLE SIZE OF AWA (SEE MODULE XMWA)
- CLS - CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE
- CODE - CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'
- END OF TOC CHAIN DESIGNATOR (-32768)
- FRE - CURRENT AMOUNT OF FREE SPACE IN AWA (SEE MODULE XMWA)
- NS - ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)
- IDIM - TOC ENTRY DIM FIELD (WORD 8)
- LIME - BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE
- MULT - ARRAY OF MULTIPLIERS USED TO DETERMINE NUMBER OF WORDS 
  IN COLUMNS OF TWO DIMENSIONAL DATA ELEMENTS (FUNCTION OF 
  DATA TYPE)
- OPTION - ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY
- ORG - ORIGIN ADDRESS OF AWA (SEE MODULE XMWA)
- POS - VALUE OF 2 OR 22 INDICATING FIRST OR SECOND POSITION IN 
  TOC DISPLAY LINE
- SIZE - TOC ENTRY SIZE FIELD (WORD 7)
- TCMAX - MAXIMUM SIZE OF TOC WHICH CAN BE TOTALLY ACCOMMODATED BY 
  INTERNAL BUFFER
- TYPE - TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)
- TYPEID - ARRAY OF TYPE ID CODES FOR DE AND ORDE DISPLAYS
- UNIT - LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY

**NOTES**

- USES CLOSE, EXEC, IAMO, IXOR, KCVT, OPEN, READF, XDSTA, XREG, 
  XRG, XRMOV, XRG3, XR5FA, XR5BF, XR5BG
- IN THE EVENT THE COMPACTED AWA TOC AND HEADER DATA WILL NOT FIT 
  IN THE ALLOCATED BUFFER SPACE, AS MUCH OF EACH CHAIN AS POSSIBLE 
  WILL BE DISPLAYED UNTIL THE LINK FIELDS LEAD BEYOND THE BUFFER.
- IT IS ASSUMED THAT XDOTC AND XDSTO USE THE SAME SIZE BUFFER FOR 
  TOC MANIPULATION; THUS, A DATA BASE FILE MAY NOT HAVE A TOC TOO 
  LARGE FOR THE XDOTC INTERNAL BUFFER.
1866 1 BEG XDTOC
1867 2 SL) FOR OUTPUT OF ALL USER CLASSES FROM AWA TO TERMINAL
1868 2 IF NEXT TOKEN IS NOT EOS
1869 3 THEN
1870 4 IF TOKEN IS A HYPHEN
1871 4 THEN
1872 4 INCREMENT TO NEXT TOKEN
1873 4 IF TOKEN IS NOT A CHARACTER T
1874 4 THEN
1875 4 EXIT TO :ERROR3: IF TOKEN IS NOT A CHARACTER P
1876 4 SET OUTPUT UNIT FOR LINE PRINTER
1877 4 ENDF
1878 4 INCREMENT TO NEXT TOKEN
1879 3 ENDF
1880 3 IF TOKEN IS NOT EOS
1881 3 THEN
1882 3 EXIT TO :ERROR4: IF TOKEN IS NOT A COMMA
1883 3 INCREMENT TO NEXT TOKEN
1884 4 IF TOKEN IS NOT A COMMA
1885 4 THEN
1886 4 EXIT TO :ERROR5: IF TOKEN DOES NOT INDICATE A VALID CLASS (D, F, I, S OR B)
1887 5 SET OPTION FOR INDICATED CLASS
1888 5 INCREMENT TO NEXT TOKEN
1889 4 ENDF
1890 4 IF TOKEN IS NOT EOS
1891 4 THEN
1892 4 EXIT TO :ERROR4: IF TOKEN IS NOT A COMMA
1893 5 INCREMENT TO NEXT TOKEN
1894 4 EXIT TO :ERROR6: IF TOKEN IS NOT A FOUR CHARACTER NAME
1895 5 INCREMENT TO NEXT TOKEN
1896 4 EXIT TO :ERROR4: IF TOKEN IS NOT EOS
1897 5 CALL XRER TO RETRIEVE DATA BASE FILE TOC ENTRY
1898 5 EXIT TO :ERROR6: IF NOT LOGGED IN TOC
1899 5 GET TYPE AND SET NAME IN HEADER
1900 5 CONSTRUCT FILE NAME
1901 5 READ FIRST DATA BASE FILE TOC RECORD
1902 5 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AWA TOC
1903 5 READ SUBSEQUENT TOC RECORDS
1904 5 EXIT TO :ERROR7: IF FILE ACCESS FAILS
1905 5 DO FOR EACH NON-EMPTY TOC CHAIN
1906 6 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
1907 6 IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NUL CHAIN EXISTED
1908 7 THEN
1909 7 MARK PREVIOUS ENTRY AS AN END OF CHAIN
1910 7 ENDF
1911 7 EMDO
1912 5 DO FOR EACH ENTRY IN TOC
1913 6 IF NOT MARKED AS AN END OF CHAIN
1914 6 THEN
1915 6 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
1916 6 ENDF
1917 5 ENDO
1918 5 EXCLUDE CHAIN 8 (DATA BASE FILES) FROM DISPLAY
1919 4 ENDF
1920 3 ENDF
1921 2 IF Referencing AWA TOC
1922 2 THEN
1923 2 CALL XRER TO RETRIEVE AWA TOC
1924 3
IF DEBUG AND/OR TRACE FLAGS ARE SET
THEN
ENDIF
ENDIF
ENDIF
OUTPUT TOC HEADER
DO FOR EACH CLASS INDICATED BY OPTION
OUTPUT CLASS HEADER
LOCATE CLASS CHAIN HEAD
DO UNTIL END OF CHAIN FOUND (-32768)
INDEX TO NEXT TOC ENTRY
IF CHAIN "QINTS WITHIN LIMIT OF BUFFER
THEN
FORMAT NAME & SIZE FIELDS
IF DEBUG AND/OR TRACE FLAGS SET
THEN
FORMAT I-DIM FIELD IN OCTAL
ENDIF
FORMAT TYPE FIELD IN INTEGER
IF CLASS 2 OR 3
THEN
FORMAT TYPE FIELD USING DATA TYPE TABLE
IF CLASS 2
THEN
FORMAT I-DIM & J-DIM FIELDS
ENDIF
ELSE
IF CLASS 8
THEN
FORMAT TYPE FIELD USING FILE TYPE TABLE
ENDIF
ENDIF
ELSE
PRINT 'DATA LOST' MESSAGE
EXIT PROCESSING FOR THIS CHAIN
ENDIF
PRINT ENTRY
ENDDO
IF PROCESSING ANA TOC
THEN
CALL XSTA TO DISPLAY ANA USAGE STATISTICS
ENDIF
EXIT XDTOC
1976 1 C0----------
1977 1 C00
1978 1 C00
1979 1 C00
1980 1 C00
1981 1 C00
1982 1 C00
1983 1 C00
1984 1 C00
1985 1 C00
1986 1 C00
1987 1 C00
1988 1 C00
1989 1 C00
1990 1 C00
1991 1 C00
1992 1 C00
1993 1 C00
1994 1 C00
1995 1 C00
1996 1 C00
1997 1 C00
1998 1 C00
1999 1 C00
2000 1 C00
2001 1 C00
2002 1 C00
2003 1 C00
2004 1 C00
2005 1 C00
2006 1 C00
2007 1 C00
2008 1 C00
2009 1 C00
2010 1 C00
2011 1 C00
2012 1 C00
2013 1 C00
2014 1 C00----------
2015 1 C10
2016 1 C10
2017 1 C10
2018 1 C10
2019 1 C10
2020 1 C10
2021 1 C10
2022 1 C10
2023 1 C10
2024 1 C10
2025 1 C10
2026 1 C10
2027 1 C10
2028 1 C10
2029 1 C10
2030 1 C10
2031 1 C10----------
BEGIN XDRRT
SET WRDNO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
SET MORES = 0 (NUMBER OF AVA REQUESTS IN RENBUF)
DO FOR ALL UDBR TOC ENTRIES UNTIL CLASS IS DRDE
BUILD REQUEST FOR DATA FROM AVA
INCREMENT MORES BY 1
IF REQUEST BUFFER IS FULL (MORES=8) THEN
PERFORM REQUEST TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
ENDIF
ENDDO
IF THERE ARE REMAINING REQUESTS (MORES>0) THEN
SET NEXT REQUEST TO END OF REQUEST LIST
PERFORM REQUEST TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
IF THERE IS A PARTIAL DATA RECORD LEFT (WRDNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UDB FILE
ENDIF
DO FOR EACH DRDE UDBR TOC ENTRY
STORE DATREC IN UDBR TOC ENTRY
CALL DRDFM TO CREATE FILE NAME
IF DRDE FILE IS TYPE 3 THEN
CALL OPEN TO OPEN FILE AS CORRECT TYPE
ENDIF
DO UNTIL EOF IS READ
CALL READY TO READ 1 RECORD
IF READY ERROR TO :ERR1:
ERREXIT IF READY ERROR TO :ERR1:
STORE RECORD LENGTH AT FRONT AND REAR OF DATA
INCREMENT WRDNO BY LENGTH + 2
IF THERE IS ENOUGH DATA TO WRITE (WRDNO>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UDB FILE
ENDIF
ENDDO
IF THERE IS REMAINING DATA (WRDNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UDB FILE
ENDIF
SET DATREC TO NEXT AVAILABLE RECORD FOR DATA
ELSE
CALL OPEN TO OPEN FILE AS TYPE 1
ERREXIT IF OPEN ERROR TO :ERR1:
COMPUTE TOTAL SIZE OF FILE IN WORDS
DO UNTIL ALL DATA IS COPIED TO UDB (SIZE=0)
IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
SET LENGTH = SIZE
ENDIF
CALL READY TO READ LENGTH DATA
ERREXIT IF READY ERROR TO :ERR1:
CALL WRITF TO WRITE LENGTH DATA
ERREXIT IF WRITF ERROR TO :ERR3:
INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
ENDIF
ENDDO
CALL CLOSE TO CLOSE DRDE FILE
ERREXIT IF CLOSE ERROR TO :ERR1:
ENDIF
EXIT XDRRT
BEGIN PROCEDURE
CALL XREAD TO MAKE REQUEST
EXERRT IF THERE IS AN OVERFLOW TO :ERR4:
DO FOR NUMBER OF REQUESTS (MOREB) IN BUFFER
CALL EXEC TO SET DATA FROM SAM
SET DATREC AND WRODB INTO THIS UTDB TOC ENTRY
INCREMENT WRODB BY SIZE OF THIS ELEMENT
IF THERE IS ENOUGH DATA TO WRITE (WRODB>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
ENDDO
REINITIALIZE REQUEST BUFFER (MOREB=0)
END READ

BEGIN WRITE
COMPUTE LENGTH TO WRITE IN WORDS AND RECORDS
IF THERE IS NOT AT LEAST 1 RECORD TO WRITE THEN
SET LENGTH TO MINIMUM OF 1 RECORD
ENDIF
IF THIS IS A DRED TYPE 3 FILE THEN
EXERRT IF THERE ARE MORE BLOCKS THAN SPECIFIED TO :ERR2:
ENDIF
CALL WRITF TO WRITE LENGTH DATA TO UTDB FILE
EXERRT IF WRITF ERROR TO :ERR3:
DECREMENT WRODB BY LENGTH
INCREMENT DATREC BY LENGTH/128
IF THERE IS REMAINING DATA (WRODB>1) THEN
CALL XMOV TO MOVE REMAINING DATA UP IN BUFFER
ENDIF
END WRITE

:ERR1:
CALL XMSG WITH DRED NAME AND RC
SET ABFLG TO SAY A DRED ERROR
GO TO :ERR3:

:ERR2:
CALL XMSG TO SAY DRED FILE LARGER THAN SPECIFIED IN TOC
SET ABFLG TO SAY DRED FILE ERROR

:ERR3:
CALL CLOSE TO CLOSE DRED FILE
SET ABFLG TO SAY UTDB FILE ERROR
EXIT XWRITE

:ERRA:
SET ABFLG TO SAY AWA OVERFLOW
DO FOR REMAINING REQUESTS
CALL EXEC TO FREE CLASS NUMBER
ENDIF
END XWRITE
FORTRAN CALLING PROCEDURE
CALL HELDS (THREE WORD ARRAY CONTAINING 'XECAL')

FD$ EXECUTIVE INITIALIZATION SEGMENT MAIN ROUTINE
INPUT
COMMON XE - SUBSTA

OUTPUT
COMMON XB - INITIALIZED ACCORDING TO THE VALUE OF SUBSTA
(SEE APPROPRIATE INITIALIZATION SUBROUTINE)
NOTE!
USES XEINO, XEINI, XEIMS, XEINS, XERIN

BEGIN XECAL
CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SREDIT:, :INTEDIT:) SUBSTA (-1 TO 3)
:GLOBAL: CALL XEINO TO INITIALIZE GLOBAL COMMON
:DIRECT: CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES
:EXECUT: CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL
:SREDIT: CALL XEINS TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING
:INTEDIT: CALL XEINS TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING
ENDCASE
END XECAL
30 1 CDO   FORTRAN CALLING PROCEDURE
40 1 CDO
41 1 CDO         CALL XEIND
42 1 CDO
45 1 C*******
44 1 CDO         INITIALIZE XB COMMON FOR DIRECTIVE LEVEL
46 1 CDO
47 1 C*******
48 1 CDO         INPUT
49 1 CDO         COMMON XE - CARTAG, FLAGS
50 1 CDO         XDFRM - FDS DIRECTIVE PROMPT FILE
51 1 CDO
52 1 C*******
53 1 CDO         OUTPUT
54 1 CDO         COMMON XB - DIRECT, NUMDIR
55 1 CDO
56 1 C*******
57 1 CDO         NOTES
58 1 CDO         USES CLOSE, EXEC, OPEN, PRM, READ, XREXT, XREG, XMSG, XVBN
59 1 CDO
60 1 C*******
61 1 *
62 1 *
63 1 *
64 1 *
65 1 BEGIN XEIND
66 2 READ FDS DIRECTIVE PROMPT FILE
67 2 STORE DIRECTIVES IN DYNAMIC COMMON
68 1 END XEIND
108 1 BEGIN XEINE
109 2 INITILIZE COMMON TO ZEROS
110 3 SET FILE (ART- RCE NUMBER
111 4 SET STATES TO DIRECTIVE LEVEL
112 5 SET TKMLNG
113 6 INITIALIZE TOKENS
114 7 READ LIBRARY DIRECTOR Y FIRST RECORD
115 1 EXIT TO :ERROR: IF READ FAILED
116 2 STORE NUMBER OF PROCESSORS IN NPROC
117 3 READ PROCESSOR NAME RECORD
118 4 EXIT TO :ERROR: IF READ FAILED
119 5 CALL XREX TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AMA
120 6 EXIT TO :ERROR: IF REQUEST FAILED
121 7 DO FOR EACH ID SEGMENT IN SYSTEM
122 8 DO FOR EACH PROCESSOR IN LIBRARY DIRECTORY
123 9 IF NAMES ARE THE SAME
124 10 THEN
125 11 NOTE EXISTANCE OF ID
126 12 ENDIF
127 13 ENDDO
128 14 IF ANY PROCESSORS NOT MARKED
129 15 THEN
130 16 LIST PROCESSORS MISSING ID SEGMENTS
131 17 EXIT XEINO WITH INITIALIZATION FAILED
132 18 ENDF
133 19 CALL OPEN TO OPEN PDB/MDB DIRECTORY ( XPOB)
134 20 EXIT TO :ERROR: IF OPEN FAILED
135 21 CALL READF TO READ 1ST RECORD OF XPOB (LIST OF MDB'S)
136 22 EXIT TO :ERROR: IF READ FAILED
137 23 PERFORM DCLOG TO LOG MDB'S IN AMA
138 24 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER
139 25 CALL READF TO READ THAT RECORD OF XPOB
140 26 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)
141 27 CALL CLOSE TO CLOSE XPOB
142 28 PERFORM DCLOG TO LOG MDB'S IN AMA
143 29 IF THERE ARE REQUESTS IN THE AMA REQUEST BUFFER, THEN
144 30 CALL XREX TO PROCESS THE AMA REQUESTS
145 31 ENDF
146 32 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
147 33 EXIT XEINE
148 1 END XEINE
149 2 BEGIN DCLOG
150 3 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
151 4 BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
152 5 IF 8 AMA REQUESTS HAVE BEEN BUILT, THEN
153 6 CALL XREX TO PROCESS AMA REQUESTS
154 7 ENDF
155 8 EXIT TO :ERROR: IF A REQUEST FAILED
156 9 ENDDO
157 2 END DCLOG
158 2 :ERROR: LIBRARY INITIALIZATION ERROR TERMINATION
159 1 END XEINE
FORTRAN Calling Procedure

CALL XEINI

************

166 1 C01  INITIALIZE XE AND XB COMMON FOR INTERFACE TABLE EDITING

169 1 C01  COMMON XE -

171 1 C02  MASTER = MASTER STATE

173 1 C02  TOKENS = LEXICAL TOKEN VALUES

174 1 C02  NOPROC = NUMBER OF ENTRIES IN LIBRARY DIRECTORY

175 1 C02  COMPTR = INDEX OF NEXT TOKEN IN COMBUF

176 1 C02  COMBUF = COMMUNICATIONS BUFFER

177 1 C02  PRNAME = PROCESSOR NAME FROM EXECUTION CONTROLLER

178 1 C02  XNAME = OLD INTERFACE TABLE TO BE EDITED

179 1 C02  XNAME = DEFAULT INTERFACE TABLE

180 1 C02  XNAME = PROMPT TABLE

181 1 C02  COMMON XB -

183 1 C03  WKBUFF = LENGTH OF WBUFF (CONSTANT)

184 1 C03  WKBUFF = WORKING BUFFER CONTAINING INTE. TABLE WITH

185 1 C03  SHORT PROMPTS

186 1 C03  DIRECT = SUPPORTED INTE DIRECTIVES

187 1 C03  NUMDIR = NO. OF ENTRIES IN INTE DIRECTIVES

188 1 C03  MENTAB = NEW INTE. TABLE NAME

189 1 C03  NUMARG = NO. OF ARGUMENTS

190 1 C03  WARG = NO. OF WORDS IN SPAC. AND HEADER OF WBUFF

191 1 C03  LITLEN = NO. OF WORDS IN LITERAL AREA

192 1 C03  LITPIX = INDEX TO START OF LITERAL AREA

193 1 C03  AISE - ARRAY MAPPING ARG. TYPE TO EFFECTIVE LENGTH

194 1 C03  ************

200 1 C05  NOTES -

201 1 C05  USES FILES -

202 1 C05  XNAME = DEFAULT INTERFACE TABLE FOR THIS PROCESSOR

203 1 C05  XNAME = PROMPT TABLE FOR THIS PROCESSOR

204 1 C05  ************

209 1 C05  USES ROUTINES

210 1 C05  XRMOV = EXEC

211 1 C05  XRMSG = CLOSE

212 1 C05  XREST = OPEN

213 1 C05  XREG = READF

214 1 C05  XRUPK = XRWSB

215 1 C05  XRCPK

216 1 C05  ************

218 1 C05  BEGIN XEINI

219 2 IF CALLED AS A RESULT INTE DIRECTIVE, THEN
POSITION TO 1ST TOKEN AFTER 'INTE'
EXP:EXIT IF TOKEN IS NOT ',' :ERR1:
INCREMENT TO NEXT TOKEN
EXP:EXIT IF TOKEN IS NOT A NAME :ERR2:
SET PRCNAM TO THIS NAME
INCREMENT TO NEXT TOKEN
SET INTNAME TO O
SET NEWTAB TO 'BINTAB'
IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN
EX:EXIT IF TOKEN IS NOT A COMMA :ERR2:
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT EOS, THEN
IF TOKEN IS A NAME, THEN
SET INTNAME TO THE NAME
INCREMENT TO NEXT TOKEN
ENDIF
IF TOKEN IS NOT EOS, THEN
EX:EXIT IF TOKEN IS NOT A COMMA :ERR2:
INCREMENT TO NEXT TOKEN
IF TOKEN IS A NAME, THEN
SET NEWTAB TO THE NAME
INCREMENT TO NEXT TOKEN
ENDIF
EX:EXIT IF TOKEN IS NOT EOS :ERR2:
ENDIF
ELSE
INTNAME AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLER
NEWTAB IS SET TO 'BINTAB'
ISSUE MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED
ENDIF
MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE
EX:EXIT IF REQUEST IS UNSUCCESSFUL :ERR1:
START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, D.C
EXIT IF PROCESSOR PRCNAM IS FOUND
ELSE
INCREMENT TO NEXT ENTRY
ENLOOP
EX:EXIT :ERR8:
ENDSEARCH
SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND
IF INTNAME = 0, THEN
CONSTRUCT THE NAME OF THE DEFAULT INTERFACE TABLE AS "PRCNAM"
OPEN THE DEFAULT INTERFACE TABLE FILE
EX:EXIT IF OPEN FAILED :ERR2:
READ THE DEFAULT INTERFACE TABLE
EX:EXIT IF THE READ FAILED :ERR6:
READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM
OF WBUFF
EX:EXIT IF READ FAILED :ERR14:
ELSE
MAKE MANAGER REQUEST FOR INTNAME INTERFACE TABLE
EX:EXIT IF ERROR OR COULD NOT FIND :ERR9:
ENDIF
EX:EXIT IF VERS NO. VERSION NO. OF TABLE TO BE EDITED :ERR10:
CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"
OPEN THE PROMPT TABLE
EX:EXIT IF OPEN FAILED :ERR15:
READ 2ND RECORD OF THIS PROMPT TABLE
ERREXIT IF READ FAILED :ERR17:
MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUF TO AREA FOLLOWING
THE PROMPT TABLE
THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
LITERAL DISPL. THAT MATCH EACH LITERAL DATA ENTRY
THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
EXIT XEINI

:ERR2: "SYNTAX ERROR ...
:ERR4: "INTNAM" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNAM DOES MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
1 END XEINI
1 CD0 FORTRAN CALLING PROCEDURE
305 1 CD0 CALL XEINS
307 1 CD0 CALL XEINS
309 1 CD0 C********
310 1 CD0 C********
311 1 CD1 INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING
312 1 CD1 C********
313 1 CD1 C********
314 1 CD2 INPUT
315 1 CD2 COMMON XE - CARG, COMBUF, COMPTR, LU, NOPROC, TOKENS
316 1 CD2 COMMON XE - CARG, COMBUF, COMPTR, LU, NOPROC, TOKENS
317 1 CD2 COMMON XE - CARG, COMBUF, COMPTR, LU, NOPROC, TOKENS
318 1 CD2 COMMON XE - CARG, COMBUF, COMPTR, LU, NOPROC, TOKENS
319 1 CD2 COMMON XE - CARG, COMBUF, COMPTR, LU, NOPROC, TOKENS
320 1 CD3 OUTPUT
321 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
322 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
323 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
324 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
325 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
326 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
327 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
328 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
329 1 CD3 COMMON XB - DEBUG, DIRECT, NEWTAB, NUNPRINT, NUMENT,
330 1 CD5 NOTES
331 1 CD5 USES FILES - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE
332 1 CD5 USES ROUTINES
333 1 CD5 EXEC
334 1 CD5 OPEN
335 1 CD5 READ
336 1 CD5 XEPR
337 1 CD5 XEPR
338 1 CD5 XEPR
339 1 CD5 XEPR
340 1 CD5 XEPR
341 1 CD5 XEPR
342 1 CD5 XEPR
343 1 CD5 XEPR
344 1 CD5 XEPR
345 1 CD5 XEPR
346 1 CD5 XEPR
347 1 C********
1 BEGIN XEMS
2 BUILD AWS REQUEST TO RETRIEVE "XLIBD"
3 ERREXIT IF TOKEN IS NOT COMMA : ERR02;
4 INCREMENT TO NEXT TOKEN
5 IF TOKEN IS NAME, THEN
6 BUILD AWS REQUEST TO RETRIEVE OLDTAB
7 RETAIN THIS NAME AS NEWTAB
8 INCREMENT TO NEXT TOKEN
9 ELSE
10 SET OLDTAB TO ZERO
11 EMDIF
12 IF OLDTAB NOT ZERO, THEN
13 ERREXIT IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED : ERR04;
14 SET PROMPT MODE AS UPDATE
15 SET NO. ENTRIES AS OLDTAB SIZE / 7
16 READ OLDTAB INTO WORKING BUFFER
17 ERREXIT IF FIRST REQUEST (RETRIEVE "XLIBD") FAILED : ERR01;
18 READ "XLIBD" INTO COMMON
19 ELSE
20 SET PROMPT MODE AS CREATE
21 SET NO. ENTRIES TO ZERO
22 EMDIF
23 OPEN, READ AND CLOSE FILE XSPRM
24 SET COUNT AND SERROR DIRECTIVES INTO XB COMMON
25 SET SUBSTATE FLAG TO SEQ. EDIT. (=2)
26 1 EXIT XEMS
27 2 : ERR01: CALL XMSG - 'INITIALIZATION ERROR ...'
28 2 : ERR02: CALL XMSG - 'SYNTAX ERROR'
29 2 : ERR04: DEFAULT MESSAGE TO '... NOT FOUND'
30 IF ERROR WAS NO AWS SPACE THEN
31 SET MSG TO '... NO AWS SPACE'
32 EMDIF
33 CALL XMSG TO DISPLAY MESSAGE
34 2 : ERR12: CALL XMSG - '... ALREADY EXISTS'
35 1 END XEMS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

COMM. XB - LITPTR, NUMARS, WKBNG, WKBUFF

NOTE: WKBUFF IS INPUT WITH THE INTERFACE

TABLE'S CHARACTERISTICS IN THE TOP

AND THE "PACKED" LITERAL AREA IN THE

BOTTOM.

COMM. XB - LITPTR, LITOWN, NARC, M'5UF

NOTE: WKBUFF IS OUTPUT WITH THE LITERAL AREAS

IN THEIR "UNPACKED" FORM.

USES Routines

XIEXT
XRMV
XRKKB
XRMSG
XRES

**
BEGIN XEINT
DO UNTIL ALL LITERAL ENTRIES HAVE BEEN PROCESSED
START SEARCH UNTIL ALL ARGUMENTS SEARCHED
EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
D-BIT IS ON
BUILD 3 WORD LITERAL ENTRY (3, I-SUB, J-SUB)
EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
D-BIT IS ON
IF ARGUMENT IS COMPLETE (D-BIT ON), THEN
CREATE BIT MASK WORD(S) IN NEW LITERAL ENTRY AREA
ELSE
MOVE BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA
ENDIF
DO UNTIL ALL BITS OF BIT MASK PROCESSED
Determine number of contiguous bits ON (or OFF)
AND MOVE CORRESPONDING NUMBER OF DATA WORDS
(COR O'S) INTO NEW LITERAL ENTRY AREA
ENDDO
ORELSE
INCREMENT TO NEXT ARGUMENT
ENDLOOP
EXIT IF NOT CALLED BY LIBRARY MAINTENANCE :ERR5:
SKIP TO NEXT LITERAL ENTRY
ENDSEARCH
ENDDO
EXIT TO :RETURN:
:ERR5: CALL XRM5G - 'SYSTEM INITIALIZATION ERR 5'
:RETURN:
END XEINT
FORTRAN CALLING PROCEDURE

CALL XEINX

INITIALIZE XE AND XB COMMON FOR EXECUTION CONTROLLER

COMMON XE - COMBUF, COMPTR, FLGS, LU, MASSTA, NOPROC, TOKENS

COMMON XB - LIBD, NOPAC2, SERROR, WKBUFF, WKBUFF

NOTES

USES EXEC, PRM, XREG, XREXT, XRI6, XRMG, XUEC, XVABN
1 BEGIN XEINX
2 IF INITIALIZATION FROM DIRECTIVE
3 THEN
4 IF DIRECTIVE IS NAME
5 THEN
6 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT EOS
7 ELSE
8 IF DIRECTIVE IS AUTO
9 THEN
10 IF TOKEN IS A WHPM
11 THEN
12 INCREMENT TO NEXT TOKEN
13 EXIT TO :ERROR3: IF TOKEN IS NOT THE NAME 'T'
14 CHANGE EXECUTION CONTROL MODE TO AUTO-T
15 INCREMENT TO NEXT TOKEN
16 ENDIF
17 EMDIF
18 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A COMMA
19 INCREMENT TO NEXT TOKEN
20 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A NAME
21 STORE NAME IN SEGNAME
22 CALL XREG TO RETRIEVE SEQUENCE TABLE
23 EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
24 INCREMENT TO NEXT TOKEN
25 SET SEONSTA TO FIRST SEQUENCE NUMBER
26 SET SERNDA TO LAST SEQUENCE NUMBER
27 IF TOKEN NOT EOS
28 THEN
29 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
30 INCREMENT TO NEXT TOKEN
31 IF TOKEN IS AN INTEGER
32 THEN
33 STORE STARTING RANGE NUMBER
34 SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
35 EXIT TO :ERROR3: IF NOT FOUND
36 INCREMENT TO NEXT TOKEN
37 ENDIF
38 IF TOKEN NOT E-,,
39 THEN
40 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
41 INCREMENT TO NEXT TOKEN
42 EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
43 ERREXIT TO :ERROR5: IF 'ENDING SEG' < 'BEGINNING SEG'
44 SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
45 EXIT TO :ERROR3: IF NOT FOUND
46 INCREMENT TO NEXT TOKEN
47 EXIT TO :ERROR2: IF TOKEN NO. :OS
48 EMDIF
49 ENDIF
50 SET SERNSTA TO SEONSTA
51 EMDIF
52 EMDIF
53 INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
54 EXIT TO :ERROR1: IF INITIALIZATION FAILS
55 EXIT XEINX
56 2 :ERROR1: INITIALIZATION FAILURE TERMINATION
57 2 :ERROR2: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
551 2 :ERR03: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552 2 :ERR04: SET SUBSTA TO DIRECTIVE LEVEL
553 2 IF ERROR WAS NOT AWA SPACE THEN
554 2 EXIT WITH NO AWA SPACE ERROR
555 2 ELSE
556 2 EXIT WITH SEQUENCE TABLE NOT FOUND ERROR
557 2 ENDIF
558 2 :ERR05: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
559 2 :ERR13: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH NUMBER NOT FOUND
560 1 END XEIXX
FORTRAN CALLING PROCEDURE:

CALL XELS (SEGNM) TO CALL SEGMENT FROM MAIN
CALL XRTH TO RETURN TO MAIN PROGRAM

YELDS ALLOWS A MAIN PROGRAM TO "CALL" A SEGMENT AND
THE ENTRY POINT XRTH PASSES CONTROL BACK TO THE MAIN PROGRAM

INPUTS IN CALLING SEQUENCE:

SEGNM - (INTEGER, 3 WORDS) ARRAY CONTAINING NAME OF THE
SEGMENT TO BE LOADED

SUBROUTINES AND FUNCTIONS CALLED:
EXEC

NOTES
1) XELS CONTAINS 2 ENTRY POINTS: XELS AND XRTH
2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
3) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XRTH"
FOLLOWED BY A CALL TO MAIN WHICH IS NOT EXECUTED
4) WHEN LOADER IS RUN, THE FOURTH PARAMETER MUST BE 1

BEGIN XELS

: XELS:
CALL .ENTR TO RETRIEVE PARAMETERS AND RETURN ADDRESS
MOVE THE ADDRESS OF THE SEGMENT NAME INTO EXEC PARAMETER LIST
CALL EXEC TO LOAD AND PASS CONTROL TO SEGMENT

: XRTH:
RETURN TO MAIN PROGRAM
END XELS
608  1 CD0       FORTRAN CALLING PROCEDURE
609  1 CD0
610  1 CD0       CALL XESCH
611  1 CD0
612  1 CD0
613  1 CD0
614  1 CD0       XESCH IS CALLED BY XEXEC AFTER COMPLETION OF ALL AUTOMATIC AND
615  1 CD0       SEMI-AUTOMATIC EXECUTIONS TO ASSURE THAT ALL DO AND SCAN PROCESSOR
616  1 CD0       CONTROL DATA IS PURGED FROM THE ANA AND ALL INCOMPLETE DATA BOX
617  1 CD0       FILES ARE PURGED FROM THE SYSTEM.
618  1 CD0
619  1 CD0
620  1 CD0       INPUT
621  1 CD0       XE COMMON - CARTAG
622  1 CD0       ANA - BOSTK, BSCHTB (SEE DO AND SCAN PROCESSORS
623  1 CD0
624  1 CD0
625  1 CD0       OUTPUT
626  1 CD0       XE COMMON - RETBUF, REPTR
627  1 CD0       XB COMMON - SCRATCH
628  1 CD0
629  1 CD0
630  1 CD5
631  1 CD5       EXTERNAL ROUTINES
632  1 CD5       CLOSE, EXEC, PURGE, XREQ, XRIG, XRMV, XRUNG
633  1 CD5
634  1 CD0
635  1 CD0
636  1 CD0
637  1 CD0
638  1 CD0       BEGIN XESCH
639  2 CD0       BUILD REQUESTS TO DELETE BOSTK AND RETRIEVE BSCHTB THEN DELETE IT
640  3 CD0
641  2 CD0       CALL XREQ TO ATTEMPT REQUESTS
642  2 CD0       ISSUE MESSAGE XERB FOR SUCCESSFUL DELETES
643  3 CD0
644  2 CD0       IF XREQ COMPLETED REQUESTS, I.E., BSCHTB EXISTED. THEN
645  4 CD0
646  4 CD0       CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN
647  4 CD0
648  4 CD0       CALL PURGE TO PURGE THE FILE
649  4 CD0
650  3 CD0
651  3 CD0
652  3 CD0
653  1 CD0
654  1 CD0       END XESCH
CD1  FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
CD1  ONE PROGRAM PER SIGNED-ON USER
CD1  ********
CD2  INPUT
CD2  SCHEDULING PARAMETERS - LU, CLASHQ, QUAL, FLAGS (SEE XE COMMON)
CD2  ********
CD3  OUTPUT
CD5  COMMON XE - COMBUF, COMPTR, MASSTA, SUBSTA, PLUS XEINE
CD5  INITIALIZATION
CD5  COMMON XE - INITIALIZATIONS FROM XEIND, XEINI, XEINS, XEINX
CD3  ********
CD5  NOTES
CD5  USES RHPAR, XDCLD, XDCLF, XELDS, XINTE, XRCPR, XRMIG, XSERE,
CD5  XTCOM, XTCMT
CD5  THE CALLS TO XELDS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
CD5  XEAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
CD5  THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CCRTROL OCCURS
CD5  BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XTCMT FROM
CD5  CALLING XSERE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
CD5  MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
CD5  WHEN MASSTA IS SET TO THE DIRECTIVE LEVEL.
CD5  ********
CD5  MUNIT
1 BEGIN XEXEC
2 RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASS, QVAL & FLAGS
3 CALL XEIMX TO INITIALIZE GLOBAL COMMON
4 DO FOREVER -- TERMINATES INSIDE HANDLER FOR IOFF
5 CALL STCOM FOR INPUT OF DIRECTIVE
6 IF ERROR OR NOT A VALID DIRECTIVE NAME
7 THEN
8 ISSUE MESSAGE ED6
9 ELSE
10 IF NAME IS INT
11 THEN
12 SET STATES TO INT LEVEL
13 CALL XEINI TO INITIALIZE DYNAMIC COMMON
14 EXIT TO :RESET; IF ERROR
15 CALL XINTE TO EDIT TABLE
16 ELSE
17 IF NAME IS SENE
18 THEN
19 SET STATES TO SENE LEVEL
20 CALL XEIMS TO INITIALIZE DYNAMIC COMMON
21 EXIT TO :RESET; IF ERROR
22 CALL XSENE TO EDIT TABLE
23 ELSE
24 IF NAME IS FOR SOME EXECUTION CONTROL OPTION
25 THEN
26 SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
27 DO UNTIL MASSTA IS AT DIRECTIVE LEVEL
28 CALL XEIXN TO INITIALIZE DYNAMIC COMMON
29 EXIT TO :RESET; IF ERROR
30 CALL XXCNT TO PERFORM EXECUTIONS
31 IF SUBSTA IS SET TO SENE LEVEL
32 THEN
33 CALL XEIMS TO REINITIALIZE DYNAMIC COMMON
34 EXIT TO :RESET; IF ERROR
35 CALL XSENE TO SUPPORT EXECUTION CONTROL
36 ENDIF
37 IF SUBSTA IS SET TO INT LEVEL
38 THEN
39 CALL XEINI TO REINITIALIZE DYNAMIC COMMON
40 EXIT TO :RESET; IF ERROR
41 CALL XINTE TO SUPPORT EXECUTION CONTROL
42 ENDIF
43 IF EXECUTION MODE WAS SEMI OR AUTO
44 THEN
45 CALL XESCN TO PURGE ANY RESIDUAL SCAM CONTROL DATA AND FILES
46 ENDIF
47 ELSE
48 CALL APPROPRIATE DIRECTIVE HANDLER VIA XCLX?
49 ENDIF
50 ENDIF
51 ENDIF
52 ENDIF
53 :RESET:
54 IF SUBSTA IS NOT DIRECTIVE LEVEL
55 THEN
56 CALL XEIND TO REINITIALIZE DYNAMIC COMMON
57 ENDIF
58 ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CDO FORTRAN CALLING PROCEDURE
2
3 1 CDO
4 1 CDO CALL XINTE
5 1 CDO
6 1 CDO
7 1 CDO
8 1 CDO
9 1 CDO
10 1 CDO OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR
11
12 1 CDO
13 1 CDO
14 1 CDO COMMON XE -
15 1 CDO
16 1 CDO COMMON XB -
17 1 CDO
18 1 CDO COMMON YB -
19 1 CDO DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE
20 1 CDO TABLE EDITOR ROUTINES
21
22 1 CDO
23 1 CDO NOTES
24 1 CDO
25 1 CDO USES ROUTINES XINIX
26 1 CDO COMMON XD -
27 1 CDO
28 1 CDO
29 1 CDO
30 1 CDO
31 1 *
32 1 * XINTE IS THE INTERFACE ROUTINE FOR THE INTERFACE TABLE EDITOR
33 1 * WHEN CALLED IN THE FDS EXECUTIVE'S OVERLAY STRUCTURE.
34 1 *
35 1 *
36 1 BEGIN XINTE
37 2 CALL XINIX TO EXECUTE INTERFACE TABLE EDITOR
38 2 CALL XERTN TO RETURN TO XEXEC IN MAIN SEGMENT
39 1 END XINTE
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
CALL XIMIX

INPUT

COMMON XI -
COMBUF = TERMINAL COMMUNICATIONS OUTPUT BUFFER
COMPR = INDEX TO NEXT TOKEN IN COMBUF
BACKSL = TOKEN FOR "/"
TOKENS = IDENTIFYING VALUES OF TOKENS IN COMBUF

COMMON XB -
LENLEN = LENGTH OF LITERAL AREA OF INTERFACE TABLE IN WBUF
NARG = INDEX TO START OF SHORT PROMPT AREA OF MKBUF
NEWTAB = NAME OF INTERFACE TABLE TO BE CREATED BY THIS EDIT. INPUT = 0 IF XINTER IS CALLED BY FDS LIBRARY MAINTENANCE PROGRAM.
NUMARG = NUMBER OF ARGUMENTS IN THIS INTERFACE TABLE
MKBUF = WORKING BUFFER CONTAINING INTERFACE TABLE AND SHORT PROMPTS WITH LITERAL IN EXPANDED FORM.

OUTPUT

COMMON XI -
MASSTA = MASTER STATUS SET TO 'Z' LEVEL IF ERROR OR IF 'Z' INPUT

COMMON XB -
PRMTND = CURRENT PROMPT MODE. SET TO 5 (MODIFY MODE) IF "7" INPUT
MKBUF = WORKING BUFFER CONTAINING INTERFACE TABLE
WITH LITERALS IN COMPARESSED FORM
AMA = XXXXX = NEW INTERFACE TABLE AND LITERAL AREA (NOT DONE IF CALLED BY LIBRARY MAINTENANCE)

NOTE

USES ROUTINES
EXEC
XREG
XILIT
XIPRM
XINPT
1 CD$ XROMV
2 CD$ XRMSE
3 CD$ XTCM
4
5 ***
6 1 =
7 1 * THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'INIT' DIRECTIVE
8 1 * OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
9 1 * PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
10 1 * TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
11 1 * OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
12 1 * NOTE : ALL INITIALIZATION, INCLUDING MKBUF (OLD INTERFACE TABLE),
13 1 * HAS BEEN PERFORMED BY XEINI.
14 1 *
15 1 BEGIN XINIX
16 2 IF NP (NO. OF PARAMETERS) > 0, THEN
17 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
18 3 * PRMTID = 1 => CREATE A MODE
19 3 * PRMTID = 3 => CREATE A MODE
20 3 * PRMTID = 4 => CREATE CONTINUE MODE
21 3 * PRMTID = 5 => MODIFY MODE
22 4 DO UNTIL 'EXIT' OR '^' IS ENTERED
23 4 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTID, SIZE, TYPE, AND STATUS
24 4 OF NEXT ARGUMENT
25 4 CALL XTCOM TO PROMPT USER AND RETURN PARSED INPUT
26 5 IF 'X' WAS NOT ENTERED, THEN
27 6 IF '\' WAS ENTERED, THEN
28 7 SET PRMTID TO 5
29 8 ELSE
30 9 IF NOTHING WAS ENTERED (I.E. TOKEN IS CD$), THEN
31 9 INCREMENT TO NEXT ARGUMENT
32 9 ELSE
33 9 CALL XINPT TO PROCESS THE USER'S INPUT
34 10 ENDF
35 11 ENDF
36 12 ENDDO
37 13 IF A 'X' WAS ENTERED, THEN
38 14 SET RETURN CODE INDICATING %
39 14 (I.E. MASSTA = 0)
40 14 ELSE
41 14 COMPRESS THE LITERAL LIST AREA
42 15 ENDF
43 16 ENDF
44 17 STORE INTERFACE TABLE AS NEWNAME
45 18 IF STORE INTO AWA FAILED, THEN
46 19 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
47 20 ELSE
48 21 SET GOOD RETURN CODE
49 1 END XINIX
FORTRAN CALLING PROCEDURE

CALL XIPRM

PROMPT CONSTRUCTOR FOR INTERFACE TABLE EDITOR

COMMON XD - ARGNO, CFLAG, DFLAG, IARG, IFLAG,
ISIZE, ITYPE, LEFF, LITSI, MARG,
MIXPRI, NUMARE, PRAMO, WKBUF

COMMON XD - ARGNO, ISUB, MODSAY, PRMLEN,
PRAMO, PROMPT

NOTES

USES ROUTINES

CRMD
XIEXT
XILSD
XRMOV
XRNB
XRPCK
XRUPK
1 * CONSTRUCT PROMPT TO BE ISSUED
2 BEGIN 
3 DO UNTIL A PROMPT IS CONSTRUCTED 
4 IF PRINTNO = 5, THEN 
5 CONSTRUCT A ":" PROMPT 
6 ELSE 
7 IF PRINTNO = 4 (CONTINUE MORE), OR 
8 PRINTNO = 6 (CONTINUE HERE MORE), THEN 
9 IF ARGNO IS A SCALAR, THEN 
10 CONSTRUCT PROMPT AS \"ARG=\" OR \"ARG\\n\" 
11 ELSE 
12 IF PRINTNO NOT = 6, THEN 
13 COMPUTE ISUB AS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LASTE) 
14 ELSE 
15 SET PRINTNO TO 4 
16 COMPUTE CURRENT ELEMENT NO. (SUBSCRIPT) FROM CURRENT INDEX 
17 ENDIF 
18 IF THERE ARE NO EMPTY SLOTS BEYOND LASTE, THEN 
19 SET PRINTNO TO MOOSAV (EXIT THE CONTINUE MORE) 
20 ELSE 
21 IF ARGNO IS DOUBLY SUBSCRIPTED PARAMETER, THEN 
22 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM) 
23 CONSTRUCT PROMPT AS \"ARG=(I,J)\" 
24 ELSE 
25 CONSTRUCT PROMPT AS \"ARG=(ISUB)\" 
26 ENDIF 
27 ELSE 
28 IF ARGNO = 8P, THEN 
29 SET PRINTNO TO 5 
30 ELSE 
31 INCREMENT ARGNO TO NEXT PARAMETER 
32 SET IFLAG TO 1, 2, OR 3 INDICATING 1, 0, OR 10 
33 SET LAST ENTERED INDICATOR (LASTE) TO 0 
34 IF PRINTNO = 3, THEN 
35 IF SOME DATA VALUE(S) OR PARM NAME EXISTS FOR ARGNO, THEN 
36 CALL XILSO TO LIST DATA FOR THIS ARGUMENT 
37 ENDIF 
38 CONSTRUCT PROMPT AS \"ARG=\" OR 0 OR =0 
39 ELSE 
40 IF ARGNO IS MARKED INCOMPLETE, THEN 
41 IF A PARTIAL LITERAL LIST EXISTS, OR 
42 THIS ARGUMENT IS A SCALAR, THEN 
43 COMPUTE ISUB AS FIRST EMPTY ELEMENT 
44 IF DOUBLY SUBSCRIPTED PARAMETER, THEN 
45 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM) 
46 CONSTRUCT PROMPT \"ARG=(I,J)\" 
47 ELSE 
48 CONSTRUCT PROMPT \"ARG=(ISUB)\" 
49 ENDIF 
50 ELSE 
51 CONSTRUCT PROMPT AS \"ARG=\" OR 0 OR =0 
52 ENDIF 
53 ENDIF 
54 ENDIF 
55 ENDIF 
56 ENDIF
FOR**IN CALLING PROCEDURE
CALL XIPMT

****
**
**
**
**
**
**
**
**
**
**

INPUT
COMMON XE - COMBUF, COMPTA, TOKENS

**

OUTPUT
COMMON XB - ARGNO, PRMTND

**

* XIPMT PROCEDURES THE PROMPT DIRECTIVE
BEGIN XIPMT
IF TOKEN IS NOT COMA :ERROR2:
POSITION TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT NAME :ERROR2:
ERREXIT IF TOKEN(S) FOLLOW THE NAME :ERROR2:
IF NAME IS 'N', THEN
SET PRMTND TO 1
ELSE
IF NAME IS 'A', THEN
SET PRMTND TO 3
ELSE
ERREXIT :ERROR2:
ENDIF
ENDIF
SET ARGNO TO 0
EXIT TO :RETURN:
:ERROR2: CALL XAMSG FOR 'INVALID SYNTAX'
:RETURN:
END XIPMT
FORTRAN CALLING PROCEDURE

CALL XILST

LIST DIRECTIVE PROCESSOR (WITHIN INTERFACE TABLE EDITOR)

INPUT

COMMON XE - COMPUT, COMPTR, TOKENS

COMMON XB - CFLAG, LITPTR, HRNG, HUMARG, VERSMN, WBUF

OUTPUT

COMMON XB - ARGNO

NOTE

USES ROUTINES

EXEC
XICMB
XICMR
XICMR
XILSB
XRCPR
XRLA
XRMDV
XRMSG
XRSIF
*XILST PROCESSES THE LIST DIRECTIVE

1 BEGIN XILST

1 IF TOKEN IS '-', THEN

1 POSITION TO NEXT TOKEN

1 ERREXIT IF TOKEN IS NOT NAME; ERR02:

1 ERREXIT IF NAME IS NOT 'C', 'V', OR 'A'; ERR02:

1 SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)

1 ELSE

1 SET MODEFG TO 2

1 ENDIF

1 IF TOKEN IS EOS, THEN

1 WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION

1 AND STATUS

1 DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED

1 IF MODEFG = 1 OR MODEFG = 3, THEN

1 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT

1 ENDIF

1 IF MODEFG = 2 OR MODEFG = 3, THEN

1 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT

1 ENDIF

1 ENDDO

1 ELSE

1 DO UNTIL EOS IS REACHED

1 ERREXIT IF TOKEN IS NOT COMMA; ERR02:

1 ERREXIT IF NEXT TOKEN IS NOT NAME; ERR02:

1 SET ARGNO TO 1

1 START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED

1 EXIT IF NAME = ARGNO'S NAME IN PROPPT TABLE

1 IF MODEFG = 1 OR MODEFG = 3, THEN

1 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT

1 ENDIF

1 IF MODEFG = 2 OR MODEFG = 3, THEN

1 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT

1 ENDIF

1 ENDOO

1 ENDO

1 PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER

1 ENDO

1 ENDO

1 EXIT TO :RETURN:

1 :ERR02: CALL XRMSG TO WRITE 'INVALID SYNTAX'

1 :RETURN:

1 END XILST
1 CD0        FORTRAN CALLING PROCEDURE
384 1 CD0        CALL XISUB
385 1 CD0
386 1 CD0
387 1 CD0
388 1 CD0
389 1 CD0
390 1 CD0        EFFECTIVE SUBSCRIPT CALCULATION ROUTINE
391 1 CD0
392 1 CD0
393 1 CD0        INPUT
394 1 CD0
395 1 CD0
396 1 CD0        COMMON XI - COMBUF, COMPR, TOKENS
397 1 CD0
398 1 CD0
399 1 CD0
400 1 CD0
401 1 CD0
402 1 CD0
403 1 CD0
404 1 CD0        COMMON X0 - IDIM, ISIZE, LENEFF
405 1 CD0
406 1 CD0
407 1 CD0
408 1 CD0
409 1 CD0
410 1 CD0        XRSIG
411 1 CD0
412 1 CD0
413 1 CD0        XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
414 1 CD0        THE INPUT SUBSCRIPT
415 1 CD0        BEGIN XISUB
416 2              INCREMENT TO NEXT TOKEN
417 2              ERREXIT IF TOKEM IS NOT INTEGER VALUE :ERR14:
418 3              IF IDIM FOR THIS ARGUMENT > 0, THEN
419 4              ERREXIT IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
420 5              INCREMENT TO NEXT TOKEN
421 6              ERREXIT IF TOKEN IS NOT COMMA :ERR15:
422 7              INCREMENT TO NEXT TOKEN
423 8              ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:
424 9              CALCULATE ISUB AS (I-1)*IDIM+1
425 2              ELSE
426 2              SET ISUB TO INTEGER VALUE
427 1              ENDIF
428 2              ERREXIT IF ISUB > XISUB :ERR16:
429 2              INCREMENT TO NEXT TOKEN
430 2              ERREXIT IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
431 2              EXIT TO RETURN:
432 2              :ERR14: CALL XRSIG - 'INVALID SUBSCRIPT SYNTAX'
433 2              :ERR15: CALL XRSIG - 'DOUBLE SUBSCRIPTED - MUST SPECIFY BOTH'
434 2              :ERR16: CALL XRSIG - 'INVALID SUBSCRIPT VALUE'
435 2              :RETURN:
436 1 END XISUB
FORTRAN CALLING PROCEDURE

CALL XIDAT

**LITERAL DATA PROCESSOR**

**INPUT**

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - CFLAG, COMPLE, DFLAG, IARG, IARG6,
             ISIZE, ISI65, YTYPE, LENEFF, LITDSP,
             LITOWN, LITSIZE, MDSAV, MDSAVM, MOBITM,
             PRMTND, SEFLG, WKBNG, WKBUF

**OUTPUT**

COMMON XB - IRETC, ISUB, LITDSP, LITOWN, MDSAVM,
            PRMTND, WKBUF

**INTERNAL VARIABLES**

COMMON XS -

IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS

LITERAL DATUM

NUMCL = NUMBER OF CONSECUTIVE ELEMENTS TO BE

MARKED COMPLETED AS A RESULT OF THIS

LITERAL DATUM

STACK = PUSH-DOWN LIST (MAX. OF 4 ENTRIES)

DESCRIBING NESTED REPEAT GROUPS.

EACH ENTRY IS 3 WORDS:

WORD 1 = INDEX TO 1ST TOKEN (IN COMBUF)

WORD 2 = REPEAT COUNT

WORD 3 = FLAG INDICATING WHETHER

REPEAT GROUP IS PARENTHETICALLY

GROUPED

STKPTR = INDEX TO NEXT ENTRY TO BE BUILT IN 'STACK'

**NOTES**

USES ROUTINES

XISUB

XRBIT

XRMOV

XRMSG
1 CD5 XMMR
2 CD5 XSER
3 CD5 XSET
4 C********
5 * XDAT PROCESSES THE INPUT LITERAL LIST
6 BEGIN XDAT
7 DO UNTIL EOS IS REACHED, OR
8 UNTIL AN ERROR OCCURS
9 IF THIS IS A DATA ELEMENT, THEN
10 (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
11 IF ARG TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
12 ERREXIT IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
13 SET LENGTH TO BE MOVED (LENMOV) TO EFFECTIVE LENGTH FOR DATA
14 OF ARG'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
15 IF ARG TYPE IS CHAR, THEN
16 SET LENGTH TO BE MOVED (LENMOV) TO EFFECTIVE LENGTH
17 FOR DATA INPUT
18 ELSE THIS MUST BE CHARACTER DATA BEING INPUT
19 ERREXIT IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ER10:
20 INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
21 COMPUTE NO. WORDS IN INPUT CHARACTER STRING
22 ERREXIT IF NO. WORDS (LENMOV) > EFFECTIVE LENGTH OF
23 THIS ARGUMENT'S DATA (LENEFF) :ER10:
24 ENDIF
25 ENDF
26 INCREMENT TOKEN POINTER TO THE DATA INPUT
27 VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
28 ARGUMENT FOR DATA INPUT (ISUB <= ISIZE - LENMOV + 1)
29 ERREXIT IF INSUFFICIENT SPACE :ERR11:
30 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
31 ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
32 ENDF
33 MOVE DATA FROM INPUT COMMUNICATIONS BUFFERS TO LITERAL AREA
34 SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
35 ARGUMENT, TO LENMOV
36 IF LENMOV < LENEFF ONLY POSSIBLE FOR CHARACTER DATA, THEN
37 MOVE LENMOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
38 ENDF
39 INCREMENT TO NEXT TOKEN
40 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENMOV
41 SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
42 ELSE TOKEN INDICATES NON-DATA ELEMENT
43 IF THIS IS A "(", THEN
44 CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
45 BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITYPE) AND
46 SECONDARY DIMENSION (IDIM)
47 EXIT XDAT IF ERROR (IRET < 0)
48 ELSE
49 IF THIS IS AN "[", THEN
50 CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
51 ELSE
52 MARK ONE ELEMENT OF THIS ARGUMENT AT ISUB AS INCOMPLETE
53 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
54 LENGTH OF ONE ELEMENT (LENEFF)
55 ENDF
56 TURFF OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
57 INCREMENT TO NEXT TOKEN
ELSE
  IF TOKEN IS A REPEAT COUNT, THEN
    IF NEXT TOKEN IS ",", THEN
      SET PARENFLAG TO 1
      INCREMENT TO NEXT TOKEN
    ELSE
      SET PARENFLAG TO 0
      ENDIF
    SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
  ELSE
    EXIT (INVALID FIELD) : ERROR:
  ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
DO UNTIL TOKEN IS NOT ")"
  IF PUSH-DOWN STACK IS NOT EMPTY, AND
  (PREVIOUS TOKEN WAS DATA, AND
  (PARENFLAG OF TOP OF STACK ENTRY IS 0), OR
  (CURRENT TOKEN IS ",", AND
  (PARENFLAG OF TOP OF STACK ENTRY IS 1), THEN
  GET REPEAT COUNT OF TOP OF STACK ENTRY
  DECREMENT REPEAT COUNT BY 1
  IF REPEAT COUNT > 0, THEN
    SET TOKEN INDEX TO INDEX ON PUSH-DOWN STACK
    REPLACE NEW REPEAT COUNT ON PUSH-DOWN STACK
  ELSE
    POP (I.E. REMOVE) ENTRY FROM TOP OF STACK
    IF TOKEN IS ",", THEN
      INCREMENT TO NEXT TOKEN
    ELSE
      EXIT LOOP
      ENDIF
ENDIF
ELSE
  IF PREVIOUS TOKEN WAS NOT A SUBSCRIPT, THEN
    IF THIS TOKEN IS NOT AN EOS, THEN
      EXIT IF TOKEN IS NOT A COMMA : ERROR:
    ENDIF
  ENDIF
ENDIF
ENDIF
ENDDO
IF PROMPT MODE NOT CONTINUE (=4), AND
  (THERE ARE EMPTY ELEMENTS BEYOND LASTE, OR
  PREVIOUS TOKEN WAS A COMMA), THEN
  RETAIN PROMPT MODE (SET MOSSAW TO PRMTWD)
  SET PROMPT MODE TO CONTINUE (=4)
ENDIF
IF ALL LITERAL SLOTS FILLED, THEN
  MARK ARGNO COMPLETE
  IF ALL ARGUMENTS ARE COMPLETE, THEN
    SET COMPLETE FLAG FOR INTERFACE TABLE
  ENDIF
ELSE
  TURN OFF COMPLETE FLAG FOR INTERFACE TABLE
ENDIF
FORTRAN CALLING PROCEDURE

CALL XNPT

INTERFACE TABLE EDITOR'S INPUT PROCESSOR

INPUT

COMMON XE - COMBUF, COMTR, TOKENS

COMMON XB - DIRECT, IODEF, SUBR, MARG, NDIR, NUMARG, PRMTMO, WKBUF

OUTPUT

COMMON XB - ARNGO, IRETC

USES ROUTINES

XIDAF
XIEFT
XIPS
XIMOF
XIRPR
XRMSG
1 * XINPT PROCESSES THE USER'S INPUT TEXT
2 BEGIN XINPT
3 IF FRMTD = 5, THEN
4 SET IOFLAG OFF (=0)
5 ERREXIT IF TOKEN IS NOT A NAME :ERROR2:
5 SAVE NAME AND POSITION TO NEXT TOKEN
6 IF TOKEN IS "=" , THEN
7 POSITION TO NEXT TOKEN
8 IF TOKEN IS '0', THEN
9 SET IOFLAG TO 10
10 POSITION TO NEXT TOKEN
11 ELSE
12 SET IOFLAG TO 1
13 ENDIF
14 ELSE
15 IF TOKEN IS '0', THEN
16 SET IOFLAG TO 0
17 ENDIF
18 ENDIF
19 IF IOFLAG NOT SET, THEN
20 CASE NAME (:EXIT:, :PROMPT:, :LIST:)
21 ERREXIT IF ANOTHER TOKEN FOLLOWS :ERROR2:
22 :EXIT: SET IRET TO SO THAT PROMPTING LOOP TERMINATES
23 :PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE
24 :LIST: CALL XILST TO PROCESS LIST DIRECTIVE
25 END CASE
26 ENDIF
27 START SEARCH UNTIL #F ENTRIES
28 EXIT IF NAME FOUND IN PROMPT TABLE
29 SET AENO TO ENTRY NO.
30 SET ISUB TO 1
31 ORELSE
32 INCREMENT TO NEXT PROMPT TABLE ENTRY
33 ENDCASE
34 ERREXIT :ERROR10:
35 ENDFIELD
36 ERREXIT IF IOFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:
37 ENDIF
38 IF NEXT TOKEN IS A NAME, THEN
39 CALL XIPAR TO PROCESS A PARAMETER FIELD
40 ELSE
41 ERREXIT IF IOFLAG IS NOT I ("=") :ERROR8:
42 CALL XIDAT TO PROCESS DATA LIST
43 ENDIF
44 EXIT XINPT
45 EXIT TO :RETURN:
46 :ERROR2: CALL XRMSG "INVALID SYNTAX"
47 :ERROR8: CALL XRMSG "MUST USE PARAMETER NAME TO RIGHT OF & OR =& "
48 :RETURN:
49 END XINPT
* XIPAR PROCFSSES A USER SPECIFIED PARAMETER FIELD

BEGIN XIPAR

IF A NAME IS SPECIFIED, THEN
  INCREMENT TO NEXT TOKEN
  IF TOKEN IS '(', THEN
    PROCESS I AND J SUBSCRIPTS
    EXIT IF INVALID SUBSCRIPTING :ERR14:
    IF DOUBLE SUBSCRIPTED, THEN
      SET S-FLAG IN ARGNO'S SPECS. FIELD
      SET LITOSP IN ARGNO'S SPECS. TO NEXT LITERAL AREA SPACE (LITDWN)
      PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT
      ELSE
        SET LITOSP IN ARGNO'S SPECS TO ISUB
        ENDIF
      ELSE
        SET LITOSP IN ARGNO'S SPECS TO 0
        ENDIF
    ERREXIT IF ORDER HAS MORE THAN 4 CHARACTERS :ERR18:
    ERREXIT IF EXTRAN'OUS FIELD INPUT :ERRO2:
   ENDIF
  ENDIF
  SET DELPE N AMOUNT OF LITERAL DATA IN ARGNO'S SPECS.
  SET COMPLETE AND S FLAG IN ARGUMENT'S CHARACTERISTICS
  IF ALL ARGUMENTS ARE COMPLETE, THEN
  SET INTERFACE TABLE COMPLETE FLAG
  ELSE
    EL'.' MUST BE A B INPUT
    ERREXIT IF NOT AN AMPERSAND (&) INPUT :ERRO2:
    CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS
    SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
    ENDIF
  ENDIF
  EXIT TO :RETURN;

:ERRO2: CALL XMSG - 'INVALID SYNTAX'

:ERR14: CALL XMSG - 'INVALID SUBSCRIPT SYNTAX'

:ERR18: CALL XMSG - 'INVALID ORDER NAME'

:RETURN:

END XIPAR
FORTRAN CALLING PROCEDURE

CALL XILSD

INPUT

COMMON XE - 

COMMON XB - I7, A, DFLAG, IARG, IDIN,

COMMON X - BUFPTR = INDEX INTO BUFFER FOR NEXT ASCII DATA

COMMON X - BUFFER = LINE TO BE OUTPUT

COMMON X - BUFFER = LINE TO BE OUTPUT

INTERNAL VARIABLES

EXTERNAL Routines

USES ROUTINES

REFERENCES
N MNNNNNNNNNNNt JJ'fNN^NNNNNNNMNV 6i vt

1. **XILSP** will list the data associated with one argument
2. A flag is returned as a prompt.
3. **BEGIN XILSP**
4. Set argument name into buffer.
5. Use flag to determine which of 'b', 'n', or 'n'.
6. Will go into the print buffer.
7. If b-flag is off indicating no literal data, then
8. If a parameter name is specified, then
9. Put parameter name into buffer.
10. If s-flag is on indicating two subscripts, then
11. Compute and convert to character format each subscript
12. If literal of argument is > 0, then
13. Compute and convert this subscript
15. ENSDF
16. Write out the print buffer built.
17. Endif.
18. Else.
19. Locate literal list and mask.
20. If symbolic string, then
21. Call files to print symbolic string.
22. Else.
23. Do until all elements processed.
24. Do until a buffer of data has been generated, or
25. Until all elements processed.
26. Compute and convert the subscript.
27. If mask for element indicates no data, then
28. Put "-" into buffer.
29. Else.
30. Convert the data using xro6, xre14, or xri6.
31. Put data and "-" into buffer.
32. If all elements of this argument have been processed, then
33. Remove the trailing comma in the print buffer.
34. Write out the print buffer built.
35. Endif.
36. End.
37. End.
38. End XILSP.
1  C00  FORTRAN CALLING PROCEDURE
2  C00  CALL XILSS
3  C00  C00  C00
4  C00  C00
5  C00  C00
6  C00  C00
7  C00  C00
8  C00  C00
9  C00  C00
10 C00  COMMON XE = LU
11 C00  COMMON XB = DEBUG, LISTLU, WKBUF
12 C00  COMMON XS = BUFFER = PRINT LINE BUFFER ALREADY INITIALIZED WITH
13 C00  NAME =
14 C00  BUFPTR = INDEX INTO BUFFER OF NEXT POSITIVE.
15 C00  DAPTR = INDEX INTO WKBUF OF SYMBOLIC STRING DATA
16 C00  C00
17 C00  C00
18 C00  COMMON XS - BUFFER, BUF PTR, DAT PTR
19 C00  C00
20 C00  C00
21
INTERNAL VARIABLES

CONTROL = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING
FOR EACH OF THE TOKEN VALUES 1-32. EACH CONTROL TABLE
ENTRY IS 3 WORDS:

WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER

WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER
OR FLAG DESCRIBING HOW TO COMPUTE

THEN

WORD 3 (TOKSZ1) = NO. OF WORDS IN SYMBOLIC STRING

ENTRY | TOKEN | WORD 1 | WORD 2 | WORD 3
-------|-------|--------|--------|--------
906     | CD4   |        |        |        |
907     | CD4   |        |        |        |
908     | CD4   |        |        |        |
909     | CD4   |        |        |        |
910     | CD4   |        |        |        |
911     | CD4   |        |        |        |
912     | CD4   |        |        |        |
913     | CD4   |        |        |        |
914     | CD4   |        |        |        |
915     | CD4   |        |        |        |
916     | CD4   |        |        |        |
917     | CD4   |        |        |        |
918     | CD4   |        |        |        |
919     | CD4   |        |        |        |
920     | CD4   |        |        |        |
921     | CD4   | INTEGER|        |        |
922     | CD4   | REAL   |        |        |
923     | CD4   | DOUBLE |        |        |
924     | CD4   |        |        |        |
925     | CD4   | NAME   |        |        |
926     | CD4   |        |        |        |
927     | CD4   |        |        |        |
928     | CD4   |        |        |        |
929     | CD4   | CHAR. STR. |        |        |
930     | CD4   |        |        |        |
931     | CD4   |        |        |        |
932     | CD4   |        |        |        |
933     | CD4   |        |        |        |
934     | CD4   |        |        |        |
935     | CD4   |        |        |        |
936     | CD4   |        |        |        |
937     | CD4   |        |        |        |
938     | CD4   |        |        |        |
939     | CD4   |        |        |        |
940     | CD4   |        |        |        |
941     | CD4   |        |        |        |
942     | CD4   |        |        |        |
943     | CD4   |        |        |        |
944     | CD4   |        |        |        |
945     | CD4   |        |        |        |
946     | CD4   |        |        |        |
947     | CD4   |        |        |        |
948     | CD4   |        |        |        |
949     | CD4   |        |        |        |
950     | CD4   |        |        |        |
951     | CD4   |        |        |        |
952     | CD4   |        |        |        |
953     | CD4   |        |        |        |
954     | CD4   |        |        |        |
955     | CD4   |        |        |        |
956     | CD4   |        |        |        |
957     | CD4   |        |        |        |
C****

1 CD5 NOTES
2 CD5 USES Routines

1 CD5 EXEC
2 CD5 XRE16
3 CD5 XRE14
4 CD5 XRE18
5 CD5 XXMSG
6 CD5 XXNOV

1 CD5

C****

1 BEGIN XILSS
2 MOVE A * CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
3 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
4 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
5 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
6 EXIT TO ERROR 2 IF FIELD = 0
7 IF SIZE < 0, THEN
8 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
9 ENDIF
10 IF TOKSIZ < 0, THEN
11 SET TOKSIZ TO SIZE + 2
12 ENDIF
13 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
14 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
15 CLEAR PRINT BUFFER TO BLANKS
16 ENDIF
17 IF FIELD > 0, THEN
18 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
19 ELSE
20 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
21
22 :ONE: CALL XRE16 WITH VALUE IN NEXT WORD OF SYMB. STRING
23 AND PUT RESULTS INTO PRINT BUFFER
24
25 :TWO: CALL XRE14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
26 AND PUT RESULTS INTO PRINT BUFFER
27
28 :THREE: CALL XRE18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
29 AND PUT RESULTS INTO PRINT BUFFER
30
31 :FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
32
33 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
34
35 :SIX: CALL XRE16 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
36 INTO PRINT BUFFER FOLLOWED BY AN "R"
37
38 :EXIT: PUT A * CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
39 INDEX BY 1
40 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
41 EXIT XILSS
42 ENDCASE
43
44 INCREMENT PRINT BUFFER INDEX BY SIZE
1015 1 CD0
1016 1 CD0  FORTRAN CALLING PROCEDURE
1017 1 CD0
1018 1 CD0  CALL XICHR
1019 1 CD0
1020 1 C********
1021 1 CD1
1022 1 CD1  PRINT THE CHARACTERISTICS OF AN ARGUMENT
1023 1 CD1
1024 1 C********
1025 1 CD2
1026 1 CD2  INPUT
1027 1 CD2
1028 1 CD2  COMMON XE - LU
1029 1 CD2
1030 1 CD2  COMMON XB - ARNO, IFLAG, ISIZE, ITYPE,
1031 1 CD2  LENEFF, NARG
1032 1 CD2
1033 1 C********
1034 1 CD5
1035 1 CD5  NOTES
1036 1 CD5
1037 1 CD5  USES Routines
1038 1 CD5
1039 1 CD5
1040 1 CD5  EXEC
1041 1 CD5
1042 1 CD5  XREP
1043 1 CD5
1044 1 CD5
1045 1 C********
1046 1 CD5  WRITE ARGUMENT CHARACTERISTICS
1047 1 CD5  BEGIN XICHR
1048 2  BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPT:, I/O TYPE AND
1049 2  DATA TYPE
1050 2  WRITE OUT THE PRINT BUFFER
1051 1 END XICHR
1053 1 CD0       FORTRAN CALLING PROCEDURE
1054 1 CD0
1055 1 CD0
1056 1 CD0     CALL XIEXT
1057 1 CD0
1058 1 CD0
1059 1 CD0
1060 1 CD0     EXTRACT VARIOUS FIELDS OF AN ARGUMENTS CHARACTERISTICS
1061 1 CD0     AND PUT VALUES INTO COMMON
1062 1 CD0
1063 1 CD0
1064 1 CD0
1065 1 CD0     ---
1066 1 CD0     ---
1067 1 CD0
1068 1 CD0     COMMON XV - ARGMNO, ISIZES, WBUF
1069 1 CD0
1070 1 CD0     ===
1071 1 CD0
1072 1 CD0
1073 1 CD0     ---
1074 1 CD0     ---
1075 1 CD0
1076 1 CD0
1077 1 CD0     COMMON XV - CFLAG, DFLAG, IARG, IARG4,
1078 1 CD0     ICLASS, IDIM, IFLAG, ISIZE,
1079 1 CD0     ISUB, ITYPE, LENEFF, LITEDP,
1080 1 CD0     LITISZ, NDIXTM, NDIXTN, SFLAG
1081 1 CD0
1082 1 CD0
1083 1 CD0
1084 1 CD0
1085 1 CD0
1086 1 CD0
1087 1 CD0
1088 1 CD0
1089 1 CD0     * EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS
1090 1 CD0     * ARGUMENT
1091 1 CD0
1092 1 CD0     BEGIN XIEXT
1093 2 USE XIEXT
1094 2 USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
1095 2 IN THE WORKING BUFFER
1096 2 EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
1097 1 END XIEXT
1097 1 CBO     FORTRAN CALLING PROCEDURE
1098 1 CBO     CALL XILIT
1099 1 CBO
1100 1 CBO
1101 1 CBO
1102 1 CBO
1103 1 CBO
1104 1 CDO     PACK LITERAL ENTRIES INTO FORMAT FOR STORAGE OF INTERFACE
1105 1 CDO     TABLE INTO AMA
1106 1 CDO
1107 1 CDO
1108 1 CDO
1109 1 CDO
1110 1 CDO
1111 1 CDO COMMON XB - ARGNO, DFLAG, IARG4, ISIZE,
1112 1 CDO     LENEFF, LITDSP, LITDOWN, LITPTR,
1113 1 CDO     LITSZ, NARG, NOKTM, NOBITM
1114 1 CDO     NUMARG, SFLAG, WXBLNG, WKBUF
1115 1 CDO
1116 1 CDO
1117 1 CDO
1118 1 CDO COMMON XB - LITDOWN, LITLEN, LITPTR, WKBUF
1119 1 CDO
1120 1 CDO
1121 1 CDO
1122 1 CDO
1123 1 CDO     INTERNAL VARIABLES
1124 1 CDO
1125 1 CDO COMMON XS - LITUP = INDEX INTO WKBUF OF AREA FOR NEXT LITERAL
1126 1 CDO     ENTRY TO BE MOVED INTO
1127 1 CDO COMMON XS - LITUP = INDEX INTO WKBUF OF LITERAL ENTRY TO BE
1128 1 CDO     COMPRESSED AND MOVED
1129 1 CDO
1130 1 CDO
1131 1 CDO
1132 1 CDO     NOTES
1133 1 CDO
1134 1 CDO
1135 1 CDO     USES ROUTINES
1136 1 CDO
1137 1 CDO XXEXT
1138 1 CDO XXBIT
1139 1 CDO XXMOV
1140 1 CDO XXXQ
1141 1 CDO XXSET
1142 1 CDO
1143 1 CDO
1145 1 * PACK LITERAL AREA INTO FORMAT FOR STORAGE OF INTERFACE TABLE
1146 1 BEGIN XILIT
1147 2 DO UNTIL ALL LITERAL AREAS PROCESSED
1148 3 DO UNTIL ALL ARGUMENTS SEARCHED
1149 4 IF THIS LITERAL ENTRY BELONGS TO THIS ARGUMENT, THEN
1150 5 IF ALL ELEMENTS OF THIS ARGUMENT ARE COMPLETE, THEN
1151 6 MOVE ALL DATA FOR LITERAL ENTRY UP IN WORKING BUFFER
1152 7 ELSE
1153 8 DO UNTIL ALL BITS OF BIT MASK PROCESSED
1154 9 IF THE BIT IS ON, THEN
1155 10 MOVE CORRECT NUMBER OF WORDS (LENEFF) OF LITERAL
1156 11 UP IN THE WORKING BUFFER
1157 12 ENDEF
1158 13 ENDDO
1159 14 ENDF
1160 15 CALCULATE NEW DISPLACEMENT AND SET IN LITDSP
1161 16 ENDF
1162 17 ENDDO
1163 18 ENDDO
1164 1 END XILIT
### Symbol Definition Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>202</td>
</tr>
<tr>
<td>ERROR2</td>
<td>616</td>
</tr>
<tr>
<td>ERROR3</td>
<td>782</td>
</tr>
<tr>
<td>ERROR4</td>
<td>379</td>
</tr>
<tr>
<td>ERROR5</td>
<td>709</td>
</tr>
<tr>
<td>ERROR6</td>
<td>617</td>
</tr>
<tr>
<td>ERROR7</td>
<td>710</td>
</tr>
<tr>
<td>ERROR8</td>
<td>618</td>
</tr>
<tr>
<td>ERROR9</td>
<td>783</td>
</tr>
<tr>
<td>ERR10</td>
<td>432</td>
</tr>
<tr>
<td>ERR11</td>
<td>433</td>
</tr>
<tr>
<td>ERR12</td>
<td>434</td>
</tr>
<tr>
<td>ERR13</td>
<td>784</td>
</tr>
<tr>
<td>EXIT</td>
<td>1004</td>
</tr>
<tr>
<td>FIVE</td>
<td>685</td>
</tr>
<tr>
<td>FIVE1</td>
<td>1001</td>
</tr>
<tr>
<td>FIVE2</td>
<td>1000</td>
</tr>
<tr>
<td>LIST</td>
<td>687</td>
</tr>
<tr>
<td>ZONE</td>
<td>994</td>
</tr>
<tr>
<td>PROMPT</td>
<td>686</td>
</tr>
<tr>
<td>RETURN</td>
<td>619</td>
</tr>
<tr>
<td>RETURN1</td>
<td>435</td>
</tr>
<tr>
<td>RETURN2</td>
<td>380</td>
</tr>
<tr>
<td>RETURN3</td>
<td>785</td>
</tr>
<tr>
<td>RETURN4</td>
<td>273</td>
</tr>
<tr>
<td>RETURN5</td>
<td>711</td>
</tr>
<tr>
<td>SIX</td>
<td>1002</td>
</tr>
<tr>
<td>THREE</td>
<td>998</td>
</tr>
<tr>
<td>TO</td>
<td>996</td>
</tr>
<tr>
<td>TOCHR</td>
<td>1047</td>
</tr>
<tr>
<td>TIDAT</td>
<td>502</td>
</tr>
<tr>
<td>TILDAT</td>
<td>1091</td>
</tr>
<tr>
<td>TILIT</td>
<td>1146</td>
</tr>
<tr>
<td>TILSD</td>
<td>835</td>
</tr>
<tr>
<td>TILSS</td>
<td>973</td>
</tr>
<tr>
<td>TILST</td>
<td>338</td>
</tr>
<tr>
<td>TINIX</td>
<td>113</td>
</tr>
<tr>
<td>TINPT</td>
<td>664</td>
</tr>
<tr>
<td>TINTE</td>
<td>35</td>
</tr>
<tr>
<td>TIPAR</td>
<td>752</td>
</tr>
<tr>
<td>TIPAY</td>
<td>276</td>
</tr>
<tr>
<td>TIPRM</td>
<td>195</td>
</tr>
<tr>
<td>TISUB</td>
<td>415</td>
</tr>
</tbody>
</table>

**Note:** The original layout was poor.
55 1 BEGIN XLMAN
56 2 CALL RRPAR TO GET INPUT PARAMETERS
57 3 INITIALIZE COMMON TO ZERO
58 4 SET CLASS NUMBER TO ZERO
59 5 CALL EXCM TO GET A CLASS NUMBER
60 6 EXIT XLMAN IF SECURITY CODE IS NOT VALID
61 7 CALL XRMV TO INITIALIZE TOKENS IN COMMON
62 8 DO FOREVER
63 9 :PROMPT:
64 10 :INITIALIZE MASTER AND SUBSTATE FLAGS
65 11 CALL XTCCM TO PROMPT FOR OPTION
66 12 IF XICM RETURN CODE IS NOT ZERO OR
67 13 FIRST TOKEN IS NOT AN INTEGER OR
68 14 INTEGER > 7 THEN
69 15 CALL XRMVE TO WRITE INVALID RESPONSE
70 16 GO TO :PROMPT:
71 17 ENDIF
72 18 CASE INTEGER (:XLRPM:, :XLP4M:, :XLRPM:, :XLP4D:, :XLMOD:, :
73 19 :XLMSE:, :XLDDBF:, :XLDDBF:, :EXIT:)
74 20 :XLRPM:
75 21 SET NUMOR TO INTEGER
76 22 CALL XELDS TO LOAD XLRPM TO CREATE SYSTEM PROMPT FILE
77 23 :XLP4M:
78 24 SET VALFLG TO SAY ORIGINAL XLP4M REQUEST
79 25 CALL XELDS TO LOAD XLP4M TO ADD A PROCESSOR
80 26 DO UNTIL VALFLG SAYS EXIT (X)
81 27 CALL XELDS TO LOAD XLIMIT TO ENTER DEFAULT VALUES
82 28 CALL XELDS TO LOAD XLP4M TO COMPLETE PROCESSING
83 29 ENDDO
84 30 :XLDPE:
85 31 CALL XELDS TO LOAD XDEL TO DELETE A PROCESSOR
86 32 :XLMOD:
87 33 SET VALFLG TO SAY ORIGINAL XLMOD REQUEST
88 34 CALL XELDS TO LOAD XLMOD TO MODIFY A PROCESSOR
89 35 DO UNTIL VALFLG SAYS EXIT (X)
90 36 CALL XELDS TO LOAD XLIMIT TO ENTER DEFAULT VALUES
91 37 CALL XELDS TO LOAD XLMOD TO COMPLETE PROCESSING
92 38 ENDDO
93 39 :XLMSE:
94 40 CALL XELDS TO LOAD XLMSE TO ADD A MESSAGE
95 41 :XLDDBF:
96 42 CALL XELDS TO LOAD XLDDBF TO HANDLE DATA BASE FILES
97 43 ENDDO
98 44 :EXIT:
99 45 CALL EXCM TO RELEASE CLASS NUMBER
100 1 END XLMAN
103 1 CD***********
104 1 CD
105 1 CD
106 1 CD
107 1 CD
108 1 CD
109 1 CD
110 1 CD
111 1 CD
112 1 CD
113 1 CD
114 1 CD
115 1 CD
116 1 CD
117 1 CD
118 1 CD
119 1 CD
120 1 CD
121 1 CD
122 1 CD
123 1 CD
124 1 CD
125 1 CD
126 1 CD
127 1 CD
128 1 CD
129 1 CD
130 1 CD
131 1 CD
132 1 CD
133 1 CD
134 1 CD
135 1 CD
136 1 CD
137 1 CD
138 1 CD
139 1 CD
140 1 CD
141 1 CD
142 1 CD
143 1 CD
144 1 CD
145 1 CD
146 1 CD
147 1 CD
148 1 CD
149 1 CD
150 1 CD
151 1 CD
152 1 CD
153 1 CD
154 1 CD
155 1 CD
156 1 CD
157 1 CD
158 1 CD***********
1 BEGIN XLPRM
   2 :PRM1:
   3 CALL XCOM TO PROMPT USER FOR OF DIRECTIVES
   4 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
   5 CALL CREAT TO CREATE PROMPT FILE
   6 IF FILE ALREADY EXISTS THEN
   7 CALL XCOM TO PROMPT USER FOR SCRATCH/CANCEL
   8 EXIT XLPRM IF RESPONSE IS CANCEL
   9 CALL PURGE TO PURGE PROMPT FILE
  10 ERREXIT IF PURGE ERROR TO :FILERR:
  11 GO TO :PRM1:
  12 ELSE (CREATE NEW FILE)
  13 ERREXIT IF CREATE ERROR TO :FILERR:
  14 CALL XCOM TO PROMPT USER FOR LIST OF DIRECTIVES
  15 ERREXIT IF LIST IS INCONSISTENT WITH 0 OF DIRECTIVES TO :ERR1:
  16 DO FOR EACH DIRECTIVE
  17 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
  18 CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
  19 ENDDO
  20 CALL WRITF TO WRITE LIST OF DIRECTIVES TO FILE
  21 ERREXIT IF WRITF ERROR TO :FILERR:
  22 DO FOR EACH DIRECTIVE
  23 CALL XCOM TO PROMPT USER FOR DEFINITION
  24 DO UNTIL EOD IS REACHED IN RESPONSE
  25 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
  26 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
  27 MOVE RESPONSE INTO BUFFER
  28 SET CONTROL CHARACTERS IN BUFFER
  29 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
  30 ENDGO
  31 SET REMAINDER OF BUFFER TO NULL
  32 CALL WRITF TO WRITE DEFINITION
  33 ERREXIT IF WRITF ERROR TO :FILERR:
  34 ENDGO
  35 CALL CLOSE TO CLOSE FILE
  36 ERREXIT IF CLOSE ERROR TO :FILERR:
  37 CALL XRMSG TO DISPLAY FILE CREATED MESSAGE
  38 2 ENDF
  39 1 END XLPRM
  40
  41 :ERR1: (ERROR IN RESPONSE)
  42 CALL XRMSG TO DISPLAY ERROR IN RESPONSE
  43 RETURN TO RESTART FOR ANOTHER RESPONSE
  44
  45 :FILERR: (FILE ACCESS ERROR)
  46 CALL XRMSG TO DISPLAY ERROR
  47 CALL CLOSE TO CLOSE FILE
  48 CALL PURGE TO PURGE FILE
  49 1 END XLPRM
FORTRAN CALLING SEQUENCE:

CALL KELDS ('XLPRO')

XLPRO WILL ADD A PROCESSOR TO THE LIBRARY DIRECTORY. IF
THE PROCESSOR HAS A DEFAULT INTERFACE TABLE, IT WILL ALSO
CREATE A PROMPT FILE AND INTERFACE TABLE FILE

INPUT FROM COMMON:

VALFLG - PROCESS CONTROL
0 - ORIGINAL REQUEST TO ADD
1 - COMPLETE PROCESSING

INTERNAL VARIABLES:

ABSTR - (INTEGER, 128 WORDS) CONTAINS THE ABSTRACT OF THE
PROCESSOR IN LATTER FORM. PRESENTLY IS A NULL RECORD.

BLOCKS - (INTEGER, 1 WORD) NUMBER OF BLOCKS TO ALLOCATE

TO INTERFACE TABLE FILE

DIRECT - (INTEGER, 6 WORDS) NAME ARARY CONTAINING VALID
DIRECTIVES FOR THE INTERFACE TABLE EDITOR

NAMEY - (INTEGER, 3 WORDS) COMBINATION OF PROCESSOR NAME

NAME AND VERSION NUMBER

PRMAM - (INTEGER, 3 WORDS) PROCESSOR NAME

PROMS - (IDENTIFIERS) PROMPTS FOR USER IF BE PROMPTED WITH

TYPE - (.TEGER, 1 WORD) TYPE OF PARAMETER USED IN

CALCULATING SIZE

VERS - (INTEGER, 1 WORD) VERSION NUMBER OF PROCESSOR

RTE AND FMR ROUTINES USED

EXEC, IAND, KEVT, CREAT, OPEN,
READ, WRIT, CLOSE, PURGE

FDS ROUTINES USED:

XLFL, XLMS, XLPL, XLPS
XREP, XRMV, XMSE, XRPC, XRSET, XRUP, XTCOM

COMMON USED:

< EQUATION >

CEX(3), ISCBO,
CEX(5), MASTA,
CEX(6), SUBSTA,
CEX(7), VALFLS,
CEX(16), PRMAH,
CEX(85), EOSTOK,
CEX(86), IMTOK,'
1  BEGIN XLPRO
2   IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
3     INITIALIZE MASTER AND SUB STATES
4     PERFORM LDID TO UPDATE LIBRARY DIRECTORIES
5     IF PROCESSOR HAS AN INTERFACE TABLE THEN
6         SET RETN = 2
7
8     PRINT "CALL XTCON TO PROMPT FOR 8 PARAMETERS"
9     ERREXIT IF XTCON RETURN CODE IS NOT ZERO OR
10        ERREXIT IF 8 PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
11     CREATE HEADEP ENTRY WITH 8 PARAMETERS AND PROCESSOR NAME
12     CALL XLISP TO WRITE INSTRUCTIONS FOR ENTERING SPECS
13
14     DO FOR 8 PARAMETERS
15
16     CALL XLSPS TO CREATE ONE PARAMETER ENTRY
17
18     SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
19
20     CALL XLIFL TO CREATE PROMPT FILE
21     PRINT "SET RETN = 3"
22
23     PRINT "CALL XTCON FOR DEFAULT VALUES DECISION"
24     ERREXIT IF RETURN CODE IS NOT ZERO OR
25        ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
26
27     IF RESPONSE WAS YE THEN
28         SET FLAG TO CALL INTERFACE TABLE EDITOR
29
30
31     EXIT XLPRO
32     ENDIF
33     ELSE
34         SET CODES ARRAY TO ADD ONLY ABSTRACT
35
36     CALL XLIFL TO CREATE PROMPT FILE
37     PERFORM XLPRO - NO RETURN EXPECTED
38     ENDIF
39
40     ENDIF
41     CALL TLFL TO CREATE THE DEFAULT INTERFACE TABLE FILE
42     SET VLFLAG TO SET ORIGINAL REQUEST TO ADD A PROCESSOR
43     PERFORM XLPRO - NO RETURN EXPECTED
44
45     PRINT "CALL XRAMS TO DISPLAY ERROR MESSAGE"
46     GO TO (:PRMPT1:,:PRMPT2:,:PRMPT3:), RETN
47
48     END XLPRO
BEGIN LIBRARY DIRECTORY
IF OPEN FILE ERRORS THEN
CALL READ AND CLOSE, OPEN LIBRARY DIRECTORY
ELSE IF ERROR THEN
CALL PIOUS TO MOVE NAME INTO XLDB
ENDIF
ENDIF
END
1 CD**********
2 CD
3 CD0
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0
9 CD0
10 CD0
11 CD0
12 CD0
13 CD0
14 CD0
15 CD0
16 CD0
17 CD0
18 CD0
19 CD0
20 CD0
21 CD0
22 CD0
23 CD0
24 CD0
25 CD0
26 CD0
27 CD0
28 CD0
29 CD0
30 CD0
31 CD0
32 CD0
33 CD0
34 CD0
35 CD0
36 CD0
37 CD0
38 CD0
39 CD0
40 CD0
41 CD0
42 CD0
43 CD0
44 CD0
45 CD0
46 CD0
47 CD0
48 CD0
49 CD0
50 CD0
51 CD0
52 CD0
53 CD0
54 CD0
55 CD0
56 CD0
57 CD0
58 CD0
59 CD0
60 CD0
61 CD0
62 CD0
63 CD0
64 CD0
65 CD0
66 CD0
67 CD0
68 CD0
69 CD0
70 CD0
71 CD0
72 CD0
73 CD0
74 CD0
75 CD0
76 CD0
77 CD0
78 CD0
79 CD0
80 CD0
81 CD0
82 CD0
83 CD0
84 CD0
85 CD0
86 CD0
87 CD0
88 CD0
89 CD0
90 CD0
91 CD0
92 CD0
93 CD0
94 CD0
95 CD0
96 CD0
97 CD0
98 CD0
99 CD0
100 CD0
101 CD0
102 CD0
103 CD0
104 CD0
105 CD0
106 CD0
107 CD0
108 CD0

FORTRAN CALLING SEQUENCE:

CALL XLCDB

XLCDB CREATES A NEW DATA BASE FILE (MDB/PDB) FROM AN OLD
DATA BASE FILE (MDB/PDB) AND DELETES THE OLD FILE

INPUTS IN COMMON:

XE(3) QUAL, XE(7) NUMBR, XE(8) SECU
XE(142) ICR, XE(5) OLDML, XE(6) NEWML
XE(9) TOTAL

INTERNAL VARIABLES:

XB(17) FREC - FIRST RECORD NUMBER IN DATBUF
XB(18) LREC - LAST RECORD NUMBER IN DATBUF
XB(40) IDC8 - DCB FOR OLDML
XB(56) IDC82 - DCB FOR NEWML
XB(200) TOCBUFF-BUFFER FOR COMPLETE DATA BASE TOC

RTE ROUTINES USED:
CLOSE, CREAT, KCVT, OPE, PURGE, READ, WRIT

FDS ROUTINES USED:
XDOBD, XREX, XRMSG

COMMON USED:

EQUIVALENCE (XE(3), QUAL), (XE(7), NUMBR), (XE(8), SECU)
+XE(142), XCR, (XB(3), OLDML),
+XB(9), NEWML, (XB(9), TOTAL),
+XB(10), FLCBR, (XB(13), OLDML),
+XB(113), MOTOC, (XB(14), SIZE),
+XB(15), MSE, (XB(16), TOCPTR),
+XB(17), FREC, (XB(18), LREC),
+XB(39), JERR, (XB(40), IDC8),
+XB(50), IDC82, (XB(52), IDC8),
+XB(200), TOCBUFF, (XB(200), TOCBUFF)
10 1 BEGIN XLDB
11 2 CALL OPEN to OPEN OLD FIL
12 3 ERREXIT IF OPEN ERROR to :ERR3:
13 4 CALL READ to READ IN HEADER RECORD
14 5 ERREXIT IF READ ERROR to :ERR2:
15 6 COMPUTE NUMBER OF TOC RECORDS (NOTOC) AS (# ENTRIES + 16) \times 16
16 7 IF NOTOC > 1 THEN
17 8 CALL READ to READ IN REMAINING TOC RECORDS
18 9 ERREXIT IF READ ERROR to :ERR2:
19 10 ENDIF
20 11 SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
21 12 CALL CREAT to CREATE NEWFIL
22 13 ERREXIT IF CREAT ERROR to :ERR2:
23 14 DO FOR EACH TOC ENTRY
24 15 IF REQUEST WAS RDB TO NEWFIL THEN
26 17 ERREXIT IF NAME > 4 CHARS OR
28 19 ERREXIT IF CLASS IS DRDE AND NAME > 2 CHARS TO :ERR1:
30 21 APPEND RDB CONVENTION TO FRONT OF NAME
32 22 ELSE
33 23 REMOVE RDB CONVENTION FROM NAME
34 24 ENDIF
35 25 ENDDO
36 26 CALL WRITE to WRITE NEW TOC RECORDS TO NEWFIL
37 27 ERREXIT IF WRITE ERROR to :ERR1:
38 28 SET TOCPTR to FIRST TOC ENTRY
39 29 SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
40 30 CONVERT SIZE FROM BLOCKS TO WORDS
41 31 DO UNTIL ALL RECORDS ARE COPied (SIZE = 0)
42 32 SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
43 33 IF SIZE < LENGTH THEN
44 34 SET LENGTH TO SIZE
45 35 ENDIF
46 36 CALL READ to READ LENGTH WORDS OF DATA
47 37 ERREXIT IF READ ERROR to :ERR1:
48 38 DECREMENT SIZE BY LENGTH READ
49 39 UPDATE FIRST AND LAST RECORD NUMBERS
50 40 START SEARCH FOR TOCPTR = TOCPTR to LAST TOC ENTRY
51 41 EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
52 42 IF CLASS IS AN INTERFACE TABLE THEN
53 44 COMPUTE INDEX INTO DATBUF FROM FIRST RECORD #, RECORD # IN TOC ENTRY
55 46 AMX INDEX IN TOC ENTRY
57 48 SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
59 50 ENDIF
61 52 ENDSERCH
63 54 CALL WRITE to WRITE LENGTH WORDS OF DATA TO NEWFIL
65 56 ERREXIT IF WRITE ERROR to :ERR1:
67 58 ENDDO
CALL CLOSE TO CLOSE NEWFIL
CALL CLOSE TO CLOSE OLDFIL
CALL XDDDB TO DELETE OLDFIL FROM PDB LOG FILE
CALL PURGE TO PURGE OLDFIL FROM SYSTEM
EXIT XLCDB

:ERR1:
CALL CLOSE TO CLOSE NEWFIL
CALL PURGE TO PURGE NEWFIL

:ERR2:
CALL CLOSE TO CLOSE OLDFIL

:ERR3:
IF REQUEST WAS PDB TO MDB THEN
SET QUAL TO SAT DELETE MDB FILE
ELSE (REQUEST WAS MDB TO PDB)
SET QUAL TO SAT DELETE PDB FILE
ENDIF
CALL XDDDB TO DELETE MDB/PDB FROM LOG FILE
IF ERROR WAS FILE MANAGER THEN
CALL XRM2G TO DISPLAY ERROR AND RETURN CODE
ELSE
CALL XRM2G TO DISPLAY ERROR
ENDIF
END XLCDB
FORTRAN CALLING SEQUENCE:

CALL XLODF ('XLDBF')

XLDBF IS THE DATABASE FILES MAIN ROUTINE. IT DETERMINES WHICH
DATA BASE ROUTINE IS NEEDED AND GATHERS INPUTS NEEDED BY THAT
ROUTINE. HANDLES ALL DB LOG FILE RELATED TRANSACTIONS.

INPUTS IN COMMON:

XE(5) ISCU, XE(7) NUMBR,
XE(85)TOKENS, XE(142) ICR

INTERNAL VARIABLES IN COMMON

XE(3) QUAL - USER QUALIFIER REQUIRED BY XDB ROUTINES AND XMOFM
XE(8) SECU - TEMPORARY SLOT FOR ISCU DURING THIS OVERLAY
XBER5 OLDFILE - OLD FILE NAME
XBER6 NEWFILE - NEW FILE NAME
XBER9 TOTAL - SIZE OF OLDFILE/NEWFILE IN BLOCKS
XBER10FILEN - 4 BASE CHARACTERS OF FILE NAME
XBER12QUALIF - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPENW

FBS ROUTINES USED:

XDBDA, XDBFY, XERTM, XLDDB, XLPCB
XPMO, XERT, XMSE, XMCM, XERF, XTCM

COMMON USERS:

ENQUEUENCE (XE(3), ISCU ),
+XE(3) QUAL , (XE(7), NUMBR ),
+XE(8) SECU , (XE(85), TOKENS),
+XE(142) ICR , (XE(145) COMBFU),
+XE(85)OLDFILE, (XE(6), NEWFILE),
+XE(9) TOTAL, (XE(10), FILEN),
+XE(12)QUALIF, (XE(99), IERR ),
+XE(100) ICRDB )
537 1 BEGIN XLDBF;
538 2 IF REQUEST IS TO CREATE/MODIFY LOG FILE THEN
539 3 CALL OPEN TO OPEN MDB/PDB LOG FILE
540 3 IF OPEN ERROR SAYS FILE NOT FOUND THEN
541 4 CALL XLCRN TO CREATE MDB/PDB LOG FILE
542 3 ELSE
543 4 ERREXIT IF OPEN ERROR TO :ERR:
544 4 CALL XLMOD TO MODIFY MDB/PDB LOG FILE
545 3 ENDIF
546 ELSE
547 DO UNTIL USER REQUESTS EXIT (X)
548 4 CALL XCOM TO PROMPT USER FOR NAME AND USER ID
549 4 IF RESPONSE IS NOT EXIT (PERCENT) THEN
550 5 ERREXIT IF RESPONSE IS INVALID TO :ERR:
551 5 SAVE A CHARACTER NAME AND ID IN COMMON
552 5 IF REQUEST WAS PDB TO MDB THEN
553 6 SET QUALIFIER TO SEARCH FOR PDB NAME
554 5 ELSE (REQUEST WAS FOR MDB TO PDB)
555 6 SET QUALIFIER TO SEARCH FOR MDB NAME
556 5 ENDIF
557 5 CALL XDBOV TO VERIFY EXISTENCE OF MDB/PDB DEPENDING ON QUALIFIER
558 5 ERREXIT IF NAME WAS NOT FOUND TO :ERR:
559 5 ERREXIT IF FILE MANAGER ERROR TO :ERR:
560 5 IF REQUEST WAS PDB TO MDB THEN
561 6 SET QUALIFIER TO ADD PDB TO LOG FILE
562 5 ELSE (REQUEST WAS MDB TO PDB)
563 6 SET QUALIFIER TO ADD MDB TO LOG FILE
564 5 ENDIF
565 5 CALL XDBOA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
566 5 ERREXIT IF DUPLICATE NAME OF
567 5 ERREXIT IF MAXIMUM NUMBER OF ENTRIES EXIST TO :ERR:
568 5 ERREXIT IF FILE MANAGER ERROR TO :ERR:
569 5 IF REQUEST WAS PDB TO MDB THEN
570 6 CALL XRFNM TO SET OLDFILE TO PDB NAME
571 6 SET NEWFILE TO MDB NAME
572 6 ELSE (REQUEST WAS MDB TO PDB)
573 6 SET OLDFILE TO MDB NAME
574 6 CALL XRFNM TO SET NEWFILE TO PDB NAME
575 5 ENDIF
576 5 CALL XLCBO TO COPY OLDFILE TO NEWFILE
577 4 ENDIF;
578 3 ENDIF
579 2 ENDIF
580 1 EXIT XLDBF
581 2 :ERR:
582 2 CALL XRXMSG TO DISPLAY ERROR
583 2 RETURN TO PROMPT FOR ANOTHER 4 CHARACTERS AND USER ID
584 2 :ERR:
585 2 CALL XRXMSG TO DISPLAY FILE ACCESS ERROR
586 1 END XLDBF
It at X7 ^t ^t at lt ft at l

OF THE

Ul^iGliv^ i.

IS POOR

< W	 V

W0. N

iJ 	 <H

~

O	 O	 O

01. i N F 	 W	 K

♦•

u<O 	 >~ W 	 K	 7t ^^

W/

►

t/l	 K Y Z 	 O	 • » N! 	 KKJ 	 <at < 	 IL	 M^ Vm0

N	 OH< 	 O1.-	 W	 N NOM

W	 >HJ 	 !W O 	 O	 K •t tJJ

d	 m<W	 <O u 	 W	 V ^f^MVIJ	 >rW0	 F	 a. MrNW	 -I ac 	 N

217x291­

at V

69	 z

O O W	 < V d	 ! M	 Yt W

« Z V

♦

O O W	 < V d	 ! M	 Yt W

1tj**j}'^ t^	 ^ 7t F M • M 	 M aC	 W	 ft	 M

r^w^
1 BEGIN XLDEL

2 PROMPT:
3 CALL XLCA TO PROMPT FOR PROCESSOR NAME
4 IF XLCA RETURN CODE IS NOT ZERO OR
5 RESPONSE IS NOT A VALID PROCESSOR NAME THEN
6 CALL XRMG TO WRITE "ERROR MESSAGE"
7 GO TO :PROMPT:
8 EXIT XLDEL IF RETURN CODE SAYS % ENTERED
9 ENDF:
10 CALL OPEN, READ AND CLOSE TO READ IN LIBRARY DIRECTORY
11 ERREXIT IF FILE ERROR TO :FILERR:
12 IF PROCESSOR IS NOT IN LIBRARY DIRECTORY THEN
13 CALL XRMG TO DISPLAY "ERROR"
14 GO TO :PROMPT:
15 ENDF:
16 RECENT # PROCESSORS BY 1
17 CALL PURGE TO PURGE OLD LIBRARY DIRECTORY
18 ERREXIT IF FILE ERROR TO :FILERR:
19 EXIT XLDEL IF #PROCESSORS IS ZERO
20 CALL CREAT, WRITF AND CLOSE TO RECREATE LIBRARY DIRECTORY
21 ERREXIT IF FILE ERROR TO :FILERR:
22 CREATE THE PROMPT FILE NAME
23 CALL PURGE TO PURGE THE PROMPT FILE
24 ERREXIT IF PURGE ERROR TO :FILERR:
25 CALL XRMG TO SAY FILE PURGED SUCCESSFULLY
26 IF PROCESSOR HAD AN INTERFACE TABLE THEN
27 CREATE DEFAULT IT NAME
28 CALL PURGE TO PURGE DEFAULT IF
29 ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:
30 CALL XRMG TO DISPLAY "FILE :AGED" MESSAGE
31 ENDF:
32 GO TO :PROMPT:
33 :FILERR: CALL XRMG TO WRITE FILE ACCESS ERROR
34 1 END XLDEL
**FUNCTION CALL SEQUENCE:**

- **CD00**

CALL XLINS

**INTERNAL VARIABLES:**

- **CD01**

PROMPT - INTEGER, 120 WORDS) ALL 7 LINES IN AN ARRAY TO BE DISPLAYED

**RTE ROUTINES USED:**

- **CD05**

EXEC

**COMMON USED:**

- **CD05**

EQUIVALENCE (RE(1), LU )

**BEGIN XLINS**

**CALL EXEC TO DISPLAY ALL 7 LINES**

**END XLINS**
FORTRAN CALLING PROCEDURE:

CALL XELDS (XLINT)

XLINT SEGMENT SETS UP COMMON TO CALL THE INTERFACE TABLE TO
ACCEPT DEFAULT VALUES FOR THE INTERFACE TABLE

FD3 FUNCTIONS AND SUBROUTINES USED:

XEINT, XERTM, XIMIX, XMOV

COMMON USED:

EQUIVALENCE (XES(5), MALLSTA),
*(XE(6), SUBSTA), (XH(17), NUMDR),
*(XH(2), DIRECT), (XH(25), LSTFLG),
*(XH(37), ARCH), (XH(51), PRINTM),
*(XH(73), ISIZE), (XH(89), LITDOM),
*(XH(90), LITPRT), (XH(91), LITLEN),
*(XH(92), NAME),
*(XH(93), NUMARG), (XH(97), HNCTAB),
*(XH(100), WRLN), (XH(107), WKBUFF),
*(XH(140), EMB )
FORTRAN CALLING SEQUENCE:
CALL XMODS ('XMOD')

XMOD ALLOWS A USER TO MODIFY THE VERSION, SPECS, AND PROMPTS FOR A PROCESSOR AS WELL AS ADD/DELETE PARAMETERS FROM AN INTERFACE TABLE.

RTE ROUTINES USED:
CLOSE, IAND, OPEN, PONRT, READE, WRITF

FDS ROUTINES USED:
XLSTL, XLMIG, XLSPD, XREPB, XREX, XREMV, XRENS, XREP, XREM, XREX, XRENC

COMMON USED:

---

EROS Contents
BEGIN XLMOD
1 IF VALSEG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
2 SET RTN = 1
3 
4 IF XCMN RETURN CODE NON-ZERO OR
5 ERREXIT IF XCMN RETURN CODE NON-ZERO OR
6 ERREXIT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO :PMERR:
7 CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
8 ERREXIT IF THERE IS A FILE ERROR TO :FILERR:
9 ERREXIT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO :PMERR:
10 SAVE INTERFACE TABLE BIT AND VERSION NUMBER
11 PERFORM VERSION TO UPDATE VERSION NUMBER
12 IF THE PROCESSOR HAD AN INTERFACE TABLE THEN
13 SET NEW VERSION NUMBER IN INTERFACE TABLE
14 CREATE DEFAULT INTERFACE TABLE NAME
15 CALL OPEN AND READF TO READ IN HDR AND SPECS
16 IF THERE ARE LITERALS THEN
17 CALL READF TO READ IN LITERALS
18 ENDIF
19 CALL CLOSE TO CLOSE FILE
20 ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
21 CREATE PROMPT TABLE NAME
22 CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
23 ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
24 CALL NAMF TO RENAME PROMPT FILE >XLMP
25 ERREXIT IF NAMF ERROR TO :FILERR:
26 SET CODES ARRAY TO MODIFY-ABSTRACT AND NO CHANGES TO PARAMETER SPECS
27 PERFORM DELPM TO DELETE PARAMETERS
28 PERFORM MODPM TO MODIFY PARAMETERS
29 PERFORM ADDPM TO ADD PARAMETERS
30 CALL XLPLF TO CREATE NEW PROMPT FILE
31 PERFORM DEFALT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
32 ELSE
33 CALL NAMF TO RENAME PROMPT FILE >XLMP
34 ERREXIT IF NAMF ERROR TO :FILERR:
35 SET CODES ARRAY TO MODIFY ABSTRACT ONLY
36 CALL XLPLF TO CREATE NEW PROMPT FILE
37 PERFORM XLMOD - NO RETURN EXPECTED
38 ENDIF
39 CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
40 ERREXIT IF FILE ERROR TO :FILERR:
41 CALL XLFLF TO CREATE NEW DEFAULT INTERFACE TABLE FILE
42 SET VALFILE TO SAY ORIGINAL REQUEST TO MODIFY
43 PERFORM XLMOD - NO RETURN EXPECTED
44 
45 IF VALFILE SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
46 SET RTN = 1
47 
48 CALL XLFLF TO CREATE NEW DEFAULT INTERFACE TABLE FILE
49 ERREXIT IF FILE ERROR TO :FILERR:
50 
51 ENDF
52 Call XMSG TO DISPLAY ERROR MESSAGE
53 GO TO (:PMOD1, :PMOD2, :PMOD3, :PMOD4, :PMOD5, :PMOD6, :PMOD7), RTN
54 Call XMSG TO DISPLAY FILE ERROR
55 END XLMOD
BEGIN VERSION
SET RTN = 2
BEGIN PROM1:
CALL XCTOM TO PROMPT FOR VERSION NUMBER
IF RETURN CODE IS NOT CR THEN
EREREOT IF RETURN CODE IS NON-ZERO ON
EREREOT IF VERSION IS INVALID TO PROM1:
IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
CALL XMODE TO PUT NEW VERSION IN ENTRY
CALL OPEN, WRITE, CLOSE TO UPDATE LIBRARY DIRECTORY
EREREOT IF FILE ERROR TO PROM1:
ENDIF
END IF
END VERSION

BEGIN PROM2:
DO UNTIL RETURN CODE IS CR ENTERED
IF NUMBER OF PARAMETERS > 1 THEN
CALL XCTOM TO PROMPT FOR DELETE PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPM TO INTERPRET RESPONSE
PERFORM CHAT TO CHECK FOR EXISTING DATA
SET ARGNO TH WORD IN CODES TO SAY 'DELETED'
CALL XNOM TO MOVE DATA TO DELETE PARAMETER
DECEMENT # PARAMETERS BY 1
ENDIF
ELSE
CALL XRMUS TO DISPLAY NO PARAMETERS CAN BE DELETED
EXIT DELEPAM
ENDIF
END PROM2

BEGIN PROM3:
DO UNTIL RETURN CODE IS CR ENTERED
CALL XCTOM TO PROMPT FOR MODIFY PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPM TO INTERPRET RESPONSE
PERFORM CHAT TO CHECK FOR EXISTING DATA
SET ARGNO TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
CALL XNOM TO MOVE DATA TO SPECIFICATIONS
SET IT COMPLETE BIT OFF
ENDIF
END PROM3

END PROM3
END MODPRAM
959 1 BEGIN ADDPRM
960 2 SET RTN = 5
961 2 :PROMS:
962 2 DO UNTIL RETURN CODE IS CR ENTERED
963 3 IF NUMBER OF PARAMETERS < 63 THEN
964 4 CALL XTCOM TO PARMOPT FOR ADD PARAMETER BEFORE/AFTER PARAMETER NAME
965 4 IF RETURN CODE IS NOT CR ENTERED THEN
966 5 PERFORM RSPHD TO INTERPRET RESPONSE
967 5 SET ARGNO TH NON-DELETED WORD IN CODES TO SAY 'ADDED'
968 5 INCREMENT NUMBER OF PARAMETERS BY 1
969 5 CALL XMMOV TO MOVE DATA TO MAKE SPACE FOR NEW PARAMETER
970 5 CALL XILSPS TO GET NEW SPEC FOR THIS PARAMETER
971 5 SET IT COMPLETE BIT OFF
972 4 ENDIF
973 3 ELSE
974 4 CALL XMMSG TO DISPLAY NO MORE PARAMETERS CAN BE ADDED
975 3 EXIT ADDPRM
976 3 ENDIF
977 2 ENDOD
978 1 END ADDPRM
979 1 +
980 1 *
981 1 *
982 1 BEGIN DEFAULT
983 2 SET RTN = 7
984 2 :PROMS:
985 2 CALL XTCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
986 2 CALL XTCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
987 2 ERREXIT IF RETURN CODE IS NON-ZERO TO :PMERR:
988 2 IF RESPONSE IS YES THEN
989 3 SET VALFLAP TO SAY CALL INTERFACE TABLE EDITOR
990 2 EXIT XLMOD
991 2 ENDIF
992 1 END DEFAULT
BEGIN

EREXIT IF RETURN CODE IS NON-ZERO TO SPHERAE:

EREXIT IF NEGATIVE FLAG ON

END

BEGIN

IF DOUBLE LITERAL ENTRY LENGTH IS ON

ELSE IF THERE IS COMPLETE LITERAL DATA THEN

END

5-146
1017    1C0************
1018    1C00        FORTRAN CALLING PROCEDURE
1019    1C00        CALL XELDS ('XLM5G')
1020    1C00        CALL XELDS ('XLM4G')
1021    1C00        CALL XELDS ('XLM3G')
1022    1C00        CALL XELDS ('XLM2G')
1023    1C00        CALL XELDS ('XLM1G')
1024    1C01        XLM5G PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XRM5G
1025    1C01        XLM4G
1026    1C01        XLM3G
1027    1C01        XLM2G
1028    1C01        XLM1G
1029    1C02        INPUT
1030    1C02        XE COMMON - LU, ISEGU, FLAGS, TOKENS, ICR
1031    1C02        TERMINAL - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES
1032    1C02        UPDATING MODE, MESSAGE NUMBER AND TEXT
1033    1C02        MESSAGE FILE - DIRECTORY AND OLD TEXT
1034    1C02        MESSAGE FILE - DIRECTORY AND TEXT UPDATES
1035    1C02        LOCAL
1036    1C02        AREA - NUMERICAL AREA INDICATOR FOR MESSAGE
1037    1C02        DIRECT - MESSAGE DIRECTORY (SEE SDD 6.2.4.12)
1038    1C02        INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY
1039    1C02        IDCB - FILE MANAGER DATA CONTROL BLOCK
1040    1C02        ERR - FILE MANAGER & XTCOM RETURN CODE
1041    1C02        MUMB - MESSAGE NUMBER WITHIN MESSAGE AREA
1042    1C02        NMBLKB - BLOCK NUMBER WITHIN MESSAGE
1043    1C02        CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION
1044    1C02        UPDATE MODE - NUMBER OF BLOCK CONTAINING MESSAGE
1045    1C02        RECPM - MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 33, 65 or 97)
1046    1C02        NOTES
1047    1C05        USES APOSN, CLOSE, CREAT, EXEC, IAND, KCVT, OPEN, READF, WRTF,
1048    1C05        XORIN, XOR, XRM5G, XRM4G, XRM3G, XTCOM, XRDG
1049    1C05        WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE
1050    1C05        EXISTING TEXT IN PLACE.
1051    1C05        MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X
1067 1 BEGIN XLMSG
1068 2 OPEN XLMSG
1069 3 IF FILE NOT FOUND
1070 4 THEN
1071 5 OUTPUT "MESSAGE FILE CREATION"
1072 6 DO FOR EACH OF THE 32 DIRECTORY ENTRIES
1073 7 PROMPT FOR AREA ID AND MAXIMUM NUMBER OF MESSAGES FOR THIS ENTRY NUMBER
1074 8 IF RESPONSE WAS NULL
1075 9 THEN
1076 10 CLEAR ENTRY
1077 11 ELSE
1078 12 STORE ID
1079 13 COMPUTE AREA ORIGIN AND STORE
1080 14 CLEAR LAST MESSAGE NUMBER
1081 15 COMPUTE NUMBER OF BLOCKS AND STORE
1082 16 ENDIF
1083 17 ENDDO
1084 18 CREATE A CLEARED FILE OF TOTAL REQUIRED SIZE
1085 19 ELSE
1086 20 READ DIRECTORY
1087 21 ENDIF
1088 22 DO UNTIL USER INPUTS X
1089 23 PROMPT FOR MESSAGE NUMBER
1090 24 SEPARATE AREA AND MESSAGE NUMBER AND COMPUTE BLOCK NUMBER AND MESSAGE LOC
1091 25 IF VALID AREA AND BLOCK NUMBER <= NUMBER OF BLOCKS
1092 26 THEN
1093 27 READ BLOCK
1094 28 IF FIRST WORD OF MESSAGE IS NOT NULL (MESSAGE ALREADY EXISTS)
1095 29 THEN
1096 30 DISPLAY OLD MESSAGE TEXT
1097 31 ENDIF
1098 32 PROMPT FOR TEXT
1099 33 IF NON-NULL RESPONSE
1100 34 THEN
1101 35 STORE TEXT IN BLOCK
1102 36 REWRITE BLOCK
1103 37 IF MESSAGE NUMBER > LAST MESSAGE NUMBER
1104 38 THEN
1105 39 REPLACE LAST MESSAGE NUMBER WITH NEW NUMBER
1106 40 ENDIF
1107 41 ENDIF
1108 42 ELSE
1109 43 OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110 44 ENDIF
1111 45 ENDDO
1112 46 REWRITE DIRECTORY BLOCK
1113 47 END XLMSG
1149 1 BEGIN XL CR
1150 2 CALL CREAT TO CREATE MDB/PDB LOG FILE
1151 2 ERREXIT IF CREATE ERROR TO :FILERR:
1152 2 INITIALIZING LOG RECORD BUFFER TO ZEROS
1153 2 SET # MDB FILES CURRENTLY USED TO ZERO
1154 2 SET MAXIMUM NUMBER MDBS TO 20
1155 2 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
1156 2 ERREXIT IF WRITF ERROR TO :FILERR:
1157 2 DO FOR EACH REMAINING LOG RECORD
1158 3 DO FOR THIS PAIR OF USER ID'S
1159 4 CALL XCTOM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
1160 4 SET MAXIMUM # PDB FILES TO RESPONSE
1161 4 SET # PDB FILES CURRENTLY USED TO ZER0
1162 3 ENDDO
1163 3 CALL WRITF TO WRITE 1 PDB RECORD TO LOG FILE
1164 3 ERREXIT IF WRITF ERROR TO :FILERR:
1165 2 ENDDO
1166 2 CALL CLOSE TO CLOSE FILE
1167 2 ERREXIT IF CLOSE ERROR TO :FILERR:
1168 1 EXIT XL CR
1169 2 :FILERR:
1170 2 CALL XRMSS TO DISPLAY FILE ERROR
1171 2 CALL CLOSE TO CLOSE FILE
1172 1 END XL CR
FORTRAN CALLING SEQUENCE:

CALL XLPL (NOCOD, CODES)

INPUTS IN CALLING SEQUENCE:

NOCOD - NUMBER OF CODES IN THE CODE ARRAY
CODES - ARRAY OF CODES THAT REPRESENT:
  CODES(1) PROCESSOR ABSTRACT
CODES(2) THRU CODES(NOCOD) PARAMETERS 1 THRU N
  WITH VALUES OF:
  0 - NO CHANGE
  1 - MODIFIED
  2 - DELETED
  3 - ADDED

INPUTS IN COMMON:

XE(1) LU, XE(3) ISECU, XE(16) PRCNAM,
XE(142) LCR, XB(96) NOPARMS, XB(108) PARMS

RTE FUNCTIONS USED:

CLOSE, CREAT, EXEC, OPEN, READ, READF, WRIT

FDS FUNCTIONS USED:

XERTN, XRCPR, XPMOV, XRMGS, XRTPCK, XRUPK, XTOM

COMMON USED:

EQUIVALENCE, XE(1), LU,

+ XE(3), ISECU, XE(7), VALPL6,
+ XE(16), PRCNAM, XE(85), TOKEMS,
+ XE(142), LCR, XE(144), COMPR,
+ XE(145), COMBUP, XB(96), NOPARMS,
+ XB(108), PARMS
1 BEGIN XLPL
2  COMPUTE SIZE OF FILE AS 9 PARAMETERS +3
3  CALL CREAT TO CREATE PROMPT FILE
4  ERREXIT IF CREAT ERROR TO :FILERR:
5  STUFF SYNTAX RECORD 9 AND 9 PARAMETERS INTO LIST OF SHORT PROMPTS
6  CALL WRTTTF TO WRITE SHORT PROMPT RECORD(S)
7  ERREXIT IF WRITE ERROR TO :FILERR:
8  IF ABSTRACT CODE IS NOT ABORT THEN
9  CALL OPEN TO OPEN OLD PROMPT FILE >XL TMP
10  ERREXIT IF OPEN ERROR TO :FILERR:
11  CALL READ TO READ EXISTING ABSTRACT
12  ERREXIT IF READ ERROR TO :FILERR:
13  CALL EXEC TO DISPLAY EXISTING ABSTRACT
14  CALL XTCOM TO PROMPT USER TO MODIFY ABSTRACT
15  IF RESPONSE IS CR (NO MODIFICATION) THEN
16    CALL WRTTTF TO WRITE EXISTING ABSTRACT TO NEW FILE
17    ERREXIT IF WRITE ERROR TO :FILERR:
18  ELSE (NEW ABSTRACT WAS ENTERED)
19    PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
20  ENDIF
21  ELSE (ABSTRACT CODE IS AB)'
22  CALL XTCOM TO PROMPT USER TO ENTER NEW ABSTRACT
23  PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
24  ENDIF
25  IF PARAMETERS IS NOT ZERO THEN
26    PERFORM EXTPTN TO GET DEFINITIONS FOR EACH PARAMETER
27    ENDIF
28  CALL CLOSE TO CLOSE PROMPT FILE
29  ERREXIT IF CLOSE ERROR TO :FILERR:
30  CALL CLOSE TO CLOSE OLD FILE >XL TMP
31  CALL PURGE TO PURGE OLD FILE >XL TMP
32  ENDIF
33  EXIT XLPL:
34  EXIT
35  :FILEER:
36  CALL XROSG TO DISPLAY ERROR CODE
37  CALL CLOSE TO CLOSE NEW FILE
38  CALL CLOSE TO CLOSE OLD FILE >XL TMP
39  CALL PURGE TO PURGE OLD FILE >XL TMP
40  SET VALFLG = 0
41  CALL XTNTN TO RETURN TO MAIN ***NO RETURN TO HERE***
42  END XLPL
01  BEGIN EXPRM
02  DO FOR EACH ENTRY IN CODES ARRAY
03      BEGIN CASE (:NOCHG; :MOD; :DEL; :ADD); CODES*1
04      :NOCHG: (SPECs HAD NO CHANGES)
05          CALL READ TO READ EXISTING DEFINITION FROM >XLTP
06          ERREXIT IF READF ERROR TO :FILERR:
07          CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
08          ERREXIT IF WRITE ERROR TO :FILERR:
09      :MOD: (SPECs WERE MODIFIED)
10          CALL READ TO READ EXISTING DEFINITION FROM >XLTP
11          ERREXIT IF READF ERROR TO :FILERR:
12          CALL EXEC TO DISPLAY EXISTING DEFINITION
13          IF RESPONSE IS CR (NO RESPONSE) THEN
14          CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
15          ELSE (NEW DEFINITION WAS ENTERED)
16          PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
17          ENDF
18      :DEL: (SPECs WERE DELETED)
19          CALL POSN TO POSITION >XLTP OVER THIS ENTRY
20          ERREXIT IF POSNT ERROR TO :FILERR:
21      :ADD: (SPECs WERE ADDED)
22          CALL XTOM TO PROMPT USER TO ENTER NEW DEFINITION
23          PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
24      END CASE
25      DO UNTIL RESPONSE IS CR (EXIT)
26          CALL XTOM TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
27          EXIT EXPRM IF REASON IS CR
28          SEARCH SHORT PROMPTS FOR RESPONSE
29          ERREXIT IF NOT FOUND TO :ERR:
30          CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
31          ERREXIT IF READF ERROR TO :FILERR:
32          CALL EXEC TO DISPLAY EXISTING DEFINITION
33          CALL XTOM TO PROMPT USER TO MODIFY DEFINITION
34          IF RESPONSE IS POSITIVE
35          CALL POSN TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
36          ERREXIT IF POSNT ERROR TO :FILERR:
37          PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
38      ENDF
39  END EXPRM
40  ENDO
FORTTRAN CALLING SEQUENCE:

CALL XLPHO

XLPHO MODIFIES THE EXISTING PDB LOG FILE

INPUTS FROM COMMON:

XE(1) LU, XE(3) GUAL, XE(8) SECU,
XE(142) ICA, XE(99) IERR, XE(100) IOCBO.

INTERNAL VARIABLES IN COMMON:

XE(5) UNO - NUMBER OF USER ID A-Z (1-26)
XE(51) RECNO - RECORD # WHERE UNO'S PDB LIST IS (2-14)
XE(99) PHAM - FILE NAME OF USER'S PDB (FULL 6 CHAR)  
XE(65) DIFF - NUMER OF FILES THAT MUST BE DELETED
XE(244) OUTFIL - AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

RTE ROUTINES USED:

CLOSE, EXEC, KEVT, PURGE, R4ADF, WRITF

FDS ROUTINES USED:

XCPRA, XREXT, XPRMV, XRMAG,
XRGMN, XRFR, XFCOM

COMMON USED:

equivalence (XE(1), LU ),
+ (XE(3), GUAL ), (XE(8), SECU ),
+ (XE(65), TOKENS), (XE(142), ICA ),
+ (XE(144), COMPTR), (XE(99), IERR ),
+ (XE(5), UNO ), (XE(51), RECNO ),
+ (XE(52), PHAM ), (XE(55), DIFF ),
+ (XE(56), RTN ), (XE(57), MSO ),
+ (XE(99), IERR ), (XE(100), IOCBO ),
+ (XE(116), IBUF ), (XE(244), OUTFIL ),
+ (XE(304), IOCBO )
1387 1 BEGIN XLPMO
1388     DO UNTIL USER RESPONSES EXIT (3)
1389     CALL XTRIM TO PROMPT USER FOR USER ID
1390     ERREXIT IF RESPONSE IS INVALID TO :ERR1:
1391     SAVE USER ID AND NUMBER (1-26) IN COMMON
1392     CALL READ TO READ IN THE RECORD CONTAINING THIS ID
1393     ERREXIT IF READ ERROR TO :FILERR:
1394     DISPLAY MAX # ALLOWED FOR THIS USER AND # CURRENTLY USED
1395     CALL XTRIM TO PROMPT USER FOR NEW MAXIMUM
1396     IF RESPONSE IS A MODIFICATION (NOT (CR)) THEN
1397     ERREXIT IF RESPONSE IS NOT VALID TO :ERR1:
1398     COMPUTE DIFFERENCE AS # CURRENTLY USED - NEW MAXIMUM
1399     IF DIFFERENCE <= 0 THEN
1400     CALL WRITE TO REWRITE UPDATED RECORD TO FILE
1401     ERREXIT IF WRITE ERROR TO :FILERR:
1402     ELSE
1403     DISPLAY LIST OF CURRENT FILES
1404     SET #PURGED = 0
1405     DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1406     CALL XTRIM TO PROMPT USER TO DELETE #OVER FILES
1407     DO FOR EACH FILE NAME IN RESPONSE
1408     SEARCH LIST FOR FILE NAME
1409     ERREXIT IF INVALID NAME TO :ERR1:
1410     MOVE FILE NAME TO PURGE LIST
1411     INCREMENT #PURGED BY 1
1412     DECREMENT DIFFERENCE BY 1
1413     COMPRESS OLD FILE NAME OUT OF LIST
1414     ENDDO
1415     ENDDO
1416     IF RESPONSE WAS NOT EXIT THEN
1417     CALL WRITE TO REWRITE UPDATED LOG RECORD
1418     ERREXIT IF WRITE ERROR TO :FILERR:
1419     DO FOR #PURGED FILES
1420     CALL XTFN TO CREATE FILE NAME
1421     CALL PURGE TO PURGE FILE
1422     ENDDO
1423     ENDDIF
1424     ENDDIF
1425     ENDIF
1426     DO FOR CLOSE TO CLOSE MDB/PDB LOG FILE
1427     EXIT XLPMO
1428     ERR1:
1429     ERREXIT:
1430     CALL XTRIM TO DISPLAY ERROR IN RESPONSE
1431     RETURN TO REISSUE LAST PROMPT
1432     FILERR:
1433     CALL XTRIM TO DISPLAY FILE ERROR
1434     CALL CLOSE TO CLOSE MDB/PDB FILE
1435     END XLPMO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

CLASS - (INTEGER, 1 WORD, XE(12)) IS THE PARAMETER CLASS
IPMT - (INTEGER, 1 WORD) IS THE INDEX INTO PARM WHERE
THE SHORT PROMPT IS TO BE STORED
ISPEC - (INTEGER, 1 WORD) IS THE INDEX INTO PARM WHERE
THE SPEC ARE TO BE STORED
MSG - (INTEGER, 1 WORD) IS THE CURRENT ERROR CONDITION
TYPE - (INTEGER, 1 WORD, XE(11)) IS THE PARAMETER TYPE

ROUTINES USED:

KCVT, XRCPR, XRMOV, XRMST, XRSST, XTRM

XE COMMON USED:

EQUVALENCE (XE(8), RETC ), (XE(9), RETM ),
(XE(10), CLASS ), (XE(11), TYPE ),
(XE(145), COMBUF ), (XE(37), ARGNO ),
(XE(96), NOPARM ), (XE(108), PARMS )
1474 1 BEGIN XLSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
1476 3
1477 4 :PRMPT1:
1478 5 CALL XCOM TO PROMPT FOR PROMPT, CLASS, TYPE AND I/O FLAGS
1479 6 ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
1480 7 INITIALIZE THIS SHORT PROMPT AND ENTRY
1481 8 ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
1482 9 ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
1483 0 CALL XMNAP TO MOVE SHORT PROMPT INTO SPECS
1484 1 CALL XRSET TO SET CLASS IN ENTRY
1485 2 CALL XRSET TO SET TYPE IN ENTRY
1486 3 ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
1487 4 ERREXIT IF CLASS IS ORDE AND TYPE IS SYMBOLIC STRING TO :PRMERR:
1488 5 CALL XRSET TO SET I/O FLAGS IN ENTRY
1489 6 ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
1490 7 ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
1491 8 CALL XRSET TO SET I/O FLAGS IN ENTRY
1492 9 IF CLASS IS DATA ELEMENT THEN
1493 0 IF TYPE IS NOT SYMBOLIC STRING THEN
1494 1 SET RETRN TO 2
1495 2
1496 3 :PRMPT2:
1497 4 CALL XCOM TO PROMPT FOR I AND J DIMENSIONS
1498 5 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1499 6 ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
1500 7 IF J DIMENSION IS NOT ENTERED THEN
1501 8 ELSE
1502 9 ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
1503 0 IF J DIMENSION < 2 THEN
1504 1 SET J DIMENSION TO 1
1505 2 ELSE
1506 3 STORE I DIMENSION IN ENTRY
1507 4 :EOF
1508 5 ENDIF
1509 6 SET SIZE = I DIMENSION * J DIMENSION * TYPE LENGTH
1510 7 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1511 8 ELSE
1512 9 SET RETRN TO 3
1513 0
1514 1 XCOM TO PROMPT FOR MAXIMUM SIZE
1515 2 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1516 3 ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
1517 4 SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
1518 5 ENDIF
1519 6 ENDIF
1520 1 EXIT XLSPS
1521 2 :PRMERR: CALI XMNGE TO DISPLAY ERROR MESSAGE
1522 3 GO TO (:PRMPT1, :PRMPT2, :PRMPT3), RETN
1523 4 END XLSPS
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 *01 FDS MANAGER. SCHEDULED BY FDS CONFIGURATION MANAGER. ONE PROGRAM
2 *01 PER SIGNED ON USER.
3
4 **********
5 *02 INPUT
6 *02 FDS STATUS TABLE (SEE SDD 6.2.2)
7 *02 FDS MANAGEMENT REQUESTS (SEE SDD 6.2.6.2)
8 *02
9 **********
10 *03 OUTPUT
11 *03 FDS MANAGER RESPONSES (SEE SDD 6.2.6.3)
12
13 **********
14 *05 NOTES
15 *05 USES XMNG, XMAVA, XMFW, XMCH, XMGT, XUDMP
16 *05 RMPAR, SLIB, ANRQ, CHNPAR, XEeq
17 *05
18 *05 XMAVA IS A MANAGER GLOBAL DATA AREA CONTAINING THE AWA AND VARIOUS
19 CONTROL VALUES
20 1 *05
21 1 **********
22
23 1 #
24 1 *
25 1 *
26 1 BEGIN XMGR
27 2 # INPUTS: P1= LU
28 2 # P2= XVSTB ENTRY ADDRESS
29 2 # P3= USER ID
30 2 # P4= OPTIONS
31 2 # P5= NUMBER OF DWA TRACKS
32 2 CALL RM?AR
33 2 LOCK THE MANAGER INTO THE PARTITION
34 2 SET XVSTA( FDS STATUS TABLE ENTRY ADDRESS)
35 2 RESOLVE XVSTB ADDRESS
36 2 CALL XMGR INITIALIZE DWA FUNCTIONS
37 2 IF ERROR RETURN THEN
38 2 EXIT TO :XREM
39 2 ENDIF
40 2 GET EXEC'S CLASS NUMBER
41 2 GET PROCESSOR'S CLASS NUMBER
42 2 CALL SLIB.get privileged
43 2 SET EXEC'S CLASS NUMBER IN STBEC
44 2 SET PROCESSOR'S CLASS NUMBER IN STBPC
45 2 SET CURRENT TASK IN STBAT FROM EXEC ID STBEX
46 2 CALL SLIB ENABLE
47 2 FORM EXEC'S NAME FROM 'KEY' & ASCII LU(STBLA)
48 2 SCHEDULE EXEC WITH WAIT
49 2 CALL XMCM
50 2 DO UNTIL EXEC REQUESTS TERMINATION OR ABENDS(P1=9 OR -32768)
51 2 IF REQUEST IS IN P1
52 3 IF REQUEST IS FOR AWA MANAGEMENT (P1=1) THEN
53 4 PERFORM XMNG
54 5 ELSE
55 6 IF REQUEST IS FOR SEQUENCE TABLE EXECUTION (P1=2) THEN
56 7 PERFORM' XMGS
57 8 ELSE
58 9 IF REQUEST IS TERMINATION (P1=9 OR -32768) THEN
59 10 ISSUE MESSAGE '** XMGR INVALID REQUEST (II) FROM XEXMN'
I1=REQUEST & NN IS LU

SET UP FOR FDS TERMINATION
EXIT TO :XMEND
ENDIF
ENDIF

ENDIF

ENDIF

IF DUMP OPTION THEN
CALL XDUMP TO DUMP OUR PARTITION
ENDIF
RELEASE DWA TRACK SPACE
FREE EXEC'S & PROCESSOR'S CLASS NUMBERS
CALL RNRQ SET LOCK ON STATUS TABLE
UNLOCK THE MANAGER FROM THE PARTITION
CALL SLIBR DISABLE
CLEAR EXEC'S ID SEGMENT
CALCULATE EQT ADDRESS FOR ATTENTION ID
CLEAR THE STATUS TABLE ENTRY
CALL SLIBR ENABLE
CALL RNRQ CLEAR THE LOCK ON THE STATUS TABLE
ISSUE MESSAGE '***XMOZ SIGN OFF FOR LU "NN"'
CALL SLIBR GET PRIVILEGED AND DISABLED
CALL SLIST TO MAKE US DORMANT
CLEAR OUR OWN ID
RESET EQT TO REMOVE ATTENTION ID
EXIT :SIEEQ TO THE RTE DISPATCHER

END  XMG
1  *D0  CALLING PROCEDURE
90 1  *D0  JSB  XMXGT
91 1  *D0  XMXGT
92 1  *D0  XMXGT
93 1  *D0  XMXGT
94 1  *D1  SEQUENCE TABLE EXECUTION FROM &SEQTB
95 1  *D1  XMXGT
96 1  *D1  XMXGT
97 1  *D2  INPUT
98 1  *D2  XMXGT
99 1  *D2  XMXGT
100 1  *D2  XMXGT
101 1  *D3  OUTPUT
102 1  *D3  XMXGT
103 1  *D3  XMXGT
104 1  *D3  XMXGT
105 1  *D3  XMXGT
106 1  *D5  Routines used
107 1  *D5  XMXGT
108 1  *D5  XMXGT
109 1  *D5  XMXGT
110 1  *D5  XMXGT
111 1  *D5  XMXGT
112 1  *D5  XMXGT
113 1  *D5  XMXGT
114 1  *D5  XMXGT
115 1  *D5  XMXGT
116 1  *D5  XMXGT
117 1  *D5  XMXGT
118 1  *D5  XMXGT
BEGIN XMXFT
CALL XMFTM TO FIND $SEGTO TOC ENTRY
FIND ADDRESS OF $SEGTO
CALL XMSTT TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
CALL XMSTT TO CONVERT STARTING SEQUENCE NUMBER INTO CURRENT DISPLACEMENT
COMPUTE CURRENT ENTRY ADDRESS
DO UNTIL THE LAST TABLE ENTRY IS EXECUTED OR
UNTIL THE TERMINATION ENTRY IS EXECUTED
IF PROCESSOR REQUIRE'S AN INTERFACE TABLE (WORD 3 BIT 8 IS SET) THEN
EXIT TO :ERROR: IF INTERFACE TABLE NOT SPECIFIED (WORD 4 = 0) (PARMS = 1)
CALL XMSTT TO SEARCH AWA FOR INTERFACE TABLE (CHAIN 4)
EXIT TO :ERROR: IF TABLE NOT FOUND (PARMS = 2)
IF TABLE NOT IN AWA, THEN
CALL INRT TO RETRIEVE FROM AWA
EXIT TO :ERROR: IF SPACE IS NOT FOUND
ENDIF
EXIT TO :ERROR: IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PARMS = 3)
EXIT TO :ERROR: IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PARMS = 4)
EXIT TO :ERROR: IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PARMS = 5)
ENDIF
SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED
EXIT TO :ERROR: IF NOT FOUND
CALL SLIBR TO BE PRIVILEGED
SET CURRENT TASK TO THIS PROCESSOR
CALL SLIBR TO BE UN-PRIVILEGED
IF PROCESSOR REQUIRE'S AN INTERFACE TABLE THEN
CALCULATE LENGTH OF INTERFACE TABLE HEADER AND SPEC'S
CALL EXEC TO CLASS 1/0 WRITE HEADER AND SPEC'S
ENDIF
CALL EXEC TO SCHEDULE PROCESSOR WITH WAIT
CALL XASCH TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XMFTM TO FIND $SEGTO TOC ENTRY
FIND ADDRESS OF $SEGTO
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARM = 1)
CALL XMARK TO HONOR AWA REQUEST
ENDDO
CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
EXIT TO :ERROR: IF PROCESSOR REQUESTED TERMINATION (PARM1 = 8)
EXIT TO :ERROR: IF PROCESSOR ABENDED (PARM1 = -32768)
IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PARM1 = 3) THEN
EXIT TO :ERROR: IF RESET NUMBER IS ZERO
CALL XMSTT TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY DISPLACEMENT
EXIT TO :ERROR: IF RESET SEQUENCE NUMBER IS NOT FOUR DIGIT
IF TERMINAL ENTRY WAS JUST EXECUTED THEN
SET UP PARM'S TO SHOW RESET SEQUENCE NUMBER
PERFORM :END: ==No Return Expected==
ENDIF
ELSE IF REQUEST IS NOT NORMAL COMPLETION (PARM1 = 0) THEN
DISPLAY ERROR MESSAGE - INVALID REQUEST
EXIT TO :ERROR: IF CURRENT AT IS DORMANT
CALL XMKIL TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAM TO RESCHEDULE PROCESSOR
EXIT TO :ERROR: TO TERMINATE SEQUENCE
ENDIF
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDDO
171 2 SET PARM1 = 0 (NORMAL COMPLETION)
172 2 PERFORM :END; - NO RETURN EXPECTED
173 2 :ERROR1: SET PARM1 = 1
174 2 :ERROR5: SET PARM1 = 8 AND PARM5 TO APPROPRIATE REASON CODE
175 2 :END;
176 2 CALL SLIBR TO BECOME PRIVLEDGED
177 2 SET CURRENT TASK IN PRM, STATUS TABLE, AND ANA TO EXEC
178 2 CALL SLIBX TO BECOME UN-PRIVLEDGED
179 2 CALL XPAR TO POST EXEC AND WAIT FOR NEXT REQUEST
180 1 END XMXT
CALLING PROCEDURE

JSB XMAFR
DEF **3
DEF ADDR
DEF SIZE

*********
PLACE A FE ON THE FE CHAINS AND MERGE WITH ANY ADJACENT FES

*********
INPUT

ADDR - ADDRESS OF AREA BEING FREED
SIZE - SIZE OF AREA BEING FREED. IF LESS THAN 3 THE FOLLOWING
WORK(S) WILL ALSO BE FREED SUCH THAT THE MINIMUM FE SIZE
OF 3 WORDS IS MAINTAINED.

*********
EXTERNAL SYMBOLS FROM XMAWA
XMBCP, XMFCP, XMFC, XMFRE

*********
OUTPUT (EXTERNAL SYMBOLS FROM XMAWA)
XMBCP, XMFCP, XMFC, XMFRE

*********
NOTES

USES .ENTR

**********
213 1 BEGIN XMAFR
214 2 SET NEW FE SIZE FIELD TO MAX(SIZE, 3)
215 2 INCREMENT TOTAL FREE SPACE BY NEW FE SIZE
216 2 INDEX TO FORWARD CHAIN POINTER (FCP) HEAD
217 2 START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)
218 3 EXIT IF 'ADDR' < FCP VALUE
219 3 SET NEW FE FCP TO CURRENT FCP VALUE
220 3 SET CURRENT FCP VALUE TO 'ADDR'
221 3 SET NEW FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE
222 3 SET NEXT FE'S BCP VALUE TO 'ADDR'
223 3 PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE
224 3 OR ELSE
225 3 INDEX TO NEXT FE FCP
226 3 END LOOP
227 3 SET NEW FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
228 3 SET CURRENT FE'S FCP VALUE TO 'ADDR'
229 3 SET NEW FE BCP TO BCP HEAD VALUE
230 3 SET BCP HEAD TO 'ADDR'
231 3 END SEARCH
232 3 IF NEW FE BCP NOT = END-OF-CHAIN
233 3 THEN
234 3 PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
235 3 END IF
236 3 IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW
237 3 (OR CONSOLIDATED) FE, THEN
238 3 MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
239 3 ENDIF
240 1 END XMAFR
241 1 BEGIN MERGE
242 2 IF FE 1 IS ADJACENT TO FE 2
243 2 THEN
244 3 INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
245 3 SET FE 1 FCP TO VALUE OF FE 2 FCP
246 3 IF FE 1 FCP NOT = END-OF-CHAIN
247 3 THEN
248 4 SET FE 3 BCP TO ADDRESS OF FE 1
249 4 ELSE
250 4 SET BCP HEAD TO ADDRESS OF FE 1
251 4 ENDIF
252 2 ENDIF
253 1 END MERGE
CALLING PROCEDURE

255 1 000
256 1 000 CALLING PROCEDURE
257 1 000 JNB XMAGT
258 1 000 DEF ++3
259 1 000 DEF OPTM
260 1 000 DEF SIZE
261 1 000
262 1 000
263 1 001 FIND A BLOCK OF FREE SPACE IN THE ANA AT LEAST 'SIZE' WORDS LARGE
264 1 001
265 1 001
266 1 000
267 1 002 INPUT
268 1 002 OPTM - INDICATOR OF WHICH FREE CHAIN TO SEARCH
269 1 002 O = FORWARD POINTER CHAIN (FOR TDC SPACE)
270 1 002 1 = BACKWARD POINTER CHAIN (FOR DATA SPACE)
271 1 002 SIZE - NUMBER OF WORDS NEEDED (A MINIMUM OF 3 WORDS WILL BE
272 1 002 ALLOCATED EVEN IF 'SIZE' IS 1 OR 2)
273 1 002
274 1 002 EXTERNAL SYMBOLS FROM XWMA
275 1 002 XMFC, XMFC, XMFR
276 1 000
277 1 000
278 1 003 OUTPUT
279 1 003 A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
280 1 003 INDICATING NONE AVAILABLE
281 1 003
282 1 003 EXTERNAL SYMBOLS FROM XWMA
283 1 003 XMFC, XMFC, XMFR
284 1 000 XMFC, XMFC
285 1 000
286 1 000
287 1 005 NOTES
288 1 005 IF SOME FE EXACTLY 'SIZE' WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
289 1 005 ERROR RETURN (A-REG = -32768) IS TAKEN
290 1 005
291 1 005 USES .EMIR
292 1 005
293 1 000
1 BEGIN XMSRC
2 SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FEMHEAD(OPTH)
3 START SEARCH WHILE POINT NOT = END-OF-CHAIN (-32768), AND
4 WHILE TOC SPACE FENCE HAS NOT BEEN CROSSED
5 EXIT IF FE SIZE = MAX('SIZE', 3)
6 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
7 DECHAIN FE
8 RETURN ADDRESS OF AREA
9 EXIT IF FE SIZE >= MAX('SIZE', 3) + 3
10 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
11 IF ALLOCATING FROM HEAD OF SPACE (OPTH = 0)
12 THEN
13 CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
14 RECHAIN NEW FE
15 RETURN ADDRESS OF AREA
16 ELSE
17 CHANGE SIZE FIELD TO FE SIZE = MAX('SIZE', 3)
18 COMPUTE AND RETURN ADDRESS OF AREA
19 ENDIF
20 OR ELSE
21 INDEX TO NEXT FE
22 END LOOP
23 SET RETURN CODE TO D(NOT FOUND)
24 END SEARCH
25 IF TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
26 INCREMENT TOC SPACE FENCE OR 'SIZE'
27 ENDF
28 END XMSRC
CALLING PROCEDURE

1 *00 JSR XMANG
1 *00 DEF CLUNO
1 ************
1 *01 PROVIDE AMA MANAGEMENT BASED ON REQUEST LIST (SFE FDS SDD
1 *01 TABLE 6.2 - III)
1 *01 ************
1 *02 INPUT
1 *02 CLUNO - CLASS I/O NUMBER CONTAINING REQUEST LIST
1 *02 ************
1 *02 REQUEST LIST (SEE TABLE 6.2 - III)
1 *02 ************
1 *03 OUTPUT
1 *03 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
1 *03 ************
1 *03 REQUEST LIST FIELD 8
1 ************
1 *05 NOTES
1 *05 USES EXEC,XMAFR,XMAGT,XMAPK,XMNK
1 *05 XMAIN,XMDAL,XMODA,XMODS,XMDP
1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AMA MANAGEMENT REQUESTS
1 *05 FOR DWA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DWA
1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
1 *05 THE DWA AND COPY THEM BACK TO THE AMA AS NECESSARY.
380 1 BEGIN XNAVG
381 2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER
382 3 CLEAR RETURN PARM1
383 4 CLEAR RETURN PARM2
384 5 CLEAR REWRITE FLAG
385 6 INITIALIZE TO FIRST REQUEST CODE
386 7 DO UNTIL END OF LIST (D, EIGHT REQUESTS PROCESSED OR PARM1 > ZERO
387 8 IF REQUEST FOR TOC (CODE 10)
388 9 THEN
389 10 IF REQUESTED SIZE > .5 X TOC SIZE, THEN
390 11 CALL XNAVG TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
391 12 ENDIF
392 13 WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O
393 14 STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT
394 15 SET REQUESTED SIZE FROM TOC SIZE
395 16 SET REWRITE FLAG
396 17 ELSE
397 18 IF REQUEST TO CLEAR (CODE 17)
398 19 THEN
399 20 GET A(SDMA) FROM XMDMA
400 21 SAVE HEADER AND DIRECTORY-SIZE
401 22 CLEAR XMDMA THRU XMDA
402 23 BUILD AN FE AT XMDA FOR AW'SIZE
403 24 CALL XNAVG TO ALTERATE A TOC ENTRY FOR SDMA
404 25 CHAIN IN TOC ENTRY TO XMDA
405 26 CALL XNAVG TO ALTERATE SPACE FOR SDMA
406 27 SET LOCATION, SIZE, & KEY IN THE TOC
407 28 SET DIRECT ENTRY ADDRESS AT XMDA
408 29 CLEAR THE DIRECTORY
409 30 SET LP, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY
410 31 ELSE
411 32 CALL XNFTN TO SEARCH TOC FOR INDICATED ENTRY
412 33 CASE (:VERIFY; :VERALO, :VERALO, :RENAME, :DELIVER, :DELIVER, :STORE),
413 34 (:RETRIEVE, :RETRIEVE) REQUEST CODE
414 35 :VERIFY:
415 36 IF ENTRY NOT FOUND
416 37 THEN
417 38 SET RETURN PARM1 AND PARM2 (2 & INDEX)
418 39 ENDIF
420 36 :VERALO:
421 37 IF ENTRY ALREADY EXISTS
422 38 THEN
423 39 IF ALLOCATE REQUEST (33)
424 40 THEN
425 41 SET RETURN PARM1 AND PARM2 (3 & INDEX)
426 42 ELSE
427 43 IF TYPE, SIZE AND 1-DIM FIELDS DO NOT MATCH
428 44 THEN
429 45 SET RETURN PARM1 AND PARM2 (4 & INDEX)
430 46 ENDIF
431 47 ELSE
432 48 CALL XNAVG TO ALLOCATE TOC SPACE
433 49 IF CLASS EQ 3 OR 8, THEN
434 50 CHAIN IN NEW TOC ENTRY
435 51 SET DATA SPACE ADDRESS TO ZERO
436 52 ELSE
437 53 CALL XNAVG TO ALLOCATE DATA SPACE
437  8  IF SPACE NOT AVAILABLE
        THEN
        SET RETURN PARM1 AND PARM2 (1 & INDEX)
        ELSE
          IF DATA ELEMENT (CLASS 2)
            THEN
              IF CHARACTER STRING (TYPE 4 - 8)
                THEN
                  INITIALIZE AREA TO BLANKS
                ELSE
                  INITIALIZE AREA TO ZEROS
                ENDIF
              ENDIF
            ELSE
              CHAIN IN NEW TOC ENTRY
            ENDIF
          ENDIF
        ENDIF
  ENDIF
457  6  :RENAME:
        IF ENTRY NOT FOUND
          THEN
            SET RETURN PARM1 AND PARM2 (2 & INDEX)
          ELSE
            CALL XMTFN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION
            IF ENTRY FOUND
              THEN
                SET RETURN PARM1 AND PARM2 (3 & INDEX)
              ELSE
                CALL XMAGT TO ALLOCATE NEW TOC ENTRY
                IF SPACE NOT AVAILABLE
                  THEN
                    SET RETURN PARM1 AND PARM2 (1 & INDEX)
                  ELSE
                    COPY OLD ENTRY ATTRIBUTES INTO NEW ENTRY AND CHAIN IN TO TOC
                    DECHAIN OLD ENTRY
                    CALL XMFFR TO RETURN OLD ENTRY TOC SPACE TO FE POOL
                  ENDIF
                ENDIF
            ENDIF
          ENDIF
478  6  :DELIVER:
        IF ENTRY FOUND
          THEN
            GENERATE KEY 1 LESS THAN FOUND KEY
            CALL XMFTN FOR GENERATED KEY
            DECHAIN TOC ENTRY
            CALL XMFFR TO RETURN TOC ENTRY SPACE TO FE POOL
            CALL XMFR TO RETURN DATA SPACE TO FE POOL
            IF CLASS EQ 4 OR 6, THEN
              CALL XMDDA DMA DEALLOCATION
            ENDIF
          ELSE
            IF DELETE REQUEST (5)
              THEN
                SET RETURN PARM1 AND PARM2 (2 & INDEX)
              ENDIF
ENDIF

:STORE:
IF ENTRY NOT FOUND
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ELSE
IF (TOC TYPE > 0 AND INCONSISTENT WITH REQUEST TYPE) OR
DISPLACEMENT OR SPECIFIED SIZE < 0, OR
DISPLACEMENT + REQUESTED SIZE > ALLOCATED SIZE
THEN
SET RETURN PARM1 AND PARM2 (4 & INDEX)
ELSE
SET DATA FROM INDICATED CLASS I/O; STORE INTO AMA
FREE CLASS NUMBER
IF CLASS EQ 4 OR 6, THEN
CALL XMST  DNA STORE DATA
ENDIF
ENDIF

:RETRIEVE:
IF ENTRY NOT FOUND
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ELSE
IF VALUES REQUESTED (8)
THEN
IF (TOC-TYPE .NE. 0 AND .NE. REQUESTED-TYPE) OR
DISPLACEMENT OR SPECIFIED SIZE < 0, OR
DISPLACEMENT + SPECIFIED SIZE > ALLOCATED SIZE
THEN
SET RETURN PARM1 AND PARM2 (4 & INDEX)
ELSE
IF REQUESTED SIZE = ZERO
THEN
CALCULATE AMOUNT OF DATA TO RETRIEVE AS ACTUAL SIZE MINUS DISPLACEMENT
STORE COMPUTED SIZE IN REQUEST WORD SIX
ENDIF
IF CLASS EQ 4 OR 6 AND TOC ADDRESS EQ 0, THEN
THE ELEMENT EXISTS ONLY ON THE DNA
CALL XMSTT MOVE INTO DATA
IF NO SPACE THEN
SET RETURN PARM1 AND PARM2 TO(1, INDEX)
EXIT TO :XMREX
ENDIF
ENDIF
ENDIF
WRITE VALUES TO CLASS I/O
STORE TYPE IN LOW BYTE OF REQUEST WORD 1
STORE CLASS NUMBER IN REQUEST WORD 2
SET WRITE FLAG
ENDIF
ELSE
WRITE TOC ENTRY TO CLASS I/O
STORE CLASS NUMBER IN REQUEST WORD 2
SET WRITE FLAG
:XMREX
ENDIF
CALLING PROCEDURE

1  *D0 JSB XNTFN
2  *D0 DEF **2
3  *D0 DEF KEYS
4  *D0
5  *D1 EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'
6  *D1
7  *D1 **********
8  *D2 INPUT
9  *D2 KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC
10  *D2
11  *D2 EXTERNAL SYMBOLS FROM XWARA
12  *D2 XMHD
13  *D2
14  *D2 **********
15  *D3 OUTPUT
16  *D3 A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR
17  *D3 ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY
18  *D3 WHERE 'KEYS' COULD BE CHAINED IN
19  *D3
20  *D3 **********
21  *D5 NOTES
22  *D5 USES .ENTR
23  *D5
24  *D5 **********
25  *D6
26  *D6 BEGIN XNTFN
27  *D7 ISOLATE CLASS FROM KEY AND INDEX "A" A-PROPRIATE CHAIN HEAD
28  *D8 START SEARCH WHILE CHAIN POINTER +A) < END-OF-CHAIN (-32768)
29  *D9 COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY
30  *D0 EXIT IF MATCH
31  *D1 RETURN ADDRESS OF ENTRY
32  *D2 EXIT IF 'KEYS' < TOC ENTRY (EXPECTED ENTRY NOT IN CHAIN)
33  *D3 RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET
34  *D4 OR ELSE
35  *D5 INDEX TO NEXT TOC ENTRY ON CHAIN
36  *D6 END LOOP
37  *D7 RETURN ADDRESS OF LAST (PREVIOUS) ENTRY WITH INDIRECT BIT SET
38  *D8 END SEARCH
39  *D9 END XNTFN
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
649 1 BEGIN XMPPW
650 2 *00 ENTRY: JSB XMPPW OR CALL XMPPM
651 2 *02 DEF RETURN ADDRESS
652 2 *02 INPUTS: XMPPM HAS REPLY
653 2 *02 OUTPUTS: XMPPM HAS REQUEST,
654 2 *03 AND XVSTB IS UPDATED.
655 2 *03 DO UNTIL GOOD PARMs RECEIVED OR TOP AT TERMINATES
656 3 3 IF ABORT CURRENT FLAG CN(- STBLU) THEN
657 3 3 SET ABORT FLAG IN XMPPW
658 3 3 TURN OFF ABORT CURRENT FLAG
659 3 3 ENDIF
660 3 3 GET CURRENT TOP ASSOCIATED TASK(AT)
661 3 3 SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
662 3 3 JSB XVPPW
663 3 3 DEF *+3 RETURN
664 3 3 DFC 0 MANAGER CALL
665 3 3 DEF XMPPM PARM FIELD
666 3 3 THIS IS AN IMPLIED WAIT
667 3 3 :XMPCN GET CURRENT XVSTB ENTRY (XUSTA)
668 3 3 IF THERE HAS BEEN A CALL TO PRM (P1 FIELD IS NOT TOP AT) OR
669 4 4 ID SEGMENT IS DORMANT OR
670 4 4 ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
671 4 4 IF PARM IS NOT 0,3,8,9, OR -32768 THEN
672 5 5 SET PARM TO -32768
673 5 5 PRINT ERROR "INVALID REQUEST"
674 5 5 ELSE
675 5 5 SET PARM FIELD FROM MANAGER'S ID SEGMENT
676 5 5 ENDIF
677 5 5 ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PAM)
678 5 5 IF PARM NOT 1 OR 2 THEN
679 6 6 CALL XMPP TO ABORT TOP AT
680 6 6 PRINT ERROR "INVALID REQUEST"
681 6 6 ELSE
682 6 6 SET PARM FROM CURRENT ID SEGMENT
683 6 6 ENDIF
684 6 6 ENDDO
685 6 6 END XMPPW
CALLING PROCEDURE

FUNCTION

CALL TOC & DWA FUNCTIONS

OUTPUT

BREG D=COMPLETE
MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

XMTFN, XNAGT

BEGIN XMDIN

GET NUMBER OF DWA TRACKS FROM P5

CALCULATE SIZE OF EDWA (5+6N) N IS # OF TRACKS

CALL XMTFN (EDWA)

CALL XNAGT (D,B) TOC ENTRY FOR EDWA

CALL XNAGT (1,SIZE) DATA AREA FOR EDWA

INITIALIZE & CHAIN EDWA TOC ENTRY

SET # OF TRACKS FOR EDWA

CLEAR EDWA

CALL EXEC (DISC TRACK ALLOCATION)

IF DISC ADDRESS .EQ. -1 TRACKS NOT AVAILABLE

THEN

ISSUE MESSAGE '***XMDN "N" TRACKS NOT AVAILABLE'

ERROR RETURN

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DWA MANAGEMENT

ENDIF

END XMDIN

REPRODUCIBILITY OF THE ORIGINAL PAGE IS FRAIL
1 BEGIN XNDAL
2 * DMA ALLOCATION
3 * CALLING PROCEDURE
4 * JSB XNDAL
5 * FUNCTION
6 * ALLOCATE DMA SPACE
7 * INPUTS
8 * A(TOC ENTRY OF THE DMA ELEMENT)
9 * IN TREG
10 * OUTPUTS
11 * UPDATE TOC ENTRY FOR DMA
12 * RETURNS BREG= ZERO- ALLOCATION COMPLETE
13 * = PLUS- ERROR CONDITION
14 * NOT'S
15 * CALLS XNNX?, XNBDT
16 * IF (RDVA) NE. C, THEN
17 * SET START-ADDRESS TO FIRST TRACK WORD IN DMA DIRECTORY
18 * GET SIZE(IN WORDS FROM TOC ENTRY)
19 * SIZE(IN SECTORS) = (SIZE+63/64)
20 * DO FOR NUMBER OF TRACKS IN DMA OR DMA SIZE .GE. SIZE
21 * SET BITUM TO ZERO
22 * DO UNTIL DMA SIZE .GE. SITE OR BITUM .GE. 96
23 * CALL XRNX(0,BITUM,START-ADDRESS)
24 * STARTBIT =BITUM
25 * CALL XRNXB (1,BITUM,START-ADDRESS)
26 * DMA-SIZE=BITUM-STARTBIT
27 * ENO-D
28 * IF BITUM .GE. 96, THEN
29 * START-ADDRESS+START ADDRESS+1 TRACK ADDRESS
30 * ENDF
31 * END-DO
32 * IF D: C AREA FOUND, THEN
33 * START-A=+5" & BITUM DEFINES THE TRACK & SECTOR
34 * SET DISC ADDRESS & SIZE IN TOC ENTRY
35 * CALL XPDST (TOC-ENTRY)
36 * SET NORMAL RETURN
37 * ELSE
38 * :XNDAL
39 * ISSUE MESSAGE "***XNDS NO DMA SPACE REMAINING"
40 * SET ERROR RETURN
41 * ENDF
42 * ENDIF
43 * ENDJ
44 * END XNDAL
778 1 BEGIN XMDOA
779 2 = DNA DEALLOCATION
780 3 CALLING PROCEDURE
781 4 JSB XMDOA
782 5
783 6 FUNCTION
784 7 DELETE DNA ELEMENT WHICH CORRESPONDS
785 8 TO THE DNA ELEMENT
786 9
787 10 INPUTS
788 11 TOC ENTRY ADDRESS OF DNA ELEMENT
789 12 IN TREG
790 13
791 14 NOTES
792 15 USES XMBST
793 16
794 17 IF A(DW) .NE. 0, THEN
795 18 GET DISC ADDRESS
796 19 GET DATA SIZE
797 20 * FREE THE DISK AREA
798 21 CALL XMBST (TOC ENTRY)
799 22 EUDIF
800 1 END XMDOA
802 1 BEGIN XHOST
803 2 # DBA STORE DATA
804 2 #00 CALLING PROCEDURE
805 2 #00 JSB XHOST
806 2 #00
807 2 #01 FUNCTION
808 2 #01 UPDATE DBA ELEMENT WHICH CORRESPONDS
809 2 #02 TO THE ANA ELEMENT
810 2 #02 INPUTS
811 2 #02 TOC ENTRY ADDRESS OF ANA ELEMENT
812 2 #02 IN TREG
813 2 #02
814 2 #02 OUTPUTS
815 2 #03 UPDATE DBA ELEMENT ON DISC TRACKS
816 2 #03
817 2 #03 NOTES
818 2 #03 USES WRITE
819 2 #03
820 2 #05 IF A(EDUA) NE 0 THEN
821 2 #05 SET DISC ADDRESS FOR THE WRITE
822 2 #03 SET DATA ADDRESS FOR THE WRITE
823 2 #03 WRITE DATA
824 2 ENDIF
825 1 END XHOST
1 BEGIN XLOAD
2 * DQA RETRIEVE CALLING PROCEDURE
3 DSX XLOAD
4 FUNCTION
5 RETRIEVE DQA DATA INTO AQA
6 INPUTS
7 A(TOC ENTRY) IN YREG
8 OUTPUTS
9 ADDRESS OF DATA IN THE TOC IN XREG,
10 RETRIEVE SUCCESSFUL
11 MINUS, ERROR NO AQA DATA
12 NOTES
13 USES XHAGT, XMDMA, EXEC(READ)
14 IF NO DQA DIRECTORY, THEN
15 SET ERROR CODE -5
16 ELSE
17 CALL XHAGT, GET DATA SPACE
18 IF NO SPACE, THEN
19 SET ERROR CODE -1
20 ELSE
21 SET DATA ADDRESS IN TOC
22 GET DISC ADDRESS
23 READ DATA INTO AQA
24 SET RETURN CODE TO 0
25 ENDDIF
26 ENDDIF
27 ENDDIF
28 ENDDIF
BEGIN XMGST
CALLING PROCEDURE
JSB XMGST

FUNCTION
SET ALLOCATION & DEALLOCATION IN DWA DIRECTORY

INPUTS
AWA TOC ENTRY

OUTPUTS
UPDATES DWA DIRECTORY

GET TRACK-ADDRESS FROM TOC
SUBTRACT START OF TRACKS FROM DWA DIRECTORY FOR RELATIVE TRACK
DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)
QUOTIENT IS NUMBER OF RELATIVE WORDS
REMAINDER IS BIT-POSITION(BP)
WORD ADDRESS=DWA ADDRESS+RELATIVE TRACK+RELATIVE WORDS
SUBTRACT BIT-POSITION FROM 16 GIVING NBFW(NUMBER BITS IN FIRST WORD)
GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)
BITS-NBFW-REMAINING BITS(RB)
IF RB I.E. 0, THEN
SET NUMBER OF WORDS(NW) TO ZERO
SET NUMBER OF BITS LAST WORD(NBLW) TO ZERO
SET NBFW TO BITS
ELSE
DIVIDE RB BY 16
SET NW TO QUOTIENT
SET NBFW TO REMAINDER
ENDIF
LLOAD DATA POINTED TO BY WORD ADDRESS
ROTATE LEFT (BP+NBFW-1 BITS) SAVE BP BITS & POSITION NBFW BITS
EXCLUSIVE OR SIGN BIT (ON TO OFF; OFF TO ON)
IF NBFW .LT. 1, THEN
SHIFT RIGHT (ARITHMETIC) BY NBFW-1 PROPAGATE BITS
ENDIF
IF DATA IS NEGATIVE, THEN
SET FILL WORD TO -1
ELSE
SET FILL WORD TO 0
ENDIF
ROTATE RIGHT BY BP RESET SAVED BITS
STORE WHERE WORD-ADDRESS POINTS
DO WHILE NW .GT. 0
WORD-ADDRESS=WORD-ADDRESS+WORD-ADDRESS
SET FILL WORD INTO WHERE WORD-ADDRESS POINTS
ENDDO
IF NBFW .LT. 0, THEN
WORD-ADDRESS=WORD-ADDRESS+
LOAD DATA POINTED TO BY WORD-ADDRESS
IF NBFW .LT. 1, THEN
ROTATE LEFT NBFW-1 BITS
ENDIF
EXCLUSIVE OR SIGN BIT
IF NBFW .LT. 1, THEN
SHIFT RIGHT ARITHMETIC BY NBFW-1
ENDIF
STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00    CALLING PROCEDURE
2 *00    JSB XMAPK
2 *00    DEF OPTION
2 *00    DEF PHASE 1 FLAG
2 *00    GENERAL COLLAPSE INTERFACE
2 *00    OUTPUT
2 *03    RETURNS AREA ADDRESS FOR AREA FOUND
2 *03    RETURNS -32768 AREA NOT FOUND
2 *03    NOTES
2 *05    USES XMPK1,XMPK2,XMPK3,XMSC
2 *05    IF PHASE1 FLAG NOT SET, THEN
2 *05    CALL XMPK1 PURGE DWA ELEMENTS FROM AWA
2 *05    PERFORM XMRC(OPTN,SIZE)
2 *05    ELSE
2 *05    SET RETURN CODE TO NOT FOUND
2 *05    ENDIF
2 *05    IF RETURN CODE IS NOT FOUND, THEN
2 *05    IF OPTN=1(BACKWARD CHAIN), THEN
2 *05    CALL XMPK2 PACK TOC CHAIN
2 *05    CALL XMRC(OPTN,SIZE)
2 *05    ENDIF
2 *05    IF RETURN CODE IS NOT FOUND, THEN
2 *05    CALL XMPK3 PACK AWA DATA AREAS
2 *05    PERFORM XMRC(OPTN,SIZE)
2 *05    ENDIF
2 *05    ENDIF
2 *05    SET RETURN VALUE TO RETURN CODE
1 END XMAPK
BEGIN XMPK1
  CALLING PROCEDURE
  JSB XMPK1
  PHASE 1 OF COLLAPSE(PURGE DNA ELEMENTS)
  OUTPUT
  UPDATES ADDRESS FIELD IN TOC
  FOR EACH DNA DATA ELEMENT
  NOTES
  USES XMAFR,XMND4,XMND6,XMPK1
  CURRENTLY ONLY CLASS 4 & 6 ELEMENTS
  (INTERFACE TABLES & SEQUENCE TABLES)
  ARE DNA ELEMENTS.
  INCREMENT XMPK1 UPDATE PHASE 1 COUNT
  IF DNA DIRECTORY ADDRESS NOT EQ 0, THEN
  SAVE X & Y REGS
  DO FOR ALL CLASS 4 ELEMENTS
  CALL XMAFR(DATA,SIZE)
  ENDBB
  DO FOR ALL CLASS 4 ELEMENTS
  CALL XMAFR(DATA,SIZE)
  ENDBB
  RESTORE X & Y REGS
  ENDLF
END XMPK1
990 1 BEGIN XMPK2
991 2 *00 CALLING PROCEDURE
992 2 *00 JSB XMPK2
993 2 *00
994 2 *01 PHASE 2 OF COLLAPSE(COLLAPSE TOC ENTRIES)
995 2 *01
996 2 *01 OUTPUT
997 2 *03 REORDERS TOC ENTRIES
998 2 *03 UPDATES XMFC( TOC FENCE ADDRESS)
999 2 *03 NOTES
1000 2 *05USES XMFC,XMFCP,XMTFM,XMBCP,XMAFR,XMPK2
1001 2 *05
1002 2 *05 INCREMENT XMMP2 UPDATE PHASE 2 COUNT
1003 2 *05 DO WHILE XMFC .GT. XMFCP AND XMFCP .NE. -32768
1004 3 * ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
1005 3 * WILL A TOC COMPRESS BE DONE.
1006 3 GET FIRST-FREE(FREE) FROM XMFC
1007 3 GET JLDZ FROM THE FREE ELEMENT
1008 3 NWSIZ=OLDZ-8
1009 3 IF NWSIZ .LT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
1010 4 GET NEXT-FREE FROM FIRST-FREE'S FCP
1011 4 SET INTO XMFC DELETE FROM THE FORWARD CHAIN
1012 4 IF XMFCP .NE. -32768, THE: IF NOT THE LAST FREE ELEMENT
1013 5 SET NEXT-FREE'S BCP TO -32768
1014 5 ELSE
1015 5 SET XMCP TO -32768 DELETE FROM BACKWARD CHAIN:
1016 8 ENDF
1017 8 ELSE
1018 8 ENDIF
1019 3 ELSE
1020 4 OLDZ=NEWZ UPDATE LENGTH IN FREE ELEMENT
1021 3 ENDIF
1022 3 MXTOC=(FIRSTF)+NWSIZ THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
1023 3 OLTOC=XMFC-8 OLD TOC IS THE ENTRY MOVE THE FENCE
1024 3 TMPKEY=OLTOC'S KEY - 1 CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
1025 3 CALL XMTFM(TMPKEY) FIND PREVIOUS
1026 3 SET PRTOC FROM ARG
1027 3 COPY TOC ENTRY FROM OLTOC TO MXTOC
1028 3 STORE MXTOC ADDRESS INTO PRTOC'S CHAIN
1029 3 CALL XMAFR(OLTOC,TOC-SIZE) FREE THE OLD TOC ENTRY
1030 2 ENDB
1031 1 END XMPK2
BEGIN XMPK3
CALLING PROCEDURE
JSB XMPK3
FUNCTION
COMPRESS AWA DATA AREA BY MIGRATING DATA AREAS
TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
OUTPUT
UPDATES XMBCP,XMFCP,XMFNC,XMFRE,XMPF3
NOTES
USES XMAFR
INCREMENT XMPK3 (NUMBER OF PHASES 3 or 4)
IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
CALL XMPK2, INQUIRE TOC IS COMPRESSED
DO WHILE XMBCP .GT. XMFCNC
UNTIL 1 FREE AREA ADJACENT TO THE FENCE
IF XCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
SET HIGH-WATER TO XMFCNC
ELSE
MULTIPLE AREAS
SET HIGH-WATER TO BCP OF LAST-FREE
ENDIF
SET LOW-WATER TO A(LAST-FREE)
PERFORM XMTCC(LOW-WATER,HIGH-WATER)
EXIT IF TOC-ADDRESS .EQ. 0
PERFORM XMANV(TOC-ADDRESS)
ENDIF
END XMPK3
1064 1 BEGIN XMTSC  TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR O(NONE FOUND).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XMAWA, THE
1070 2 * LAST TOC-ENTRY IS AT XMFNC-8.
1071 2 TOC-ENTRY=XMAWA)
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 DO UNTIL TOC-ENTRY .GE. XMFNC
1074 3 IF DATA ADDRESS IN TOC-ENTRY Z:
1075 4 ME 0, AND IS
1076 4 LT LOW-WATER, AND IS
1077 4 GT HIGH-WATER, AND IS
1078 4 GT TEST-AD,
1079 3 THEN
1080 4 TEST-AD=DATA ADDRESS
1081 4 TEST-TOC=TOC-ENTRY
1082 3 ENDF
1083 3 ADD 8 TO TOC-ENT
1084 2 ENDDO
1085 2 TOC-ADDRESS=TEST-TOC RETURN 0 OR A TOC ADDRESS
1086 1 END XMTSC
1088 1 BEGIN XMANV  AWA MOVE
1089 2 * MOVE THE DATA DEFINED BY THE TOC(WHICH IS
1090 2 * IMMEDIATELY ABOVE THE LAST FREE AREA) INTO
1091 2 * THE BOTTOM OF THE LAST FREE
1092 2 * UPDATE THE LENGTH OF THE RESULTING FREE AREA.
1093 2 GET DATA ADDRESS FROM THE TOC
1094 2 SAVE THE FIRST THREE WORDS OF THE DATA AREA
1095 2 CALL XMANV(DATA ADDRESS,SIZE)
1096 2 GET FREE AREA FROM XMBEP
1097 2 DECREASE FREE AREA LENGTH BT MAX(DATA SIZE,3)
1098 2 CALCULATE NEW ADDRESS FROM FREE AREA + FREE LENGTH
1099 2 MOVE DATA FROM DATA ADDRESS TO NEW ADDRESS
1100 2 MOVE SAVED FIRST THREE WORDS TO NEW ADDRESS
1101 2 UPDATE DATA ADDRESS IN TOC WITH NEW ADDRESS
1102 1 END XMANV
FORTRAN CALLING PROCEDURE

CALL XPAR (LU, INBUF, INTLNG, MBUFF, IMUN, NAME, TYPE, SIZE, IDIM, DSPT)

XPAR ALLOWS PROCESSORS TO OBTAIN THE ATTRIBUTES OF THE
PARAMETERS REFERENCED BY THE INTERFACE TABLE.

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INBUF - INPUT/OUTPUT BUFFER OF 7*(8 PARAMETERS + 1) WORDS.

ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.

INTLNG - LENGTH OF INBUF

MBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACCROSS PROCESSOR SERVICE CALLS.

IMUN - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE WHOSE
ATTRIBUTES ARE REQUESTED.

OUTPUT

NAME - AN ALPHANUMERIC NAME UP TO SIX CHARACTERS WHICH
IDENTIFIES THE DATA ELEMENT OR DDE WHERE THE DATA IS TO
BE OBTAINED/STORED. A ZERO ENTRY INDICATES INPUT DATA
WHICH IS LITERAL DATA STORED WITHIN THE INTERFACE TABLE.

TYPE - DATA TYPE CODE OF THE PARAMETER

SIZE - TOTAL NUMBER OF WORDS OF LITERAL DATA.

TOTAL NUMBER OF WORDS OF REFERENCED INPUT DE,
TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DE IF
SUBSCRIPTED OR ZERO IF NOT SUBSCRIPTED,
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DE OR
ZERO IF OUTPUT DDE

IDIM - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DDE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIPTED OUTPUT.

DSPT - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIPTED DATA ELEMENTS ELSE ZERO.

FOR AN INPUT DDE THE RTE FILE MANAGER TYPE CODE IS
RETURNED IN DSPT.

EXTERNAL SYMBOLS

(See XPAR)

INTERNAL VARIABLES

NOTES

USES XPAR, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR

XPAR IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
POINTS XPAR, XPAR AND XPAR
1 BEGIN XPATHR
2 PERFORM XPMI(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
3 EXIT TO :XPC:13 (XPGET) IF PARAMETER IS OUT OF RANGE
4 EXTRACT NAME FROM INTERFACE TABLE
5 SET DSPTT TO ZERO
6 IF LITERAL PARAMETER (NAME IS ZERO)
7 THEN
8 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
9 ELSE
10 IF SUBSCRIPTED (INTERFACE TABLE DISP OR S FIELDS ARE NON-ZERO)
11 THEN
12 PERFORM XPARC(XPGET) TO RETRIEVE TQC ENTRY AND COMPUTE DISPLACEMENT
13 STORE TYPE, SIZE, IDIM AND DSPTT
14 ELSE
15 IF IDNT
16 THEN
17 PERFORM XPARM(XPGET) TO QUALIFY FILE NAME
18 ENDIF
19 IF INPUT PARAMETER
20 THEN
21 CALL XPARQ TO RETRIEVE TQC ENTRY
22 COPY TYPE, SIZE, IDIM AND DSPTT FROM TQC ENTRY
23 ELSE
24 SET TYPE, SIZE AND IDIM TO ZERO
25 ENDIF
26 ENDIF
27 ENDIF
28 TEND XPATHR
155 104 PROCESSING LOOP
156 104 MP - VALUE OF "P", WHERE P IS THE NUMBER OF PARAMETERS IN THE
157 104 INTERFACE TABLE
158 104 NAMEFL - ARRAY OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
159 104 ARRAY REFT
160 104 NARY - NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
161 104 SEQUENCE
162 104 REFT - EIGHT WORD ARRAY USED FOR CONSTRUCTING ANA MANAGEMENT
163 104 REQUESTS FOR XPREG
164 104 USRID - USER FILE IDENTIFIER CHARACTER (=IPARM(3))
165 104 EXTERNAL VARIABLES (SEE XPREG)
166 104 XPLCS
167 104 XPLU
168 104 XPRG
169 104 XPRQ
170 104 NOTES
171 104 USES ENTR, EXEC, XPREG, XPRIT, XVSTO
172 105 XPGET MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.
173 105 XPLCS INSTRUMENT TO BE USED BY XPREG, XPPUT, AND XPATH AND XFDOS
174 105 TO BE INITIALIZED ONLY ONCE BY ANY OF THE THREE ROUTINES.
175 105 SINCE REQUESTS FOR INPUT DATA FROM THE ANA MAY BE MADE FOR UP TO
176 105 EIGHT AT A TIME, XPREG RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
177 105 REQUESTED IN MULTIPLES OF EIGHT.
FORTRAN CALLING PROCEDURE

CALL XFFMT (LU, INBUF, INTLIN, NBUFF, N, IMUXS, OUT(I), ..., OUT(N))

**********

XFFMT ALLOWS PROCESSORS TO STORE DATA INTO DATA ELEMENTS AND

date's referenced in the interface table as follows:

1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC.

IF FOUND, DATA IS STORED

IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING

TERMINATES.

2) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC. IF THE NAME IS

NOT FOUND, IT IS ADDDED TO THE TOC. IF THE NAME IS

FOUND OR AFTER BEING ADDED TO THE TOC, THE NAME IS

PREFFRED WITH A / SYMBOL, SUFFIFRED WITH A ONE

CHARACTER USER CODE AND RETURNED SO THE CALLING

PROGRAM CAN STORE DATA.

**********

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INBUF - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 13) WORDS.

INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON

FIRST USE TO CAUSE INITIALIZATION.

INTLIN - LENGTH OF INBUFF

NBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE

WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH

AREA BUT NOT ACROSS PROCESSOR SERVICES CALLS.

N - NUMBER OF PARAMETERS TO BE STORED. IF N=0, ALL

OUTPUT PARAMETERS ARE ASSUMED TO BE SUPPLIED.

IMUXS - ARRAY (N WORDS) OF RELATIVE NUMBER OF PARAMETERS IN THE

INTERFACE TABLE REFERENCING DATA ELEMENTS AND IDRE'S

WHERE THE OUTPUT DATA IS TO BE STORED. THE ORDER OF THE

OUT(N) - LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE

STORED FROM.

SHARED EXTERNAL SYMBOL

XPSTF - FLAG INDICATING XPSTF/XFFMT NORMAL PROCESSING (+1) OR

XPSTI/XFSTI SPECIAL PROCESSING BY-PASSING SUBSCRIPT

RESOLUTION (0-10)

**********

OUTPUT

NONE

**********

INTERNAL VARIABLES

(SEE XPGET)

**********

MOVES

(SEE XPGET)

**********
243 1 BEGIN XPGET
244 2 SET FOR 'GET'
245 2 PERFORM ACCESS TO RETRIEVE DATA
246 1 END XPGET
247 1 BEGIN XPPUT
248 2 SET FOR 'PUT'
249 2 PERFORM ACCESS TO STORE DATA
250 1 END XPPUT
251 1 BEGIN XPGET
252 2 PERFORM SPIN TO INITIALIZEGLOBALS AND INTERFACE TABLE
253 2 DO FOR EACH PARAMETER REQUESTED
254 3 IF SELECTED PARAMETER IS OR 'OR' OF RANGE
255 4 CALL XPRED TO PURGE QUEUED REQUESTS
256 4 EXIT TO :XPE13:
257 5 ENDIF
258 5 IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/'PUT' PROCESSING
259 5 THEN
260 5 CALL XPRED TO PURGE QUEUED REQUESTS
261 5 EXIT TO :XERR12:
262 5 ENDIF
263 5 IF JER RUNNINg CALLING SEQUENCE
264 5 THEN
265 6 CALL XPRED TO PURGE QUEUED REQUESTS
266 6 EXIT TO :XPE13:
267 6 ENDIF
268 6 IF PROCESSING FOR 'GET'
269 6 THEN
270 7 IF PARAMETER IS MEMORY RESIDENT OF (CLASS 2)
271 7 THEN
272 8 IF INPUT IN LITERAL FORM
273 9 THEN
274 10 BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT
275 10 ELSE IF NORMAL XPGET/PUT PROCESSING (XPGPF = -1)
276 11 THEN
277 12 IF SUBSCRIBED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
278 13 THEN
279 14 PERFORM XPSB TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
280 15 ELSE
281 16 DISPLACEMENT IS ZERO
282 17 ENDIF
283 18 ELSE
284 19 USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
285 20 ENDIF
286 21 BUILD REQUEST WITH DE NAME AND DISPLACEMENT
287 22 ENDIF
288 22 ENDIF
289 22 ENDIF
290 22 CALL XPRED TO QUEUE RETRIEVAL AND STORAGE OF INPUT
291 22 ELSE PARAMETER IS DROE (CLASS 3)
292 23 THEN
293 24 BUILD REQUEST FOR TOC ENTRY
294 25 CALL XPRED TO IMMEDIATELY RETRIEVE TOC ENTRY
295 25 PERFORM XPSB TO CONSTRUCT AND STORE QUALIFIED FILE NAME
296 25 'STORE FILE ATtributes'
297 26 ENDIF
298 26 ELSE PROCESSING FOR 'PUT'
299 27 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
300 27 THEN
301 28 IF NORMAL XPGET/PUT PROCESSING (XPGPF = -1)
IF SUBSCRIPTED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRDE
CALL XPREQ TO QUEUE ALLOCATION
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AWA
CALL XPREQ TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS ORDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD RF ST TO REALLOCATE DRDE WITH NEW ATTRIBUTES
EXIT TO :ERR 14: IF FILE TYPE NOT 1-13, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
CALL XPREQ TO QUEUE REALLOCATION OF DRDE
ENDIF
ENDIF
ENDDO
CALL XPREQ TO COMPLETE QUEUED REQUESTS
EXIT ACCESS
:ERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:ERR13: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:ERR14: CALL XPREQ TO PURGE QUEUED REQUESTS
TERMINATE PROCESSOR FOR INVALID DRDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
BEGIN XPNI
  INITIALIZE GLOBAL VALUES FROM LV AND XVSTB
  IF INTERFACE TABLE BUFFER NOT INITIALIZED
  THEN
    RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER
    IF RETRIEVAL NOT SUCCESSFUL
    THEN
      TERMINATE PROCESSOR WITH 'XP10 PROCESSOR INITIALIZATION ERROR'
    ENDIF
  ENDIF
  EXIT TO XPE13: IF N < 0
END XPNI

BEGIN XPQFN
  SET / IN FIRST CHARACTER POSITION
  MOVE FOUR WORD NAME INTO MIDDLE POSITIONS
  LOCATE FIRST BLACK CHARACTER
  REPLACE BLANK WITH USER ID
END XPQFN

BEGIN XP9C
  IF DOUBLE SUBSCRIPTED
    THEN
      BUILD REQUEST FOR SUBSCRIPTS STORED IN LITERAL BLOCK
      CALL XP9ER TO QUEUE RETURN OF SUBSCRIPTS
      ELSE
      SINGLE SUBSCRIPT IS CONTAINED IN DISPLACEMENT FIELD & J-SUBSCRIPT IS = 1
      ENDIF
  BUILD REQUEST TO RETURN TOC ENTRY
  CALL XP9ER TO IMMEDIATELY RETRIEVE TOC ENTRY
  IF DOUBLE DIMENSIONED (IDIM > 0)
  THEN
  DISPLACEMENT = (IDIM*(JSUB-1) + JSUB-1) # WORDS PER ELEMENT
  ELSE
  DISPLACEMENT = SIZE*(JSUB-1) + (JSUB-1) # WORDS PER ELEMENT
  ENDIF
END XP9C
END ACCESS
**FORTRAN CALLING PROCEDURE**

1 *00  CALL XPGET (LU, INBUF, INTNLG, MRBUFF, INUM, IN, SIZE, DISP)

1 *01  *00

1 *00  **INPUT**
1 *01  LU - LOGICAL UNIT NUMBER OF USER TERMINAL
1 *02  INBUF - INPUT/OUTPUT BUFFER OF *(# PARAMETERS + 1) WORDS,
1 *02  ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
1 *02  INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
1 *02  FIRST USE TO CAUSE INITIALIZATION
1 *02  INTNLG - LENGTH OF INBUF
1 *02  MRBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
1 *02  WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
1 *02  THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS
1 *02  INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM
1 *02  WHICH TO RETURN DATA
1 *02  SIZE - TOTAL NUMBER OF WORDS TO BE RETURNED
1 *02  DISP - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER
1 *02  AT WHICH TO BEGIN DATA RETRIEVAL. A VALUE OF ZERO
1 *02  INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER
1 *02  IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT
1 *02  BE SPECIFIC, I.E., RETRIEVAL FROM AN AREA PREVIOUS TO
1 *02  THE USER'S SUBSCRIPTS IS NOT SUPPORTED

1 *03  OUTPUT
1 *03  IN - ARRAY OF AT LEAST SIZE WORDS INTO WHICH INPUT DATA IS TO
1 *03  BE STORED

1 *04  *01

1 *04  **INTERNAL**
1 *04  BSDSP - DISPLACEMENT FROM THE REAL ORIGIN OF THE DATA ELEMENT AS
1 *04  A RESULT OF USER SUBSCRIPTING. DISP IS ADDED TO THIS
1 *04  VALUE IN THE REQUEST TO THE MANAGER
1 *04  GTP - DISPLACEMENT INTO GTP DEPENDING ON WHETHER GET (D) OR
1 *04  PUT (I) PROCESSING IS INVOLVED
1 *04  JGTP - TWO WORD VECTOR CONTAINING 'JSB XPGET' OR 'JSB XPPUT'
1 *04  INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGET AND
1 *04  XPPUT CALLS

1 *04  NOTES
1 *05  USES EXEC. ,ENTR, XPATR, XPE13(XPATR), XPGET(XPATR), XPPUT(XPATR)
1 *05  USES XPTR, XPATR, XPERR(XPATR), XPE13(XPATR), XPGET(XPATR),
1 *05  XPPUT(XPATR)
**FORTRAN CALLING PROCEDURE**

1. **CALL XPGTI (LU, INTBUF, INTLNG, MRBUFF, INUM, OUT, SIZE, DISP)**

1. **INPUT**
   1. **LU** - LOGICAL UNIT NUMBER OF USER TERMINAL
   1. **INTBUF** - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 1) WORDS.
   1. **INTLNG** - INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON FIRST USE TO CAUSE INITIALIZATION
   1. **MRBUFF** - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS
   1. **INUM** - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE INTO WHICH TO STORE DATA
   1. **OUT** - ARRAY OF AT LEAST SIZE WORDS FROM WHICH OUTPUT DATA IS TO BE TAKEN
   1. **SIZE** - TOTAL NUMBER OF WORDS TO BE STORED
   1. **DISP** - DISPLACEMENT FROM THE BEGINNING OF THE OUTPUT PARAMETER AT WHICH TO BEGIN DATA STORAGE. A VALUE OF ZERO INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT BE SPECIFIED. I.E., STORAGE INTO AN AREA PREVIOUS TO THE USERS SUBSCRIPTS IS NOT SUPPORTED.

1. **OUTPUT**
1. **NONE**
1. **INTERNAL**
1. **SEE XPGTI**
1. **NOTES**
1. **XPGTI IS AN ENTRY POINT INTO XPGTI**
1 BEGIN XPGTI
2 SET FOR GET PROCESSING
3 PERFORM XPPXI TO RETRIEVE DATA
4 END XPGTI
5 BEGIN XPPTI
6 SET FOR PUT PROCESSING
7 PERFORM XPPXI TO STORE DATA
8 END XPPTI
9 BEGIN XPITI
10 SET XPSPFX(PSEP*) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
11 CALL XPATR TO ASSURE INITIALIZATION OF INBUF AND RETURN BASE DISPLACEMENT
12 EXIT TO :XPE13; (XPATH) IF INDICATED PARAMETER IS A DROG FILE
13 SAVE INTERFACE TABLE ENTRY CLASS/TYPE WORD, FLAG/DISP WORD AND SIZE WORD
14 IF LITERAL (NAME = D) THEN
15 EXIT TO :ERR15; IF DISP + SIZE > SAVED SIZE
16 ELSE
17 INCREMENT BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
18 SET TYPE FIELD TO FREE
19 2 EK0IF
20 CLEAR ENTRY SUBSCRIPT BIT
21 SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
22 SET SIZE FIELD TO SIZE
23 CALL XPGET/XPPUT TO TRANSFER DATA
24 RESTORE ORIGINAL INTERFACE TABLE ENTRY
25 RESTORE XFGPF(XPGET) TO NOMINAL VALUE
26 EXIT XPPXI
27 2 :ERR15: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
28 1 END XPPXI
CALLING PROCEDURE

JSRB XPREQ

DEF **+1 WHERE M IS THE NUMBER OF ACTUAL ARGUMENTS

DEF OPTN REQUIRED OPTION

DEF RQST REQUIRED REQUEST

DEF ADRES OPTIONAL ADDRESS (SEE OPTN)

**********

XPGET/XPPUT BUFFERED AWANAGEMENT SERVICE

**********

IM-UT

OPTN - OPTION WORD

SIGN BIT - O QUEUE REQUEST AND RETURN

1 QUEUE REQUEST AND CLOSE BUFFER

RIGHT BIT - O QUEUE REQUEST AND TRANSFER DATA TO/FROM

ADRES

1 QUEUE REQUEST ONLY

RQST - EIGHT WORD REQUEST (SEE SDD 6.2.6.2)

ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT VALUES USED ONLY WHEN OPTN(RIGHT BIT) = O)

**********

EXTERNAL SYMBOLS

XPLU - TERMINAL LOGICAL UNIT NUMBER

XPROB - ADDRESS OF 64 WORD MANAGER REQUEST BUFFER

XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

**********

OUTPUT

IF A REQUEST FAILS, A MESSAGE IS ISSUED AND PROCESSING IS TERMINATED, OTHERWISE INCOMING DATA WILL BE STORED IN THE ADDRESSES SUPPLIED.

**********

LOCAL VARIABLES

ADDS - ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDS).

ADDS - TABLE OF ADDRESSES FOR STORAGE OF Fetched DATA

APRMS - ADDRESS OF PARMS ARRAY

CLASS = CLASS NUMBER USED TO TRANSFER DATA TO MANAGER

PARMS - PARAMETER RETURN AND SCRATCH AREA

PTR - SAVE AREA FOR ADDS AND XPROB POINTER

**********

NOTES

USES ENTR, EXEC, XPLU, XPAW

IF DATA IS TO BE TRANSFERRED TO/FROM 'ADRES', THEN 'RQST' SIZE FLD (WORD 6) MUST BE SET.

**********
543 1 BEGIN XPREQ
544 2 MOVE REQUEST INTO BUFFER
545 2 IF OPTION IS TO TRANSFER DATA
546 3 THEN
547 3 IF REQUEST IS TO RETRIEVE DATA (8)
548 3 THEN
549 4 STORE ADDRESS IN TABLE
550 4 ELSE SHOULD BE A REQUEST TO STORE DATA (7)
551 4 OUTPUT DATA TO CLASS I/O
552 4 STORE CLASS NUMBER IN REQUEST WORD 8
553 3 ENDIF
554 3 ENDIF
555 2 INCREMENT POINTER
556 2 IF BUFFER FULL OR OPTION IS TO CLOSE NON-EMPTY BUFFER
557 3 THEN
558 3 CLOSE BUFFER
559 3 TRANSMIT BUFFER TO MANAGER
560 3 PAN MANAGER WITH REQUEST FOR AMA MANAGEMENT
561 3 RETRIEVE RETURN PARAMETERS
562 3 IF REWRITE FLAG SET (PARM5)
563 3 THEN
564 4 RETRIEVE REQUEST BUFFER
565 3 ENDIF
566 3 IF REQUESTS WERE SUCCESSFUL
567 3 THEN
568 4 DO FOR EACH REQUEST IN BUFFER
569 5 IF REQUEST TO RETURN DATA (8, 9 OR 16)
570 5 THEN
571 6 RETRIEVE AND STORE DATA IN ADDRESS CONTAINED IN TABLE
572 6 ENDIF
573 4 ENDDO
574 3 CLEAR POINTER AND LOCAL CLASS NUMBER
575 3 ELSE
576 4 OUTPUT FAILURE MESSAGE (XP11)
577 4 DO FOR EACH REQUEST IN BUFFER
578 5 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
579 5 THEN
580 6 FREE CLASS I/O NUMBER AND SAM BUFFER
581 5 ENDIF
582 4 ENDDO
583 3 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
584 3 ENDIF
585 2 ENDIF
586 1 END XPREQ
FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE

CALL XPDRS (LU,PRNLEN,PROMPT,TYPE,DATLEN,IDIM,DATA,RETCS)

CALLS XPDRS, DECODES COMMUNICATIONS BUFFER LOOKING FOR A SPECIFIC
"TYPE" AND STORES IT IN "DATA" FOR "DATLEN" LOGICAL ELEMENTS

INPUTS FROM CALLING SEQUENCE:

LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
PRNLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
STRING USED FOR THE USER PROMPT
PROMPT - (INTEGER, PRNLEN WORDS) IS THE CHARACTER STRING
USED AS THE USER PROMPT
TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE
EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
0 - FREE
1 - INTEGER
2 - REAL
3 - DOUBLE PRECISION
4 - CHARACTER STRING LENGTH 2
5 - CHARACTER STRING LENGTH 6
6 - CHARACTER STRING LENGTH 18
7 - CHARACTER STRING LENGTH 36
8 - CHARACTER STRING LENGTH 72

DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN
THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE
NUMBER OF WORDS.

IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA
AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0

IF IT IS A VECTOR.

OUTPUTS FROM CALLING SEQUENCE:

DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE)
AREA TO CONTAIN THE USERS RESPONSE
RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO
CALLER:
0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
1 - USER ENTERED SPACE CONTENTS OF BUFFER UNPREDICTABLE.
2 - USER ENTERED A CR, THERE IS NO RESPONSE.
3 - USER PROMPT WAS TOO LONG, MAXIMUM LENGTH IS
34 CHARACTERS OR 17 WORDS,
4 - PARAMETER LIST IS INVALID, EITHER:
A.IDIM LESS THAN ZERO
B.DATLEN LESS THAN ONE
C.TYPE VALID TYPE SPECIFIED.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>667</td>
<td>DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT</td>
</tr>
<tr>
<td>668</td>
<td>ELEMENT IS TO BE STORED</td>
</tr>
<tr>
<td>669</td>
<td>EFFTYP - THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER</td>
</tr>
<tr>
<td>670</td>
<td>STRING RESPONDER. IF TYPE IS FREE, IT IS THE</td>
</tr>
<tr>
<td>671</td>
<td>CLOSEST SUPPORTED LENGTH, ELSE IT IS THE LENGTH</td>
</tr>
<tr>
<td>672</td>
<td>SPECIFIED BY TYPE.</td>
</tr>
<tr>
<td>673</td>
<td>IND - CONTAINS THE INDEX INTO MSGS ARRAY OF THE CURRENT</td>
</tr>
<tr>
<td>674</td>
<td>ERROR MESSAGE.</td>
</tr>
<tr>
<td>675</td>
<td>LEN - LENGTH OF CURRENT ERROR MESSAGE.</td>
</tr>
<tr>
<td>676</td>
<td>LENGTH - 9 WORD ARRAY REPRESENTING THE LENGTH IN WORDS</td>
</tr>
<tr>
<td>677</td>
<td>OF THE 9 DATA TYPES 0-8 RESPECTIVELY</td>
</tr>
<tr>
<td>678</td>
<td>SKIND - COUNT OF NUMBER OF NESTED REPEATS AND INDEX</td>
</tr>
<tr>
<td>679</td>
<td>TO THE CURRENT TOP OF THE STACK</td>
</tr>
<tr>
<td>680</td>
<td>STKPRM - A WORD ARRAY FOR STACK OF PARENTHESES FLAGS</td>
</tr>
<tr>
<td>681</td>
<td>STKREP - A WORD ARRAY FOR STACK OF REPEAT COUNTS</td>
</tr>
<tr>
<td>682</td>
<td>STKTOK - A WORD ARRAY FOR STACK OF TOKEN POINTERS</td>
</tr>
<tr>
<td>683</td>
<td>WHERE EACH REPEAT GROUP BEGINS</td>
</tr>
<tr>
<td>684</td>
<td>TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED</td>
</tr>
<tr>
<td>685</td>
<td>TYPE1 - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE</td>
</tr>
<tr>
<td>686</td>
<td>TYPE1 = TYPE +1</td>
</tr>
<tr>
<td>687</td>
<td><strong>SUBROUTINES AND FUNCTIONS CALLED:</strong></td>
</tr>
<tr>
<td>688</td>
<td>EXEC, XPRDS, ERRMSG</td>
</tr>
<tr>
<td>689</td>
<td>POL ROUTINES INCLUDED:</td>
</tr>
<tr>
<td>690</td>
<td>XPRDS, STRING, SUBSCR, REPET, ERRMSG</td>
</tr>
</tbody>
</table>
1 BEGIN XPDRS
2 IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3 CALL XPDRS TO PROMPT USER AND RECEIVE RESPONSE
4 DO UNTIL EOS TOKEN IS FOUND
5 CASE TOKEN (:NUMBER; :NUMBER; :NUMBER; :STRING; :NULL;)
6 ;SUBSCR; ;REPET:
7 ;NUMBER: ERREXIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERMAG
8 ;MESSAGE NUMBER TO XPO2
9 ;ERREXIT IF THERE IS EMPTY SPACE IN DATA AREA PERFORM ERMAG
10 ;SET PREVIOUS TOKEN TO DATA
11 ;CALL XSMOV TO MOVE DATA INTO DATA AREA
12 ;INCREMENT POINTER IN DATA AREA
13 ;INCREMENT POINTER TO NEXT TOKEN
14 ;STRING: PERFORM STRING
15 ;NUMBER: PERFORM STRING
16 ;MESSAGE NUMBER TO XPO2
17 ;ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERMAG
18 ;SET PREVIOUS TOKEN TO DATA
19 ;INCREMENT POINTER IN DATA AREA
20 ;INCREMENT POINTER TO NEXT TOKEN
21 ;SUBSCR: PERFORM SUBSCR
22 ;REPET: PERFORM REPET
23 ENDCASE
24 DO UNTIL TOKEN IS NOT A CLOSED PARENTHESIS
25 IF STACK IS NOT EMPTY AND
26 (PREVIOUS TOKEN IS DATA AND TOP OF STACK PAREN FLG = 0) OR
27 (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLG = 1) THEN
28 DECREMENT TOP OF STACK REPEAT COUNT BY 1
29 IF TOP OF STACK REPEAT COUNT > 0 THEN
30 RESET TOKEN POINTER TO TOP OF STACK INDEX
31 ELSE
32 POP TOP ENTRY ON STACK
33 IF TOKEN IS A CLOSED PAREN THEN
34 INCREMENT TO NEXT TOKEN
35 ENDFi
36 ENDFi
37 END00
38 SET XPDRS RETURN CODE = XPDRS RETURN CODE
39 ENDIF
40 SET XPDRS RETURN CODE TO 515, INVALED PARAMETER LIST
41 ENDFi
42 ENDO0
43 ENDFi
44 ENDO0
45 ENDFi
46 SET XPDRS RETURN CODE = XPDRS RETURN CODE
47 ENDFi
48 ELBS
49 SET XPDRS RETURN CODE TO 515, INVALED PARAMETER LIST
50 ENDFi
51 ENDO XPDRS
BEGIN STRING
2 ERREXT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERRMSG
4 DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
6 ERREXT IF TYPE OF RESPONSE > TYPE REQUESTED AND
8 IF TYPE IS NOT FREE THEN
10 SET EFFECTIVE LENGTH = LENGTH REQUESTED
END IF
SET MESSAGE NUMBER TO XPOZ
2 ERREXT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERRMSG
4 CALL XNOV TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH
6 CALL XNOV TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH
8 SET PREVIOUS TOKEN IS DATA
2 INCREMENT POINTER IN DATA AREA
2 INCREMENT TO NEXT TOKEN
END STRING
1*
1*
1*
1*
BEGIN SUBSCR
2 INCREMENT POINTER TO NEXT TKCN
4 SET MESSAGE NUMBER TO XPOZ
6 ERREXT IF TKCN IS NOT AN INTEGER TO PERFORM ERRMSG
8 IF I-DIMENSION > 1 THEN
10 SET I TO INTEGER VALUE
12 INCREMENT POINTER TO NEXT TKCN
14 ERREXT IF TOKEN IS NOT AN INTEGER OR
16 ERREXT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
18 SET MESSAGE NUMBER TO XPOZ
20 ERREXT IF INTEGER > I-DIMENSION TO PERFORM ERRMSG
22 ERREXT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERRMSG
24 ELSE
26 ERREXT IF NEXT TOKEN IS NO; CLOSE PAREN TO PERFORM ERRMSG
28 SET MESSAGE NUMBER TO XPOZ
30 ERREXT IF SUBSCRIPT IS OUT OF RANGE
32 END IF
2 ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT
4 INCREMENT POINTER BY 3 TKCN
6 SET PREVIOUS TOKEN = SUBSCRIPT
8 SET MESSAGE NUMBER TO XPOZ
10 ERREXT IF TOKEN IS AN EOS OR
12 ERREXT IF TOKEN IS A REPEAT OR
14 ERREXT IF TKCN IS A CLOSE PAREN OR
16 ERREXT IF TOKEN IS A SUBSCRIPT TO PERFORM ERRMSG
END SUBSCR
END STRING
1 BEGIN REPET
2 SET MESSAGE NUMBER TO XPOS
3 ERREXIT IF TOKEN IS AN EOS TO PERFORM ERRMSG
4 INCREMENT STACK POINTER
5 SET MESSAGE NUMBER TO XPOS
6 ERREXIT IF THERE ARE TOO MANY NESTED REPEATS PERFORM ERRMSG
7 PUSH REPEAT COUNT ON STACK
8 SET PARENTHESES FLAG TO 0
9 IF TOKEN IS AN OPEN PARENTHESES THEN
10 INCREMENT POINTER TO NEXT TOKEN
11 SET PARENTHESES FLAG TO 1
12 ENDIF
13 PUSH TOKEN INDEX AND PAREN FLAG ON STACK
14 SET PREVIOUS TOKEN IS A REPEAT
15 END REPET
16 1
17 1
18 1
19 1 BEGIN ERRMSG
20 2 CALL EXEC TO WRITE ERROR MESSAGE
21 2 PERFORM XPRDS - NO RETURN
22 1 END ERRMSG
**CD0***

**FORTRAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE**

**CALL XPRDM (LU,PRMLEN,PROMPT,COMLEN,COMBUF,RETC)**

**CD0***

**WRITE **"PRMLEN**" WORDS OF **"PROMPT"** TO USER READS THE USER'S**

**RESPONSE; CONVERTS RESPONSE TO TOKENS IN **"COMBU"** AND PASSES**

**BACK A RETURN CODE **"RETC"** INDICATING THE SUCCESS OF XPRDM**

**AND THE USEFULNESS OF **"COMBU"**.**

**CD0***

**INPUTS FROM CALLING SEQUENCE:**

**LU** - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL

**PRMLEN** - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER

**PROMPT** - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING

**USE AS THE USER PROMPT.**

**COMLEN** - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE

**COMMUNICATIONS BUFFER (COMBUF)**

**CD0***

**OUTPUTS FROM CALLING SEQUENCE:**

**COMBUF** - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE

**ENCODED USER RESPONSE**

**RETC** - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO THE

**CALLING PROGRAM AS FOLLOWS:**

**0** - NORMAL RETURN BUFFER CONTAINS USERS RESPONSE

**1** - USER ENTERED CR CONTENTS OF BUFFER UNPREDICTABLE.

**2** - USER ENTERED CR BUFFER CONTAINS NO INFORMATION

**3** - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS

**34 CHARACTERS OR 17 WORDS.**

**CD0***

**INTERNAL VARIABLES**

**COUNT** - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN

**A CHARACTER STRING AND NUMER DIGITS IN A NUMBER.**

**DOLLMT** - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE

**DOLVD** - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE

**VALUE**

**FLECOM** - COMMA FLAG

**Q** - LAST CHARACTER NOT A COMMA

**T** - LAST CHARACTER WAS A COMMA

**FLECOM** - CONTINUE FLAG

**Q** - THIS IS NOT A RESPONSE TO A CONTINUE

**T** - THIS IS A CONTINUED RESPONSE

**FLENUM** - NUMBER FLAG

**Q** - POSITIVE NUMBER

**T** - NEGATIVE NUMBER

**FLGPOW** - POWER FLAG

**Q** - POSITIVE POWER

**CD0***
1 BEGIN XPRDM
2 IF PROMPT IS NOT TOO LONG THEN
3 CALL XMOV "" MOVE PROMPT INTO OUTPUT AREA
4 CALL EXEC TO WRITE PROMPT
5 SET XPRDM RETURN CODE TO NORMAL RETURN
6 INITIALIZE COMMUNICATIONS BUFFER
7
8 :COMLOP:
9 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
10 CALL EXEC TO READ RESPONSE
11 CALL XRPK TO CONVERT A2 RESPONSE TO R1 FORMAT
12 IF NUMBER OF WORDS READ IS NOT ZERO THEN
13 SET COMMA FLAG ON
14 DO WHILE COMMA FLAG IS OFF
15 IF INPUT BUFFER IS COMPLETEDLY SCANNED THEN
16 SET COMMA ON
17 IF COMMA FLAG IS ON THEN
18 CALL EXEC TO WRITE CONTINUE
19 GO TO :COMLOP;
20 ELSE
21 ERROR IF COMUF IS FULL PERFORM COMFUL
22 IF INPUT CHARACTER IS A COMMA THEN
23 IF COMMA FLAG IS ON THEN
24 ERROR IF COMUF CANNOT HOLD TOKEN PERFORM COMFUL
25 STORE NULL FIELD TOKEN IN COMUF
26 INCREMENT #WORDS IN COMUF BY 1
27 INCREMENT #TOKENS IN COMUF BY 1
28 ENDIF
29 SET COMMA FLAG ON
30 SET NEXT INPUT CHARACTER
31 ELSE
32 SET COMMA FLAG OFF
33 PERFORM TOKENS
34 ENDIF
35 ENDIF
36 ELSE
37 IF CONTINUE FLAG IS OFF THEN
38 SET XPRDM RETURN CODE TO SAY USER ENTERED CR
39 ENDIF
40 ELSE
41 SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
42 ENDIF
43 END XPRDM
1 BEGIN TOKENS
2 SET NEGATIVE NUMBER FLAG OFF
3 SET NEGATIVE POWER FLAG OFF
4 SET POWER = 0
5 IF INPUT CHARACTER IS A DIGIT THEN
   PERFORM DIGIT
6 ELSE
8         :A:
9     SET XPROM RETURN CODE TO SAY I ENTERED
10    SET CONTINUE FLAG ON
11
12 :B:
13    PERFORM QUOTE
14
15 :C:
16    SET PAREN INDICATOR = 0
17    GO TO :E:
18
19 :D:
20    SET PAREN INDICATOR = 1
21    GO TO :E:
22
23 :E:
24    ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN TO PERFORM COMPUF
25    STORE TOKEN (OPAR + PAREN INDICATOR) IN COMBUF
26    INCREMENT #WORDS IN COMBUF BY 1
27    INCREMENT #TOKENS IN COMBUF BY 1
28    GET NEXT CHARACTER
29
30 :F:
31    ERREXIT IF NEXT CHARACTER IS NOT A DIGIT PERFORM INVAL
32    SET INTEGER = 0
33    PERFORM DECP
34
35 :G:
36    IF INPUT CHARACTER IS A - THEN
37    SET NEGATIVE NUMBER FLAG ON
38    ENDIF
39    GET NEXT CHARACTER
40    IF INPUT CHARACTER IS A DIGIT THEN
41    PERFORM DIGIT
42    ELSE
43        IF INPUT CHARACTER IS A . THEN
44           GO TO :F:
45        ELSE
46           PERFORM INVAL - NO RETURN
47    ENDIF
48    ENDIF
49    END CASE
1 BEGIN QUOTE
3 GET NEXT CHARACTER
5 SET #CHARACTERS = 0
7 DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
9 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
11 INPUT CHARACTER IS A QUOTE AND
13 NEXT CHARACTER IS A QUOTE AND
15 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
17 INCREMENT #CHARACTERS BY 1
19 MOVE CHARACTER INTO TEMPORARY BUFFER (#CHARACTERS)
21 IF INPUT CHARACTER IS A QUOTE THEN
23 GET NEXT CHARACTER
25 ENDIF
27 GET NEXT CHARACTER
29 ENDDO
33 ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
35 ERREXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
37 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
39 STORE CHARACTER STRING TOKEN IN COMBUF
41 STORE NUMBER OF CHARACTERS IN COMBUF
43 CALL XRPCK TO CONVERT CHARACTERS FROM R1 TO A2 FORMAT
45 INCREMENT #WORDS IN COMBUF BY 2*([#CHARACTERS + 1] /2)
47 INCREMENT #TOKENS IN COMBUF BY 1
49 GET NEXT CHARACTER
51 1 END QUOTE
1019 1 BEGIN DIGIT
1020 2 PERFORM DCOL
1021 3 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1022 4 IF INPUT CHARACTER IS A . THEN
1023 5 PERFORM DECPT
1024 6 ELSE
1025 7 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1026 8 PERFORM EORD
1027 9 ELSE
1028 10 IF INPUT CHARACTER IS AN "R" THEN
1029 11 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN
1030 12 PERFORM COMFUL
1031 13 ELSE
1032 14 STORE REPEAT TOKEN IN COMBUF
1033 15 INCREMENT #WORDS IN COMBUF BY 2
1034 16 GET NEXT CHARACTER
1035 17 ELSE
1036 18 PERFORM INTEGR
1037 19 ENDIF
1038 20 ENDIF
1039 21 ELSE
1040 22 PERFORM INTEGR
1041 23 ENDIF
1042 24 ENDIF
1043 25 END DIGIT
1044 26 1
1045 27 1
1046 28 BEGIN DCOL
1047 29 SET INTEGER = 0
1048 30 SET COUNTER = 0
1049 31 DO WHILE CHARACTER IS A DIGIT AND
1050 32 WHILE INPUT BUFFER IS NOT EXHAUSTED
1051 33 SET INTEGER = (INTEGER * 10) + INPUT CHARACTER - 48
1052 34 IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1053 35 INCREMENT COUNTER BY 1
1054 36 GET NEXT CHARACTER
1055 37 ENDOD
1056 38 ENDOD
1057 39 END DCOL

ORIGINAL PAGE 8
```
1059    1 BEGIN DECP
1060    2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061    2 GET NEXT CHARACTER
1062    2 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063    3   IF INPUT CHARACTER IS A DIGIT THEN
1064    4   PERFORM DCOL
1065    4   ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066    4   EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067    3   ENDIF
1068    3   IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069    4   PERFORM END
1070    3   ELSE
1071    4   PERFORM REAL
1072    3   ENDIF
1073    2   ELSE
1074    3   PERFORM REAL
1075    2   ENDIF
1076    1 END DECP
```
1078 1 BEGIN EORD
1079 2 IF INPUT CHARACTER IS AN "E" THEN
1080 3 SET TYPE FLAG TO "E"
1081 4 ELSE
1082 5 SET TYPE FLAG TO "D"
1083 6 ENDIF
1084 7 GET NEXT CHARACTER
1085 8 ERREXIT IF INPUT BUFFER IS EXHAUSTED PERFORM INVAL
1086 9 IF INPUT CHARACTER IS A - THEN
1087 10 SET NEGATIVE POWER FLAG ON
1088 11 GET NEXT CHARACTER
1089 12 ELSE
1090 13 IF INPUT CHARACTER IS A + THEN
1091 14 SET NEGATIVE POWER FLAG OFF
1092 15 GET NEXT CHARACTER
1093 16 ENDIF
1094 17 FDENDIF
1095 18 ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
1096 19 ERREXIT IF INPUT CHARACTER IS NOT A DIGIT PERFORM INVAL
1097 20 PERFORM DCOL
1098 21 IF NEGATIVE POWER FLAG IS ON THEN
1099 22 SET POWER = -POWER
1100 23 ENDIF
1101 24 IF TYPE FLAG IS "E" THEN
1102 25 PERFORM REAL
1103 26 ELSE
1104 27 BEGIN DBL
1105 28 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
1106 29 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
1107 30 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
1108 31 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1109 32 SET DOUBLE = DUABLE * 10 ** POWER
1110 33 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1111 34 IF NEGATIVE NUMBER FLAG IS ON THEN
1112 35 SET DOUBLE = -DOUBLE
1113 36 ENDIF
1114 37 STORE DOUBLE TOKEN IN COMBUF
1115 38 INCREMENT #WORDS IN COMBUF BY 4
1116 39 INCREMENT #TOKENS IN COMBUF BY 1
1117 40 END DBL
1118 41 ENDIF
1119 42 1 END EORD
1 BEGIN INTEGER
  2 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
  3 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREM AND
  4 ERREXIT IF INPUT BUFFEP IS NOT EXHAUSTED PERFORM INVAL
  5 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  6 CONVERT DOUBLE TO INTEGER
  7 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVERFLOW
  8 IF NEGATIVE NUMBER FLAG IS ON THEN
  9 SET INTEGER = INTEGER
END IF
 10 STORE INTEGER TOKEN IN COMBUF
 11 INCREMENT #WORDS IN COMBUF BY 2
 12 INCREMENT #TOKENS IN COMBUF BY 1
 13 END INTEGER
  1 *
  2 *
  3 *
  4 BEGIN REAL
  5 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
  6 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREM AND
  7 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
  8 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  9 SET REAL = DOUBLE * 10 ** POWER
 10 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVERFLOW
 11 IF NEGATIVE NUMBER FLAG IS ON THEN
 12 SET REAL = -REAL
END IF
 13 STORE TOKEN IN COMBUF
 14 INCREMENT #WORDS IN COMBUF BY 3
 15 INCREMENT #TOKENS IN COMBUF BY 1
 16 END REAL
1153 1 BEGIN INVAL
1154 2 CALL KCVT TO CONVERT OCTAL CHARACTER NUMBER TO ASCII
1155 2 CALL EXEC TO WRITE ERROR MESSAGE
1156 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1157 1 END INVAL
1158 1 *
1159 1 *
1160 1 *
1161 1 BEGIN COMFUL
1162 2 CALL EXEC TO WRITE ERROR MESSAGE
1163 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1164 1 END COMFUL
1165 1 *
1166 1 *
1167 1 *
1168 1 BEGIN OVFLOW
1169 2 CALL KCVT TO CONVERT OCTAL TO ASCII
1170 2 CALL EXEC TO WRITE ERROR MESSAGE
1171 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1172 1 END OVFLOW
FORTRAN CALLING PROCEDURE

130  CALL XPIT ((L), RPAMS)

131  C*********  
132  C*********  
133  C*********  
134  C*********  
135  C*********  
136  C*********  
137  C*********  
138  C*********  
139  C*********  
140  C*********  
141  C*********  
142  C*********  
143  C*********  
144  C*********  
145  C*********  
146  C*********  
147  C*********

148  LETERATE A PROGRAM AFTER WAITING ON ANY BUFFERED I/O TO LU TO
149  COMPLETE THEM PASS PARAMETERS BACK TO THE FATHER TASK.

150  C***********
151  C***********
152  C***********
153  C***********
154  C***********
155  C***********
156  C***********
157  C***********
158  C***********
159  C***********
160  C***********
161  C***********
162  C***********
163  C***********
164  C***********
165  C***********
166  C***********
167  C***********
168  C***********
169  C***********
170  C***********
171  C***********
172  C***********
173  C***********
174  C***********
175  C***********
176  C***********
177  C***********
178  C***********
179  C***********
180  C***********
181  C***********
182  C***********
183  C***********
184  C***********
185  C***********
186  C***********
187  C***********
188  C***********
189  C***********
190  C***********
191  C***********
192  C***********
193  C***********
194  C***********
195  C***********
196  C***********
197  C***********
198  C***********
199  C***********
200  C***********
201  C***********
202  C***********
203  C***********
204  C***********
205  C***********
206  C***********
207  C***********
208  C***********
209  C***********
210  C***********
211  C***********
212  C***********

1 BEGIN XPIT
2 IF (L) IS NON-ZERO
3 THEN
4 WAIT ON ANY CLASS I/O TO COMPLETE (CLASS I/O CONTROL THEN GET)
5 END IF
6 RETURN PARAMETERS TO FATHER
7 TERMINATE PROGRAM
8 END XPIT
FORTAN CALLING PROCEDURE

1 000 CALL XORBIT (BIT, BITNUM, STRING)
1 000

************
1 001 BIT SET/CLEAR ROUTINE FOR MULTIPLE WORD BIT STRINGS. SET THE
1 001 VALUE OF BIT NUMBER 'BITNUM' IN STRING 'STRING' TO 'BIT'.

************
1 002 INPUT
1 002 BIT - INTEGER VALUE OF ZERO OR ONE TO BE SET IN BIT NUMBER
1 002 'BITNUM' OF 'STRING'
1 002 BITNUM - INTEGER BIT NUMBER OF BIT TO BE SET/CLEARED (FIRST BIT OF
1 002 STRING IS BIT NUMBER ZERO)
1 002 STRING - BIT STRING OF AT LEAST 'BITNUM' BITS IN LENGTH INTO WHICH
1 002 'BIT' IS TO BE SET

************
1 003 OUTPUT
1 003 STRING - BIT NUMBER 'BITNUM' HAS THE VALUE OF 'BIT'

************
1 005 NOTES
1 005 USES .ENTR, XRSER
1 005

************
1 006 BEGIN XORBIT
2 007 TRANSFER CALLING SEQUENCE
2 007 COMPUTE ADDRESS OF WORD CONTAINING BITNUM
2 007 COMPUTE BIT NUMBER WITHIN WORD
2 007 CALL XRSER TO SET/CLEAR BIT

1 008 END XORBIT
INTEGER FUNCTION

XRCPR(LENGTH, ARRAY1, ARRAY2)

1 *** XRCPR
2
3 BEGIN XRCPR
4 1 TRANSFER CALLING SEQUENCE
5 2 SETUP COMPARE
6 2 COMPARE ARRAY1 AND ARRAY2
7 2 RETURN RESULT FLAG
8 1 END XRCPR
FORTRAN CALLING PROCEDURE.

CALL XR018 (DOUBLE, ASCII)

CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN
'TP018.1Y' FORMAT

INPUT

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

OUTPUT

ASCII - NINE CHARACTER STRING REPRESENTATION OF
'DOUBLE'

LOCAL

D - WORKING LOCATION FOR ABSOLUTE VALUE OR 'DOUBLE'

REPEATEDLY MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

USES DOUBLE, FLOAT, IAND, IDINT, IOR, KCVT, XREXT

BEGIN XR018

SET SIGN FIELD

MOVE ABSOLUTE VALUE OF 'DOUBLE' INTERNAL

COMPUTE EXPONENT

SET SIGN AND VALUE OF EXPONENT FIELD

REDUCE VALUE TO RANGE OF 1 < VALUE <10

EXTRACT FIRST DIGIT, MERGE WITH SIGN AND STORE FIELD

EXTRACT SECOND DIGIT, MERGE WITH DECIMAL AND STORE FIELD

DO FOR NEXT FIVE PAIRS OF DIGITS

MULTIPLY BY 100 TO EXTRACT PAIR

EXTRACT DIGITS AND STORE FIELD

END XR018
FORTRAN CALLING PROCEDURE

1 DO
2 DO
3 CALL XREQ
4 DO
5************
6 CD1 MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE
7 CD1************
8 CD2 COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPT
9 CD2 ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER
10 CD2************
11 CD3 INPUT
12 CD3 COMMON XE - REQPT
13 CD3 CLASS I/O WRITE/READ TO CLASS 'CLASNO'
14 CD3 REQBUF AND RESPONSE IS PRINTED IF REQUESTED
15 CD3************
16 CD5 COMMUNICATES WITH FDS MANAGER FATHER TASK
17 CD5 USES EXEC, IAND, XRMOV, XRMST, XR606, XRSFL, XDPL, XVPW
18 CD5************
19 BEGIN XREQ
20 PERFORM TRACE
21 OUTPUT REQUESTS TO MANAGER
22 REQUEST AWA MANAGEMENT AND WAIT FOR RESPONSE
23 RETRIEVE MANAGER RESPONSE
24 PERFORM TRACE
25 RETURN RESPONSE IN REQPT
26 END XREQ
27 BEGIN TRACE
28 IF TRACE REQUESTED THEN
29 DO FOR EACH REQUEST
30 CALL XDPL TO FORMAT LINE
31 OUTPUT LINE
32 ENDDO
33 OUTPUT PARM1 AND PARM2
34 ENDIF
35 END TRACE
1 #00  INTEGER FUNCTION
2 #00  XNEXT(START, LENGTH, SOURCE)
3 #00  ********
4 #01  EXTRACT 'LENGTH' BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
5 #01  AND RIGHT ADJUST
6 #01  ********
7 #02  INPUT
8 #02  START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
9 #02  (SIGN BIT = 0)
10 #02  LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
11 #02  SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED
12 #02  ********
13 #05  NOTES
14 #05  USES .ENTR
15 #05  ********
17 #0
18 #
19 #
20 #
21 #
22 BEGIN XNEXT
23 2 TRANSFER CALLING SEQUENCE
24 2 IF START NOT = 0
25 2 THEN
26 3 CONSTRUCT SHIFT
27 3 LOAD A WITH SOURCE
28 3 SHIFT BA LEFT START BITS
29 3 ELSE
30 3 LOAD A WITH SOURCE
31 2 ENDIF
32 2 SAVE A
33 2 CLEAR B
34 2 CONSTRUCT SHIFT
35 2 RESTORE A
36 2 SHIFT BA LEFT LENGTH BITS
37 2 MOVE RESULT FROM B TO A
38 1 END XNEXT
FORTRAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

CONVERT A SINGLE PRECISION REAL NUMBER TO AN ASCII STRING IN
1PE14.6 FORMAT

REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED

ASCII - SEVEN WORD ASCII CHARACTER STRING REPRESENTATION OF
'REAL'

LOCAL

WORKING LOCATION FOR ABSOLUTE VALUE OR 'REAL' REPEATEDLY
MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

USES FLOAT, IAND, IFIX, IOR, KCVT, XREXT, XRSFL

* *

BEGIN XRE14

SET SIGN FIELD

MOVE ABSOLUTE VALUE OF 'REAL' INTERNAL

COMPUTE EXPONENT

SET SIGN AND VALUE OF EXPONENT FIELD

REDUCE VALUE TO RANGE OF 1 <= VALUE < 10

EXTRACT FIRST DIGIT AND STORE FIELD

SET DEIMAL FIELD

DO FOR NEXT THREE PAIRS OF DIGITS

MULTIPLY BY 100 TO EXTRACT PAIR

EXTRACT DIGITS AND STORE FIELD

END XRE14
FORTRAN CALLING PROCEDURE

CALL XR16 (INTEGER, ASCII)

CONVERT A SIXTEEN BIT SIGNED BINARY INTEGER TO A SIX CHARACTER ASCII STRING

INPUT

INTEGER - SIXTEEN BIT INTEGER TO BE CONVERTED

OUTPUT

THREE WORD CHARACTER STRING REPRESENTATION OF 'INTEGER'

LOCAL

INTERNAL LOCATION FOR 'INTEGER' REPEATEDLY MODIFIED TO PRODUCE 'ASCII'

SEVEN WORD WORKING BUFFER FOR CONSTRUCTION OF 'ASCII'

NOTES

USES XRMOV AND XRCPK

BEGIN XR16

BLANK WORKING SPACE

CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING

SET SIGN OF 'INTEGER' IN 'ASCII'

CALL XRCPK TO CONVERT FROM R1 TO A2 FORMAT

END XR16
FORTRAN CALLING PROCEDURES

CALL XRLCK (RCODE)
CALL XRULK (RCODE)

XRLCK AND XRULK PROVIDE A MECHANISM FOR SERIALIZING THE UPDATE
OF FDS GLOBAL SYSTEM TABLES AND FILES. THE RESOURCE NUMBER
STORED IN THE XVSTB RESIDENT STATUS TABLE IS USED AS THE
LOCKING MECHANISM

INPUT

YVSTB RESOURCE NUMBER

OUTPUT

RCODE - INTEGER RETURN CODE (0 - SUCCESSFUL, 1 - FAILURE)

LOCAL

STAT - STATUS OF THIS COPY OF XEXEC USE OF XVSTB RN

1 - RN LOCKED

4 - RN UNLOCKED

NOTES

USES .ENTR, RNMR.

THIS ROUTINE MAY NOT BE OVERLAYED
1 BEGIN IXLCX
2 SET REQUEST FOR LOCK FUNCTION
3 PERFORM RLOCK
4 END IXLCX
5 BEGIN IXULK
6 SET REQUEST FOR UNLOCK FUNCTION
7 PERFORM RLOCK
8 END IXULK
9 BEGIN RLOCK
10 IF REQUEST IS CONSISTENT WITH STATUS
11 THEN
12 SET NEW STATUS
13 IF RM IN STB IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
14 THEN
15 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
16 ENDIF
17 CLEAR RETURN CODE
18 ELSE
19 SET RETURN CODE
20 ENDIF
21 END RLOCK
INTEGER FUNCTION

1. RETURN THE 16-BIT MAPPED ADDRESS OF A

INPUT:
1. A - VARIABLE, ROUTINE, ETC. FOR WHICH THE ADDRESS IS DESIRED

OUTPUT:
1. XRLOC - 16-BIT ADDRESS OF A

NOTES:
1. NO EXTERNAL REFERENCES

BEGIN XRLOC

1. TRANSFER CALLING SEQUENCE

2. LOAD THE ADDRESS OF THE CALLING PARAMETER

END XRLOC
389 1 *00  FORTRAN CALLING PROCEDURE
390 1 *00
391 1 *00 CALL XRMov (LEPSHN, SOURCE, OBJECT)
392 1 *00
393 1 *00
394 1 *01  MOVE 'LENGTH' WORDS FROM 'SOURCE' TO 'OBJECT'
395 1 *01
396 1 *01
397 1 *00
398 1 *00 INPUT
399 1 *02 LENGTH - POSITIVE INTEGER INDICATING NUMBER OF WORDS TO MOV.
400 1 *02 SOURCE - ARRAY OF WORDS TO BE MOVED
401 1 *02
402 1 *00
403 1 *00 OUTPUT
404 1 *03 OBJECT - ARRAY RECEIVING MOVED WORDS
405 1 *03
406 1 *00
407 1 *05 NOTES
408 1 *05 USES .ENTR
409 1 *05
410 1 *00
411 1 * 
412 1 * 
413 1 * 
414 1 * 
415 1 BEGIN XRMov
416 1 BEGIN TRANSFER CALLING SEQUENCE
417 2 INITIALIZE MOVE
418 2 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
419 1 END XRMov
FORTRAN CALLING PROCEDURE

CALL XRE4G(Number, Locate, Length, Source)

**C**

INSERT 'LENGTH' WORDS OF TEXT FROM 'SOURCE' INTO MESSAGE.

'NUMBER' BEGINNING AFTER 'LOCATE' WORDS, CONCATENATE TO PREFIX,

TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL

**C**

INPUT

NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE

A - AREA INDICATOR AS FOLLOWS

1 - AS
2 - XE
3 - XI
4 - XS
5 - YT
6 - XE
7 - XE
8 - XL
9 - BF
10 - SC

MN - MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'

WORDS OF 'SOURCE' TO BE OUTPUT

LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECED

'SOURCE' (NOT USED IF 'MN' OF 'NUMBER' IS ZERO)

LENGTH - INTEGER NUMBER OF WORDS OF 'SOURCE' TO BE INSERTED INTO

MESSAGE. ZERO INDICATES NO INSERTION

SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED

IF 'LENGTH' IS ZER0)

COMMON

LU - USERS LOGICAL UNIT NUMBER

**C**

OUTPUT

UP TO EIGHTY CHARACTER LINE OF TEXT TO UNIT 'LU' OF THE FORM

**C**

*** AANN MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE

**C**

**C**

**C**

**C**

**C**

**C**

**C**

**C**

**C**

USES FD$ SYSTEM MESSAGE FILE XRE4G

USES CLOSE, EXEC, IAMS, KCVT, OPEN, READ, XRE4G, XUDG

**C**
1 BEGIN XRMSE
2 SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
3 SET NUMBER IN PREFIX
4 READ MESSAGE DIRECTORY RECORD
5 IF AREA VALID
6 THEN
7 SET AREA CODE IN PREFIX
8 IF MESSAGE NUMBER > 0
9 THEN
10 IF VALID MESSAGE NUMBER
11 THEN
12 COMPUTE MESSAGE RECORD NUMBER
13 READ RECORD
14 CALL XRMV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
15 ELSE
16 EXIT TO :ERROR:
17 ENDIF
18 CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
19 IF MESSAGE NUMBER > 0
20 THEN
21 CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
22 ENDIF
23 ELSE
24 SET AREA IN PREFIX
25 :ERROR: CALL XRMV TO MOVE 'XRMSE ERROR' INTO BUFFER
26 CALL XRMV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
27 ENDIF
28 OUTPUT BUFFER TO USER'S TERMINAL
29 IF DEBUG IS REQUESTED
30 THEN
31 CALL XEXEC
32 ENDIF
33 END XRMSE
INTEGER FUNCTION

XBMIF (BIT, BITNUM, STRING)

* SEARCH BIT STRING 'STRING' BEGINNING AT BIT NUMBER 'BITNUM' FOR
  THE NEXT OCCURRENCE OF BIT SETTING 'BIT'

************

* INPUT
  BIT - INTEGER VALUE THE LAST BIT OF WHICH IS TO BE COMPARED TO
  BITS OF 'STRING' FOR A MATCH
  BITNUM - UNSIGNED SIXTEEN BIT INTEGER INDICATING THE BIT NUMBER IN
  'STRING' WITH WHICH TO BEGIN THE SEARCH (FIRST BIT OF
  'STRING' IS BIT NUMBER ZERO)
  STRING - BIT STRING TO BE SEARCHED. SEARCH WILL CONTINUE THROUGH
  MEMORY UNTIL A VALUE OF 'BIT' IS DETECTED

************

* OUTPUT
  *3 FUNCTION VALUE - BIT NUMBER OF NEXT OCCURRENCE OF 'BIT' >= 'BITNUM'

************

* NOTES
  *3 USES .ENTR

*3 SEARCH WILL NOT TERMINATE UNTIL A VALUE OF 'BIT' IS DETECTED OR
  *3 ALL OF MEMORY HAS BEEN EXAMINED. THEREFORE APPROPRIATE STEPS SHOULD
  *3 BE TAKEN TO FORCE A MATCH AT THE END OF THE STRING.

*3 A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4096 WORDS) CAN BE
  *3 MEANINGFULLY ACCOMMODATED.

************
1 BEGIN XNKB
2 COMPUTE STARTING WORD NUMBER
3 COMPUTE STARTING BIT NUMBER
4 LOAD STARTING WORD
5 SHIFT WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
6 INITIALIZE BIT COUNT
7 DO WHILE SIGN BIT IS NOT EQUAL TO BIT
8 IF WORD NOT FINISHED
9 THEN
10 SHIFT WORD LEFT
11 INCREMENT BIT COUNT
12 ELSE
13 DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
14 LOAD NEXT WORD
15 ENDDO
16 CLEAR BIT COUNT
17 ENDIF
18 ENDDO
19 RETURN VALUE OF MATCHING BIT NUMBER
20 END XNKB
BEGIN XRO6
CALL XRO6 (BINARY, OCTAL)

1 FUNCTION XRO6 (BINARY, OCTAL)
2 USES ENTR
3 ENTRY XRO6
4
5 integrated into the program
6
7 ORGANIZATION
8 DATA DEFINITION...
**FORTRAN CALLING PROCEDURE**

***606***

**606**

**1**

1 **606**

**606**

**1**

1 *00 CALL XRPK (LENGTH, UNPKED, PACKED)

**607**

**607**

**1**

1 *00 CALL XRPK (LENGTH, UNPKED, PACKED)

**608**

**608**

**1**

1 *00 CALL XRPK (LENGTH, UNPKED, PACKED)

**609**

**609**

**1**

1 *00 CALL XRPK (LENGTH, UNPKED, PACKED)

**610**

**610**

**1**

1 *********

**611**

**611**

**1**

1 *01 CONVERT LENGTH CHARACTERS OF UNPKED FROM R1 FORMAT TO A2

**612**

**612**

**1**

1 *01 CONVERT LENGTH CHARACTERS OF UNPKED FROM R1 FORMAT TO A2

**613**

**613**

**1**

1 *01 CONVERT LENGTH CHARACTERS OF UNPKED FROM R1 FORMAT TO A2

**614**

**614**

**1**

1 *01 CONVERT LENGTH CHARACTERS OF UNPKED FROM R1 FORMAT TO A2

**615**

**615**

**1**

1 *********

**616**

**616**

**1**

1 *02 INPUT

**617**

**617**

**1**

1 *02 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED

**618**

**618**

**1**

1 *02 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED

**619**

**619**

**1**

1 *02 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED

**620**

**620**

**1**

1 *********

**621**

**621**

**1**

1 *03 OUTPUT

**622**

**622**

**1**

1 *03 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS

**623**

**623**

**1**

1 *03 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS

**624**

**624**

**1**

1 *03 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS

**625**

**625**

**1**

1 *********

**626**

**626**

**1**

1 *05 NOTES

**627**

**627**

**1**

1 *05 NOTES

**628**

**628**

**1**

1 *05 NOTES

**629**

**629**

**1**

1 *********

**630**

**630**

**1**

1 *

**631**

**631**

**1**

1 *

**632**

**632**

**1**

1 *

**633**

**633**

**1**

1 BEGIN XRPK

**634**

**634**

**1**

1 BEGIN XRPK

**635**

**635**

**1**

1 TRANSFER CALLING SEQUENCE

**636**

**636**

**1**

1 TRANSFER CALLING SEQUENCE

**637**

**637**

**1**

1 SET BYTE FLAG FOR HIGH BYTE

**638**

**638**

**1**

1 INITIALIZE PACKED POINTER

**639**

**639**

**1**

1 DO FOR EACH CHARACTER IN UNPKED

**640**

**640**

**1**

1 IF BYTE FLAG SET HIGH

**641**

**641**

**1**

1 THEN

**642**

**642**

**1**

1 LOAD A WITH NEXT WORD OF UNPKED

**643**

**643**

**1**

1 SHIFT CHARACTER INTO HIGH BYTE

**644**

**644**

**1**

1 ELSE

**645**

**645**

**1**

1 INCLUSIVE OR NEXT WORD OF UNPKED INTO A

**646**

**646**

**1**

1 STORE A IN PACKED

**647**

**647**

**1**

1 INCREMENT POINTER

**648**

**648**

**1**

1 ENDIF

**649**

**649**

**1**

1 FLIP BYTE FLAG

**650**

**650**

**1**

1 ENDDO

**651**

**651**

**1**

1 IF BYTE FLAG SET LOW

**652**

**652**

**1**

1 THEN

**653**

**653**

**1**

1 INCLUSIVE OR BLANK INTO LOW BYTE

**654**

**654**

**1**

1 ENDF

**655**

**655**

**1**

1 STORE A IN PACKED

**656**

**656**

**1**

1 END XRPK
FORTRAN CALLING PROCEDURE

CALL XRFN (PREFIX, NAME4, NAME6)

**********

XRFN BUILDS A QUALIFIED FILE NAME OF UPTO SIX CHARACTERS IN
LENGTH BY PREFIXING THE INPUT ONE TO FOUR CHARACTER NAME WITH
THE PREFIX CHARACTER AND APPENDING A USER QUALIFIER CODE TO THE
END

**********

** INPUT

** PREFIX - FILE TYPE PREFIX STORED IN R1 FORMAT

** NAME4 - ONE TO FOUR CHARACTER PACKED NAME TO BE QUALIFIED

** COMMON XE - QUAL

**********

** OUTPUT

** NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME

**********

** NOTES

** USES .ENTR

**********

** BEGIN XRFN

** STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING

** MOVE NAMEA INTO NEXT FOUR POSITIONS

** STORE BLANK IN SIXTH POSITION

** LOCATE FIRST BLANK CHARACTER

** REPLACE BLANK WITH USER ID CHARACTER (QUAL)

** MOVE QUALIFIED NAME TO NAME6

END XRFN
FORTRAN CALLING PROCEDURE
CALL XRSET (START, LENGTH, SOURCE, OBJECT)

*********

REPLACE 'LENGTH' BITS OF 'OBJECT', BEGINNING WITH BIT 'START',
WITH THE RIGHT MOST 'LENGTH' BITS OF 'SOURCE'

*********

INPUT

START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE REPLACED
(SIGN BIT = 0)

LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED

SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED

OUTPUT

OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED

*********

NOTES

USES .ENTR

*********

BEGIN XRSET

TRANSFER CALLING SEQUENCE

CONSTRUCT SHIFT INSTRUCTIONS

SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER

SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD

CONSTRUCT MASK AND CLEAR FIELD OF OBJECT

INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT

END XRSET
INTEGER FUNCTIONS

1.60
1.60 XRSFL(COUNT, SOURCE)
1.60 XRSFR(COUNT, SOURCE)

1.60 BEGIN XRSFL
2. SET FOR LEFT SHIFT
3. PERFORM SHIFT (FLAG, COUNT, SOURCE)
4. END XRSFL

1.61 BEGIN XRSFR
2. SET FOR RIGHT SHIFT
3. PERFORM SHIFT (FLAG, COUNT, SOURCE)
4. END XRSFR

1.65 BEGIN SHIFT
2. TRANSFER CALLING SEQUENCE
3. CONSTRUCT SHIFT INSTRUCTION
4. LOAD A WITH SOURCE
5. CLEAR B
6. SHIFT BA AS SPECIFIED
7. END SHIFT
FORTRAN CALLING PROCEDURE

CALL XRUNG (NAME6, NAME4)

XRUNG REMOVES THE PREFIX AND SUFFIX QUALIFYING CHARACTERS FROM
A SIX CHARACTER FILE NAME

INPUT
THREE TO SIX CHARACTER PACKED QUALIFIED NAME

OUTPUT
ONE TO FOUR CHARACTER PACKED NAME WITH PREFIX AND SUFFIX REMOVED

NOTES
USES .ENTR

BEGIN XRUNG

MOVE CHARACTERS 2-5 OF NAME6 INTO NAME4

IF SIXTH CHARACTER IS BLANK, THEN
LOCATE LAST NON-BLANK CHARACTER OF NAME4
BLANK THAT CHARACTER
ENDIF

END XRUNG
<table>
<thead>
<tr>
<th>100</th>
<th>CALL TRUPE (LENGTH, PACKED, UNPKED, COUNT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>CALL TRUPE (LENGTH, PACKED, UNPKED, COUNT)</td>
</tr>
<tr>
<td>300</td>
<td>CALL TRUPE (LENGTH, PACKED, UNPKED, COUNT)</td>
</tr>
<tr>
<td>400</td>
<td>CALL TRUPE (LENGTH, PACKED, UNPKED, COUNT)</td>
</tr>
<tr>
<td>500</td>
<td>CALL TRUPE (LENGTH, PACKED, UNPKED, COUNT)</td>
</tr>
</tbody>
</table>

**REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR**

**NOTES:**
- If PACKED and UNPKED are the same address space, UNPKED may overlap.
- Use #P for PACKED and UNPKED are the same address space, UNPKED may overlap.
- TRUPE cannot handle quote marks within character strings.

**CAUTION:**
BEGIN XRUPK
TRANSFER CALLING SEQUENCE
TURN ON BLANK REMOVAL
INITIALIZE COUNT
DO FOR EACH WORD OF PACKED
LOAD A WITH NEXT WORD
ROTATE A 8 BITS
DO FOR EACH BYTE OF WORD
AND OFF HIGH BYTE
IF A = QUOTE MARK
THEN
CHANGE BLANK REMOVAL OPTION
ENDIF
IF BLANK REMOVAL IS ON
THEN
IF A NOT = BLANK
THEN
INCREMENT COUNT
STORE A IN UNPKED
ENDIF
ENDIF
INCREMENT COUNT
STORE A IN UNPACKED
ENDIF
RELOAD A WITH WORD
ENDIF
ENDIF
END XRUPK
RETURN VALUE OF COUNT
ENDXRUPK
FORTAN CALLING PROCEDURE

CALL XR1SP (CHSTR, LENGH)

********************************************************
XR1SP COMPACTS A CHARACTER STRING STORED IN A2 FORMAT SUCH THAT
NO MORE THAN ONE CONSECUTIVE BLANK APPEARS EXCEPT BETWEEN
QUOTATION MARKS.
********************************************************
********************************************************
INPUT
CHSTR - CHARACTER STRING OF LENGH WORDS IN A2 FORMAT
LENGH - LENGTH OF CHSTR IN WORDS
********************************************************
OUTPUT
CHSTR - CHARACTER STRING OF LENGH WORDS IN A2 FORMAT WITH
DUPLICATE BLANKS REMOVED
LENGH - NEW LENGTH OF CHSTR IN WORDS
********************************************************
EXTERNAL REFERENCES
.ENTR
********************************************************
SPECIAL REMARKS
THIS ROUTINE ASSUMES
1. THE INPUT CHARACTER STRING HAS A POSITIVE LENGTH > 0
2. SPECIAL HANDLING OF STRINGS WITHIN QUOTATION MARKS DOES NOT
BEGIN UNTIL THE FIRST OCCURRENCE OF QUOTATION MARKS ON EACH
ENTRY
3. THE FIRST/LAST CHARACTER IN A CHARACTER STRING HAS THE
HIGH ORDER BIT OF THE QUOTE CHARACTER 'ON' TO INDICATE
BEGIN/END OF A CHARACTER STRING. THIS BIT IS TURND 'OFF'
BY THIS PROGRAM PRIOR TO EXIT. THIS CHANGE MADE TO
INCORPORATE QUOTE MARKS W/I CHARACTER STRINGS.
OTHER PROCESSORS CHANGED FOR THIS MODIFICATION WERE:
A. XILAN,XPRDM
B. XILSS/XILSS
********************************************************
BEGIN XR15P
CONVERT WORD COUNT INTO CHARACTER COUNT
SET STORE INDEX TO FIRST POSITION IN CHARACTER STRING
CLEAR BLANK FLAG
DO FOR EACH CHARACTER IN STRING
   IF CHARACTER IS SPECIAL CHARACTER INDICATING CHARACTER STRING , THEN
      REPLACE CHARACTER STRING INDICATOR WITH QUOTE MARK
   ELSE
      IF CHARACTER STRING FLAG IS CLEAR, THEN
         IF CHARACTER IS A BLANK, THEN
            IF BLANK FLAG IS SET (AT LEAST ONE PRECEDING BLANK), THEN
               SKIP THIS CHARACTER (EXIT TO ENDDO)
            ELSE
               SET BLANK FLAG
            ENDIF
         ELSE
            CLEAR BLANK FLAG
         ENDIF
      ENDIF
   ENDIF
ENDIF
STORE CHARACTER AT INDEXED POSITION
INCREMENT STORE INDEX
ENDDO
IF NUMBER OF CHARACTERS STORED 'S 000
   STORE ONE MORE BLANK
ENDIF
CONVERT STORE INDEX TO WORD COUNT AND RETURN
END XR15P
1 CD0      FORTRAN CALLING PROCEDURE
2 CD0
3 CD0
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0
9 CD0
10 CD0
11 CD0
12 CD0
13 CD0
14 CD0
15 CD0
16 CD0
17 CD0
18 CD0
19 CD0
20 CD0
21 CD0
22 CD0
23 CD0
24 CD0
25 CD0
26 CD0
27 CD0
28 CD0
29 CD0
30 CD0
31 CD0
32 CD0
33 CD0
34 CD0
35 CD0
36 CD0
37 CD0
38 CD0
39 CD0
40 CD0
41 CD0
42 CD0
43 CD0
44 CD0
45 CD0

XSERE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR

INPUT

COMMON XE - LU

COMMON XB - DEBUG, IRETC, NEWTAB, NUMENT, PROMLen

PROMPT, PRTND, WKBUF

OUTPUT

COMMON XE - RERBUF, REAPTR

COMMON XB - NUMENT, PrMLen, PRTND, PROMPT, WKBUF

NOTES

USES Routines

EXEC

XERTM

XREP

XRM0V

XRM0G

XSMPY

XSPCE

XSPPRM

XTRCM

XUG0B

XSERE
BEGIN XSERE
  DO UNTIL A % OR 'EXIT' IS ENTERED
  CALL XSPRM TO BUILD A PROMPT BASED ON PROMPT MODE FOR THE
  NEXT TABLE ENTRY (INDICATED BY TABMDX)
  CALL XICOM TO ISSUE THE PROMPT AND RETURN RESPONSE
  ERNEXIT IF ERROR IN XICOM :ERR10:
  EXIT XSERE IF % WAS ENTERED
    IF NOTHING (ONLY CR) ENTERED, THEN
    IF PROMPT MODE IS NOT 'ALL', THEN
    CALL XRMG - 'INVALID INPUT'
    ENDIF
  ELSE
  CALL XSNPT TO PROCESS INPUT BASED ON PROMPT MODE,
    CURRENT TABLE ENTRY (TABMDX), AND PROMPTED SEQUENCE
    NUMBER (PRNUM)
  ENDIF
  ENDDO
  BUILD XNA REQUEST TO DELETE/VERIFY ABSENCE OF ME,TAB
  CALL XSPCK TO PACK THE TABLE BUFFER (REMOVE DELETED ENTRIES)
  BUILD XNA REQUEST TO ALLOCATE NEWTAB
  IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
  BUILD XNA REQUEST TO STORE NEWTAB
  CALL XREN TO PROCESS THE REQUESTS
  IF THE ALLOCATE REQUEST FAILED, THEN
    CALL XRMG - 'AMA/DWA FULL, SEQUENCE TABLE NOT STORED'
  CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
  ELSE
    CALL XRMG - 'VACUOUS TABLE -- NOT STORED'
  ENDIF
  ENDF
  EXIT XSERE
  ENDF
  :ERR10: CALL XRMG - 'SYSTEM ERROR'
  END XSERE
BEGIN XSPRN

IF PROMPT MODE IS ALL, THEN

IF NUMBER OF ENTRIES (NUMENT) > 0, AND
ENTRIES EXIST BEYOND TABLE ENTRY INDEX (TABNEX), THEN
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT TABLE ENTRY INDEX (TABNEX) TO NEXT ENTRY (+7)
ENDO
BUILD PROMPT OF THE FORM ' XMMMM-PROC INT'
SET PROMPT SEQUENCE NUMBER (SEENO) TO SEQUENCE NO. OF ENTRY
ELSE
SET PROMPT NO. 6 TO CREATE
ENDIF

IF PROMPT MODE IS CREATE, THEN
SET TABLE ENTRY INDEX (TABNEX) TO NEXT ENTRY (NUMENT * 7 + 1)
IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
IF SEQUENCE NO. OF LAST ENTRY > 32690, THEN
CALL XPNL = "UNABLE TO BUILD SEQUENCE NO. > 32700"
SET PROMPT MODE TO UPDATE
ELSE
SET PROMPT SEQUENCE NUMBER (SEENO) TO NEXT MULTIPLE OF 100
BEYOND SEQUENCE NUMBER OF LAST TABLE ENTRY
ENDIF
ELSE
SET PROMPT SEQ. NO. (SEENO) TO BE 100
ENDIF
IF PROMPT MODE IS NOT UPDATE, THEN
BUILD PROMPT OF THE FORM ' XMMMM'
ENDIF
ELSE
IF PROMPT MODE IS UPDATE, THEN
BUILD PROMPT OF THE FORM
SET PROMPT LENGTH TO 0 CAUSING @: PROMPT TO BE ISSUED
ENDIF
ENDIF
END XSPRN
FORTRAN CALLING PROCEDURE

CALL XSNPT

XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE

TABLE EDITOR

INPUT

COMMON XE - COMBUF, COMPR, LK, TOKENS

COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, PRMTMD

SEGNO, TABNOX, WKBUF

OUTPUT

COMMON XE - COMPTR

COMMON XB - INSERT, IRET, NUMENT, PRMTMD, SEGNO,

TABNOX, TABSZ, WKBUF

NOTES

USES ROUTINES

XRMSG

XSDEL

XSENT

XSLIS

XSNUM

XSMT

XUDIG
**FORTRAN CALLING PROCEDURE**

**COMMON XE - COMBUF, COMPTR, TOKENS, LU**

**COMMON XB - DEBUG, LIBDSZ, NUMENT, PRMTMD, SEGNO, TABNDX, WBUF**

**OUTPUT**

**COMMON XE - COMPTR**

**COMMON XB - INTNAM, NUMENT, PRCNAM, PRMTMD, SEGNO, TABNDX, WBUF**

**NOTES**

**USES ROUTINES**

**XRCPF**

**XREXT**

**XRMOV**

**XRMSG**

**XRESET**

**XRPCK**

**XUE9G**

**BEGIN XSENT**

ERREXIT IF TOKEN INPUT IS NOT A NAME :ERROR1:

RETAIN THIS NAME AS PROC. NAME

INCREMENT TO NEXT TOKEN

START SEARCH UNTIL ALL ENTRIES OF LIBRARY SEARCHED

EXIT IF LIBRARY ENTRY = PROC. NAME

OURELSE

INCREMENT TO NEXT ENTRY

ENDLOOP

ERREXIT :ERROR2:

ENDSEARCH

IF COMMA IS NEXT TOKEN, THEN

ERREXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERROR4:

INCREMENT TO NEXT TOKEN

ERREXIT IF NEXT TOKEN IS NOT A NAME :ERROR1:

RETAIN THIS NAME AS INTERFACE TABLE NAME

INCREMENT TO NEXT TOKEN

ELSE

SET INTERFACE TABLE NAME TO ZERO
ENDIF

IF NEXT TOKEN IS NOT EOS :ERROR:

IF INSERT FLAG DOES NOT INDICATE REPLACE, THEN (I.E. INSERT OR ADD)

IF NUMBER OF TABLE ENTRIES (NUMENT) = 150, THEN

CALL XSPCK TO PACK TABLE BUFFER (REMOVE DELETED ENTRIES)

IF NUMBER OF TABLE ENTRIES STILL = 150, THEN

SET PROMPT MODE TO UPDATE

ERROR :ERROR5:

ENDIF

ENDIF

IF INSERT FLAG INDICATES INSERT (=1), THEN

IF ENTRY ABOVE INDEXED ENTRY (TABNDX) IS MARKED DELETED, THEN

SET TABLE ENTRY INDEX (TABNDX) TO BE THIS DELETED ENTRY

SET INSERT FLAG TO 0 INDICATING ENTRY REPLACEMENT

ELSE

SET MOVLEN = MIN (5, 150-NUMENT) + 7

DO FOR ALL TABLE ENTRIES FROM BOTTOM OF TABLE TO TABNDX

MOVE THE ENTRY DOWN MOVLEN WORDS

ENDDO

IF MOVLEN > 7 (I.E. MORE THAN 1 ENTRY), THEN

MARK FOLLOWING ENTRIES AS DELETED

ENDIF

SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SEQNO) INPUT/PROMPTED

EXIT XSENT

:ERROR: CALL XRMSE = 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'

:ERROR: CALL XRMSE = 'ITEM IS NOT A VALID PROCESSOR NAME'

:ERROR4: CALL XRMSE = '.... DOES NOT USE AN INTERFACE TABLE'

:ERROR5: CALL XRMSE = 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
BEGIN XSPCK
395  2  IF THE TABLE IS NOT EMPTY, THEN
396  3  DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
397  4  IF THIS ENTRY IS MARKED DELETED, THEN
398  5  SET MOVE LENGTH (MOVLEN) TO 7
399  6  DO UNTIL A NON-DELETED ENTRY IS FOUND
400  7  INCREMENT MOVLEN BY 7
401  8  ENDDO
402  9  MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
403 10  THE DELETED ENTRY
404 11  DECREMENT NUMENT BY MOVLEN/7
405 12  IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
406 13  DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
407 14  ENDDIF
408 15  ENDDO
409 16  ENDDIF
410 17  ENDDO
411 18  ENDDIF
412 19  END XSPCK
BEGIN XSLST
  IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
  DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
  SET LIMITS (BEGIN AND ENDNO) OF SEQ. ENTRIES LISTED
  SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
END IF
WRITE HEADER LINE - 'SEQUENCE TABLE XXXX'
IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
  DO FROM BEGIN TO ENDNO
  MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
  IF INT. TABLE NAME = 0, THEN
    SET LENGTH OF PRINT TO BE 7 WORDS (14 CHAR.)
  ELSE
    SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHAR.)
  ENDIF
  CALL XRIS TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
  WRITE PRINT BUFFER
ENDIF
END XSLST
FORTRAN CALLING PROCEDURE

CALL XSCAN

XSCAN PERFORMS SYNTACTICAL PROCESSING FOR THE LIST AND
DELETE DIRECTIVES OF THE SEQUENCE TABLE EDITOR.

INPUT
COMMON XE - COMBUF, COMPID, L.U. TOKENS

OUTPUT
COMMON XB - DEBUG, MUPRT, TASK2, MSGBUF

NOTES
USES ROUTINES XRMS, XMLIB

445
449
454
457
460
462
464
467
470
473
476
479
482
485
488
490
493
496
499

5-262
BEGIN XSCAN
SET LIST LIMITS (BEGNO AND ENDMO) TO ZERO
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERROR IF VALUE IS < 1 :ERROR6:
SET BEGIN LIMIT (BEGNO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERROR IF VALUE IS < 1 :ERROR6:
SET END LIMIT (ENDMO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
IF BEGIN LIMIT (BEGNO) = 0, THEN
SET BEGIN LIMIT (BEGNO) TO 1 (BEGNO IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGNO)
SET BEGIN LIMIT (BEGNO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERROR :ERROR6:
ENDIF
ENDIF
IF END LIMIT (ENDMO) = 0, THEN
SET END LIMIT (ENDMO) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGNO) TO LAST TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDMO)
SET END LIMIT (ENDMO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERROR :ERROR6:
ENDIF
ENDIF
SET RETURN CODE TO INDICATE NO ERROR
SET RETURN CODE TO INDICATE AN ERROR
EXIT XSLIS
:ERROR1: CALL XRMSE - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERROR6: CALL XRMSE - 'INVALID SEQUENCE NUMBER'
:ERROR8: CALL XRMSE - 'INVALID SEQUENCE NUMBER RANGE'
END XSCAN
FORTRAN CALLING PROCEDURE

CALL XSPMT

**CALL XSPMT PROCESSES THE SEQUENCE TABLE EDITOR PROMPT DIRECTIVE**

**INPUT**

COMMON XE - COMBUF, COMTR, LU, TOKENS

COMMON XB - DEBUG

**OUTPUT**

COMMON XB, PRTMD, TABNOS

**NOTES**

USES ROUTINES

XRMSG

XUDOS

BEGIN XSPMT

ERREXIT IF TOKEN IS NOT COMMA :ERROR1:

ERREXIT IF TOKEN IS NOT NAME :ERROR1:

ERREXIT IF TOKEN IS NOT EOS :ERROR1:

IF NAME IS 'M', THEN

SET PROMPT MODE TO CREATE

ELSE

ERREXIT IF NAME IS NOT 'A' :ERROR9:

SET PROMPT MODE TO ALL

SET TABLE ENTRY INDEX (TABNOS) TO 0

ENDIF

EXIT XSPMT

:ERROR1: CALL XRMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'

:ERROR9: CALL XRMSG - 'SYNTAX ERROR - INVALID QUALIFIER

END XSPMT
FORTRAN CALLING PROCEDURE

CALL XSDEL

C BEGIN XSDEL
CALL XSCL TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE
IF NO ERROR INDICATED, THEN
DO FROM THE BEGIN LIMIT (BEGNO) TO THE END LIMIT (ENMD)
MARK THIS SEQ. TABLE ENTRY AS DELETED
ENDDO
DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR
UNTIL A NON-DELETED ENTRY FOUND
IF ENTRY IS MARKED DELETED, THEN
DECREMENT NUMBER OF TABLE ENTRIES (NUMEN) BY ONE
ENDIF
ENDDO
ENDIF
END XSDEL
5.0 FORTRAN CALLING PROCEDURE
6.0 CALL XSLTS
3.0
********
1.0 XSLTS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR
2.0 INPUT
3.0 COMMON XE - LU
4.0 COMMON XB - DEBUG, ETC
5.0
********
6.0 NOTES
7.0 USES ROUTINES
8.0 XICAN
9.0 XSLST
10.0 XUDBE
11.0
********
12.0 FORTRAN CALLING PROCEDURE
13.0 CALL XSNUM
14.0
********
15.0 XSNUM IS THE SEQUENCE TABLE EDITOR NUMBER DIRECTIVE PROCESSOR
16.0 INPUT
17.0 COMMON XE - COMBUF, COMPTA, LU, TOKENS
18.0 COMMON XB - DEBUG, NUMERT
19.0
********
20.0 OUTPUT
21.0 COMMON XB - WKBUF
22.0
********
23.0 NOTES
24.0 USES ROUTINES
25.0 XRMSS
26.0 XUDBE
BEGIN XSNUM
ERREXIT IF TOKEN IS NOT EOS :ERROR1:
IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
  SET SEQUENCE NUMBER (SEQNO) TO 100
DO FOR ALL ENTRIES IN TABLE
  IF TABLE ENTRY IS NOT MARKED DELETED, THEN
    SET SEQ. NO. FIELD OF ENTRY TO SEQUENCE NUMBER (SEQNO)
    INCREMENT SEQUENCE NUMBER (SEQNO) BY 100
  ENDIF
ENDDO
EXIT XSNUM
:ERROR1: CALL XRSIG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
END XSNUM
**FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:**

CALL XIcom (PROMPT, PRMLEN, RETCOD)

**INPUTS FROM CALLING SEQUENCE:**

PROMPT - (INTEGER, PRMLEN WORDS) AN ARRAY OF PRMLEN WORDS

PRMLEN - (INTEGER, 1 WORD) THE LENGTH IN WORDS OF

THE PROMPT ARRAY. MAXIMUM LENGTH IS 38 WORDS.

**OUTPUTS IN CALLING SEQUENCE:**

RETCOD - (INTEGER, 1 WORD) IS A COMPLETION CODE DEFINED

AS FOLLOWS:

0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE

1 - USER RESPONDED %. BUFFER CONTAINS RESPONSE UP TO AND INCLUDING %.

2 - USER ENTERED A CR. THERE IS NO RESPONSE

3 - PROMPT WAS TOO LONG. MAXIMUM LENGTH IS 38 WORDS

4 - USER REQUESTED A CONTINUATION

**INTERNAL VARIABLES**

COMM - 6 WORD ARRAY CONTAINING THE CONTINUATION MESSAGE

PREFIX - 4 CHARACTERS USED AS PREFIXES TO PROMPT,

CORRESPONDING TO CODES IN XE(S)

RETCM - RETURN CODE FROM XIPRM EXTENDED PROMPTING

RETC1 - RETURN CODE FROM XIWAN LEXICAL ANALYSIS

SUFFIX - 5 CHARACTERS APPENDED TO END OF PROMPT

**XI COMMON USED:**

EQUIVALENCE (XE(1), LU), (XE(2), ICASE),

(EF(143), NOTOKS), (XE(146), NOTOKS)

**XS COMMON USED:**

(EF(145), NOTOKS), (XE(146), NOTOKS)
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
BEGIN XTCOM
  IF PROMPT IS NOT TOO LONG (76 CHAR) THEN
    MOVE PREFIX CHARACTER FOR EXEC LEVEL INTO OUTPUT AREA
    MOVE PROMPT INTO OUTPUT AREA
    MOVE SUFFIX CHARACTER INTO OUTPUT AREA
  :LOOP:
    ISSUE WRITE TO PROMPT USER
    INITIALIZE COMMUNICATIONS BUFFER
    TURN SYMBOLIC STRING FLAG OFF
    INITIALIZE LA RETURN CODE TO NORMAL RETURN
    PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
    DO WHILE LEXICAL ANALYSIS (LA) RETURN CODE SAYS CONTINUE AND
    (EXEC LEVEL IS NOT INTERFACE TABLE EDITOR OR
    SYMBOLIC STRING FLAG IS ON)
    CALL EXEC TO WRITE CONTINUATION MESSAGE
    PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
  ENDDO
  IF LA RETURN CODE SAYS ERROR IN RESPONSE THEN
    CALL KCVT TO CONVERT OCTAL TO ASCII
    CALL XRMSG TO WRITE ERROR MESSAGE
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
  ENDF
  IF LA RETURN CODE SAYS OVERFLOW/UNDERFLOW THEN
    CALL KCVT TO CONVERT OCTAL TO ASCII
    CALL XRMSG TO WRITE ERROR MESSAGE
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
  ENDF
  IF LA RETURN CODE SAYS EXTENDED PROMPTING WAS REQUESTED THEN
    CALL XTPRM FOR EXTENDED PROMPT
    IF EXTENDED PROMPT (EP) RETURN CODE SAYS INVALID REQUEST THEN
      CALL XRMSG TO WRITE ERROR MESSAGE
      ENDF
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
  ENDF
  IF LA RETURN CODE SAYS COMPUT IS FULL THEN
    CALL XRMSG TO WRITE ERROR MESSAGE
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
  ENDF
  SET XTCOM RETURN CODE = LA RETURN CODE
  ELSE
    SET XTCOM RETURN CODE = PROMPT IS TOO LONG
  ENDF
END XTCOM
131 1 BEGIN READSEG
132 2 CALL XRMOV TO INITIALIZE INPUT BUFFER TO BLANKS
133 3 CALL XRNC TO READ RESPONSE TO PROMPT
134 4 CALL XRUPK ROUTINE TO CONVERT A2 FORMAT BUFFER TO A1
135 5 IF NUMBER OF WORDS READ IS NOT ZERO THEN
136 6 CALL XTLM ROUTINE TO BUILD COMMUNICATIONS BUFFER
137 7 ELSE
138 8 IF LAST LA RETURN CODE WAS A CONTINUE THEN
139 9 REMOVE TRAILING COMMAS FROM COMBUF
140 10 SET LA RETURN CODE TO NORMAL RETURN
141 11 ELSE
142 12 SET LA RETURN CODE TO SAY USER ENTERED CR
143 13 ENDIF
144 14 ENDIF
145 15 END READSEG
FORTRAN CALLING PROCEDURE FOR LEXICAL ANALYSIS:

CALL XTLAM (RETC)

OUTPUTS IN CALLING SEQUENCE:

RETC - (INTEGER, 1 WORD) IS A COMPLETION CODE PASSED
BACK TO CALLER AS FOLLOWS:

0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE.
1 - USER RESPONDED X. BUFFER CONTAINS RESPONSE
UP TO AND INCLUDING X.
5 - USER REQUESTED A CONTINUATION.
6 - EXTENDED PROMPTING REQUEST WAS RECEIVED. BUFFER
CONTAINS RESPONSE UP TO AND INCLUDING THE REQUEST.
7 - COMMUNICATIONS BUFFER IS FULL.
1X - ERROR IN RESPONSE AT OR BEYOND CHARACTER XX.
2X - OVERFLOW/UNDERFLOW DETECTED AT OR BEYOND
CHARACTER XX.

INTERNAL VARIABLES

COMLEN - LENGTH IN WORDS OF COMBUF = 256
DBLINT - DOUBLE PRECISION LOCATION TO ACCUMULATE AN
INTEGER VALUE
DBLND - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER
AND FRACTIONAL VALUE FOR DOUBLE PRECISION OR REM.
FLGCOM - COMMA FLAG
FLGCH - LAST CHARACTER WAS NOT A COMMA
FLGCH2 - LAST CHARACTER WAS A COMMA
FLGEN - END LOOP FLAG
FLGEM - CONTINUE LOOP
FLGEM2 - END LOOP
FLGNEC - NEGATIVE EXPONENT FLAG
FLGEX - EXPONENT WAS POSITIVE
FLGEX2 - EXPONENT WAS NEGATIVE
FLGTY - TYPE OF REAL VALUE
FLGTY2 - SINGLE PRECISION
FLGTY3 - DOUBLE PRECISION
FLGWR - EXPONENT PART OF A REAL NUMBER
RELLND - SINGLE PRECISION LOCATION FOR REAL VALUE
SPC - SP CHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING
THE KTM INCLUDE REPRESENTATION FOR:

- + - / \ & = ? () ** ? X . Y Z ; : DEW
X IS A CLOSED BRACKET
Y IS AN OPEN BRACKET
2 IS A BACK SLASH
206  1 CD4
207  1 CD************
208  1 CD************
209  1 CD5
210  1 CD5
211  1 CD5
212  1 CD5
213  1 CD5
214  1 CD5
215  1 CD5
216  1 CD5
217  1 CD5
218  1 CD5
219  1 CD5
220  1 CD5
221  1 CD5
222  1 CD5
223  1 CD5
224  1 CD5
225  1 CD5
226  1 CD5
227  1 CD5
228  1 CD************
BEGIN XTLAN
   SET END FLAG OFF
   SET LAST CHARACTER WAS A COMMA ON
   INITIALIZE INDEX INTO INPUT BUFFER
   INITIALIZE RETURN CODE TO NORMAL RETURN
   DO WHILE END FLAG IS OFF
      IF INPUT BUFFER HAS BEEN COMPLETELY SCANNED THEN
         IF LAST CHARACTER WAS A COMMA OR A SYMBOLIC STRING IS STILL OPEN THEN
            SET RETURN CODE TO SAY CONTINUATION REQUESTED
      ENDIF
   SET END FLAG ON
   ELSE
      ERREXIT IF COMBUF IS FULL PERFORM COMFUL
   ENDIF
   IF INPUT CHARACTER IS A COMMA THEN
      PERFORM COMPA
   ELSE
      SET LAST CHARACTER WAS A COMMA OFF
      IF INPUT CHARACTER IS AN ALPHABETIC CHARACTER THEN
         PERFORM ALPHA
      ELSE
         IF INPUT CHARACTER IS A DIGIT THEN
            PERFORM DIGIT
         ELSE
            PERFORM SCHARS
         ENDIF
      ENDIF
   ENDIF
   END XTLAN

BEGIN XTLAN
   SET LAST CHARACTER WAS A COMMA ON
   IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
   STORE COMMA TOKEN IN COMBUF
   INCREMENT #TOKEMS BY 1
   END XTLAN

BEGIN XTLAN
   SET NEXT INPUT CHARACTER
   END XTLAN
BEGIN ALPHA
ERRIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
CALL XMOV TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
INPUT CHARACTER IS A NUMERIC OR
INPUT CHARACTER IS AN EXCLAMATION POINT) AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
MOVE CHARACTER INTO TEMPORARY BUFFER
GET NEXT INPUT CHARACTER
ENDO
SET CHARACTER COUNT = 6
STORE CHARACTER NAME TOKEN IN COMBUF
CALL XPACK ROUTINE TO PACK CHARACTERS INTO COMBUF
INCREMENT WORDS IN COMBUF BY 4
INCREMENT TOKENS BY 1
END ALPHA

BEGIN DIGIT
INITIALIZE POWER TO ZERO
PERFORM DCOL
IF INPUT BUFFER IS NOT EXHAUSTED THEN
IF INPUT CHARACTER IS A . THEN
PERFORM DECP
ELSE
IF INPUT CHARACTER IS AN "E" OR A "D" THEN
CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
PERFORM EOM
ELSE
IF INPUT CHARACTER IS AN "R" THEN
PERFORM REPEP
ELSE
PERFORM INTEGR
ENDIF
ELSE
PERFORM INTEGR
ENDIF
ENDIF
PERFORM INTEGR
ENDIF
1 BEGIN DCOL
2   SET INTEGER = 0
3   SET COUNTER = 0
4 DO WHILE CHARACTER IS A DIGIT AND
5     INPUT BUFFER IS NOT EXHAUSTED
6       SET INTEGER = (INTEGER + 10) * CURRENT CHARACTER - 48
7       ERNEXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
8     INCREMENT COUNTER BY1
9   GET NEXT CHARACTER
10 END DO
11 END DCOL
12
13 BEGIN DECF
14   CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
15   GET NEXT INPUT CHARACTER
16   IF INPUT BUFFER IS NOT EXHAUSTED THEN
17     IF INPUT CHARACTER IS A DIGIT THEN
18       PERFORM DCOL
19       ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
20       ERNEXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
21     ELSE
22       IF INPUT CHARACTER IS AN "E" OR A "D" THEN
23         PERFORM FORD
24       ELSE
25         PERFORM REAL
26       ENDIF
27     ELSE
28       PERFORM REAL
29     ENDIF
30   ELSE
31     PERFORM REAL
32   ENDIF
33 END DECF
349  1 BEGIN EORD
350  2    IF INPUT CHARACTER IS AN "E" THEN
351  3        SET TYPE FLAG TO "E"
352  4    ELSE
353  5        SET TYPE FLAG TO "D"
354  6    ENDIF
355  7    GET NEXT CHARACTER
356  8    ERREXIT IF INPUT BUFFER IS EXHAUSTED
357  9    SET NEGATIVE FLAG OFF
358 10    IF CHARACTER IS A - THEN
359 11        SET NEGATIVE FLAG ON
360 12    GET NEXT CHARACTER
361 13    ELSE
362 14        IF CHARACTER IS A + THEN
363 15            GET NEXT CHARACTER
364 16    ENDIF
365 17    ENDIF
366 18    ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
367 19        ERREXIT IF CHARACTER IS NOT A DIGIT PERFORM INV.
368 20    PERFORM BCOL
369 21    IF NEGATIVE FLAG IS ON THEN
370 22        SET POWER = -POWER
371 23    EENDIF
372 24    IF TYPE FLAG IS "E" THEN
373 25        PERFORM REAL
374 26    ELSE
375 27        PERFORM DBL
376 28    EENDIF
377 29    1 END EORD
477 1 BEGIN QUOTE
478 2 GET NEXT CHARACTER
479 2 SET #CHARACTERS = 0
480 2 DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
481 3 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
482 3 (INPUT CHARACTER IS A QUOTE AND
483 3 NEXT CHARACTER IS A QUOTE AND
484 3 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
485 3 INCREMENT #CHARACTERS BY 1
486 3 MOVE CHARACTER INTO TEMPORARY BUFFER
487 4 IF INPUT CHARACTER IS A QUOTE THEN
488 5 GET NEXT CHARACTER
489 5 ENDIF
490 5 GET NEXT CHARACTER
491 2 ENDDO
492 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
493 2 ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
494 2 ERREXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVALID
495 2 STORE CHARACTER STRING TOKEN IN COMBUF
496 2 CALL XRPCK TO PACK CHARACTERS INTO COMBUF
497 2 INCERMENT #WORDS IN COMBUF BY 2+ (#CHARACTERS+1)/2
498 2 INCREMENT #TOKENS BY 1
500 2 GET NEXT CHARACTER
501 1 END QUOTE
525  CD0      FORTRAN CALLING PROCEDURE
526  CD0      CALL XPARM
527  CD0
528  CD0
529  CD0
530  CD0
531  CD0
532  CD0
533  CD0
534  CD0
535  CD0
536  CD0
537  CD0
538  CD0
539  CD0
540  CD0
541  CD0
542  CD0
543  CD0
544  CD0
545  CD0
546  CD0
547  CD0
548  CD0
549  CD0
550  CD0
551  CD0
552  CD0
553  CD0
554  CD0
555  CD0
556  CD0
557  CD0
558  CD0
559  CD0
560  CD0
561  CD0
562  CD0
563  CD0
564  CD0
565  CD0
566  CD0
567  CD0
568  CD0
569  CD0
570  CD0
571  CD0
572  CD0
573  CD0
574  CD0
575  CD0
576  CD0
577  CD0
578  CD0
579  CD0
580  CD0
581  CD0
582  CD0
583  CD0

**FORTRAN CALLING PROCEDURE**

**CALL XPARM**

XPARM IS CALLED BY XTCOM TO PROVIDE EXTENDED PROMPTING WHEN A ? IS ENTERED FROM THE USER TERMINAL.

**INPUT**

- CARTRG, COMBUF, FLAGS, LU, MPROC, PRCKAM, SUBSTA,
- TOKENS
- XBO COMMON - ARGTR, WKBUF (FROM INTERFACE TABLE EDITOR)
- XL COMMON - PRSLEM (FROM XTCOM)
- VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)

**OUTPUT**

- XE COMMON - COMBUF (USED FOR SCRATCH SPACE)

**LISTING OF APPROPRIATE EXTENDED PROMPTS**

**INTERNAL VARIABLES**

- CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY
- FILE - FILE NAME OF CURRENT TABLE ENTRY
- INDEX - INDEX TO CURRENT TABLE ENTRY
- L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO
- FIRST LIST ITEM IN RECORD 1 (SEE SDD SECTION 6.2.4.3)
- MESSAGE - MESSAGE NUMBER, IF ANY, FOR CURRENT TABLE ENTRY
- N - NUMBER OF LIST ITEMS IN RECORDS 1 AND 2 (SEE SDD SECTION 6.2.4.3)
- RECORD - RECORD NUMBER WITH WHICH TO BEGIN PROCESSING FOR CURRENT
- TABLE ENTRY
- SEARCH - LIST SEARCH FLAG (1) FOR CURRENT TABLE ENTRY
- SIZE - LIST ITEM SIZE OF CURRENT TABLE ENTRY (NEGATIVE INDICATES
- LAST CHARACTER TO BE MASKED)
- TABLE - PROCESSING CONTROL TABLE FOR VARIOUS SYNTAX CONDITIONS

**ENTRY SYNTAX CONTINUE**

INDEX FILE REC SIZE SRC MESSAGE CONTINUE

<table>
<thead>
<tr>
<th>CD0</th>
<th>1</th>
<th>IX:</th>
<th>1</th>
<th>1</th>
<th>I</th>
<th>X</th>
<th>R</th>
<th>1</th>
<th>1</th>
<th>PROC</th>
<th>1</th>
<th>2</th>
<th>NO</th>
<th>NONE</th>
<th>NO</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD0</td>
<td>2</td>
<td>IX:</td>
<td>1</td>
<td>9</td>
<td>I</td>
<td>X</td>
<td>R</td>
<td>1</td>
<td>1</td>
<td>PROC</td>
<td>1</td>
<td>2</td>
<td>YES</td>
<td>XTO</td>
<td>NO</td>
<td>I</td>
</tr>
<tr>
<td>CD0</td>
<td>3</td>
<td>IX:</td>
<td>1</td>
<td>17</td>
<td>I</td>
<td>X</td>
<td>SP</td>
<td>1</td>
<td>2</td>
<td>NO</td>
<td>NONE</td>
<td>YES</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>4</td>
<td>IX:</td>
<td>1</td>
<td>25</td>
<td>I</td>
<td>X</td>
<td>L</td>
<td>2</td>
<td>-3</td>
<td>NO</td>
<td>NONE</td>
<td>NO</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>5</td>
<td>IX:</td>
<td>1</td>
<td>33</td>
<td>I</td>
<td>X</td>
<td>SP</td>
<td>1</td>
<td>2</td>
<td>YES</td>
<td>NONE</td>
<td>YES</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>6</td>
<td>IX:</td>
<td>1</td>
<td>41</td>
<td>I</td>
<td>X</td>
<td>P</td>
<td>3</td>
<td>128</td>
<td>NO</td>
<td>XTO</td>
<td>NO</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>7</td>
<td>IX:</td>
<td>1</td>
<td>49</td>
<td>I</td>
<td>X</td>
<td>SP</td>
<td>1</td>
<td>2</td>
<td>NO</td>
<td>NONE</td>
<td>YES</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>8</td>
<td>IX:</td>
<td>1</td>
<td>57</td>
<td>I</td>
<td>X</td>
<td>PRO</td>
<td>1</td>
<td>3</td>
<td>NO</td>
<td>NONE</td>
<td>NO</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>9</td>
<td>IX:</td>
<td>1</td>
<td>65</td>
<td>I</td>
<td>X</td>
<td>SP</td>
<td>1</td>
<td>2</td>
<td>YES</td>
<td>NONE</td>
<td>YES</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD0</td>
<td>10</td>
<td>IX:</td>
<td>1</td>
<td>73</td>
<td>I</td>
<td>X</td>
<td>PRO</td>
<td>1</td>
<td>3</td>
<td>YES</td>
<td>XTO</td>
<td>NO</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHERE P AND R INDICATE A PROMPT AND RESPONSE

**NOTES**
1 BEGIN XTPRM
2 PERFORM SETUP TO COMPLETE CONTROL TABLE AND INDEX TO APPROPRIATE ENTRY
3 DO UNTIL 'NO CONTINUE' FOUND (0 IN ENTRY CONTINUE FIELD)
4 IF OPEN SUCCESSFUL, THEN
5 POSITION TO INDICATED STARTING RECORD AND READ
6 EXIT TO :ERROR9: IF FAILURE
7 IF TABLE SIZE FIELD < 128 (NOT A LIST RECORD), THEN
8 IF SIZE > 0 (NO LAST CHARACTER MASKING & POSSIBLE SPANNING), THEN
9 IF RECORD SPANNED (N=SIZE > 126), THEN
10 READ SECOND RECORD AND APPEND TO FIRST RECORD DATA
11 EXIT TO :ERROR9: IF FAILURE
12 ENDIF
13 ELSE
14 SET SIZE POSITIVE
15 DO FOR EACH LIST ITEM (1-M)
16 BLANK LAST CHARACTER
17 ENDDO
18 ENDIF
19 IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), THEN
20 START SEARCH WHILE LIST ITEMS REMAIN TO BE EXAMINED
21 EXIT IF TOKEN LOCATED IN LIST
22 POSITION TO APPROPRIATE RECORD (I+L-1) AND READ
23 EXIT TO :ERROR9: IF FAILURE
24 SET SIZE TO 128 (ENTIRE RECORD TO BE DISPLAYED)
25 END LOOP
26 IF TABLE MESSAGE NUMBER FIELD > 0, THEN
27 CALL XRMGR TO DISPLAY 'NOT VALID ...' MESSAGE
28 EXIT TO ENDDO
29 END SEARCH
30 ENDIF
31 PERFORM DISPLAY
32 ELSE OPEN ERROR
33 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
34 CALL XRMGR TO DISPLAY 'NOT VALID ...' MESSAGE
35 ELSE
36 :ERROR9: CALL XRMGR TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
37 CLOSE FILE
38 ENDDO
39 END XTPRM
BEGIN SETUP
1
EXIT TO :ERR02: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?

2
CASE (/:, /:, /:, /:/) SUBSTA

3
IF FIRST TOKEN = ?, THEN
4
SET TABLE INDEX TO FIRST ENTRY
5
ELSE
6
SET TABLE INDEX TO SECOND ENTRY
7
ENDIF

8
IF FIRST TOKEN = ?, THEN
9
SET TABLE INDEX TO FOURTH ENTRY
10
ELSE
11
SET TABLE INDEX TO SIXTH ENTRY
12
ENDIF

13
IF PROMPT LENGTH = 0, THEN
14
ASSUME TABLE INDEX OF THIRD ENTRY
15
ELSE
16
ASSUME TABLE INDEX OF FOURTH ENTRY
17
ENDIF

18
IF FIRST TOKEN IS A NAME, THEN
19
INCREMENT TABLE INDEX BY TWO ENTRIES (NOW FIFTH OR SIXTH)
20
ENDIF

21
FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN

22
ENDIF

23
FORM TENTH ENTRY FILE NAME FROM > AND IT EDITOR CURRENT PROCESSOR NAME

24
IF FIRST TOKEN = ?, THEN
25
IF PROMPT LENGTH = 0, THEN
26
CHANGE TABLE INDEX TO SEVENTH ENTRY
27
COPY FILE NAME FROM ENTRY TEN TO ENTRY EIGHT
28
ELSE
29
CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY IT EDITOR
30
ENDIF

31
ELSE
32
IF PROMPT LENGTH = 0, THEN
33
CHANGE TABLE INDEX TO NINETH ENTRY
34
ENDIF

35
ENDIF

36
END CASE
37
EXIT SETUP

38
:ERR02: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING

39
END SETUP
FORTRAN CALLING PROCEDURE

CALL XUDBG (I,U, ID)

**FORTRAN CALLING PROCEDURE**

CALL XUDBG (I,U, ID)

**FORTRAN CALLING PROCEDURE**

CALL XUDBG (I,U, ID)

**FORTRAN CALLING PROCEDURE**
USING EXEC, IAND, XRLOC, XRMOV, XRUPK, XUDPL

WITH THE EXCEPTION OF 'OPTION' AND 'OUTPUT UNIT' ALL INPUTS ARE IN

OCTAL. NO ERROR CHECKING IS PERFORMED THEREFORE CARE SHOULD BE

TAKEN TO ASSURE VALID DATA.
69 1 BEGIN XU86
70 2  WRITE SNAP HEADER TO TERMINAL
71 3  DO UNTIL OPTION IS EXIT (E)
72 4  PROMPT TERMINAL FOR OPTION
73 5  IF OPTION IS NOT EXIT (E)
74 6  THEN
75 7  PERFORM PROMPT AND CONVERSION FOR STARTING ADDRESS
76 8  IF OPTION IS MODIFY (%)
77 9  THEN
78 10  PERFORM PROMPT AND CONVERSION FOR VALUES AND STORE IN MEMORY
79 11  ELSE
80 12  PERFORM PROMPT AND CONVERSION FOR ENDING ADDRESS
81 13  PERFORM PROMPT AND CONVERSION FOR OUTPUT UNIT
82 14  WRITE SNAP HEADER TO OUTPUT UNIT
83 15  DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
84 16  CALL XU86 TO FORMAT LINE
85 17  OUTPUT LINE
86 18  END DO
87 19  ENDBL
88 20  ENDIF
89 21  END DO
90 22 1 BEGIN XU86
91 23 1 BEGIN PROMPT AND CONVERSION
92 24 1 ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
93 25 1 RETRIEVE RESPONSE
94 26 1 CLEAR RESPONSE
95 27 1 DO FOR EACH CHARACTER
96 28 1 SHIFT SUM AND ADD NEXT DIGIT
97 29 1 END DO
98 30 1 END PROMPT AND CONVERSION
100 1 CD1  GENERAL FILE DUMP PROGRAM FOR FILE MANAGER FILES  
101 1 CD1  
102 1 CD1  
103 1 CD2  INPUT  
104 1 CD2  NAME - NAME OF FN FILE TO BE DUMPED  
105 1 CD2  LREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED  
106 1 CD2  IREC (FIRST RECORD IS RECORD NUMBER ONE)  
107 1 CD2  NREC - NUMBER OF LOGICAL RECORDS TO DUMP  
108 1 CD2  FMT - RUN TIME FORMAT FOR RECORDS (MAXIMUM OF 72 CHARACTERS) OR BLANK INDICATING THE DEFAULT OF OCTAL AND ASCII DUMP TYPE  
109 1 CD2  LU - LOGICAL UNIT NUMBER OF OUTPUT DEVICE  
110 1 CD2  OUTPUT  
111 1 CD2  FORMATTED DUMP OF THE INDICATE PORTION OF THE INDICATED FILE  
112 1 CD2  
113 1 CD2  NOTES  
114 1 CD2  USES EXEC, MAXO, OPEN, POINT, READF, RMPAR, XPR8S, XMOV, XUPPL  
115 1 CD2  ANY FILE WITH VARIABLE LENGTH RECORDS WILL BE DUMPED USING A RECORD BUFFER OF 1024 WORDS.  
116 1 CD2  RECORD LENGTH LIMITING THE MAXIMUM DUMPABLE  
117 1 CD2  
118 1 CD2  
119 1 CD2  
120 1 CD2  
121 1 CD2  
122 1 CD2  
123 1 CD2  
124 1 CD2  
125 1 CD2  
126 1 CD2  
127 1 CD2  
128 1 CD2  
129 1 CD2  
130 1 *  
131 1 *  
132 1 BEGIN XUPDF  
133 2 DO FOREVER  
134 3 READ FILE NAME  
135 3 EXIT XUPDF IF NAME IS NULL  
136 3 READ INITIAL RECORD NUMBER  
137 3 READ NUMBER OF RECORDS TO DUMP  
138 3 READ DUMP FORMAT  
139 3 EXIT IF FORMAT IS NULL  
140 4 THEN  
141 5 SET DEFAULT OCTAL/ASCII FORMAT  
142 6 ELSE  
143 7 ENDIF  
144 8 IF SUCCESSFUL  
145 9 THEN  
146 10 DO FOR NUMBER OF RECORDS TO DUMP  
147 11 READ RECORD  
148 12 EXIT TO ELOOP IF FAILED  
149 13 FORMAT AND PRINT RECORD  
150 14 ELSE  
151 15 ENDIF  
152 16 ELSE  
153 17 :ERROR: OUTPUT MESSAGE  
154 18 ENDIF  
155 19 ENDDO  
156 1 ENDF XUPDF
FORTRAN CALLING PROCEDURE

158 1 CD0  CALL XUDPL (ADDRESS, LINE, BUFFER)
159 1 CD0
160 1 CD0
161 1 CD0
162 1 CD0
163 1 CD0
164 1 CD0
165 1 CD0
166 1 CD0
167 1 CD0
168 1 CD0
169 1 CD0
170 1 CD0
171 1 CD0
172 1 CD0
173 1 CD0
174 1 CD0
175 1 CD0
176 1 CD0
177 1 CD0
178 1 CD0
179 1 CD0
180 1 CD0
181 1 CD0
182 1 CD0
183 1 CD0
184 1 CD0
185 1 CD0
186 1 CD0
187 1 CD0
188 1 CD0
189 1 CD0
190 5 * 
191 1 * 
192 1 * 
193 1 * 
194 1 * 
195 1 * 
196 1 * 
197 1 * 
198 1 * 
199 1 * 
200 1 * 
201 1 * 
202 1 * 
203 1 * 
204 1 * 
205 1 * 

ORIGINAL PAGE IS RY68
CD1  FORTRAN MAIN PROGRAM XUFNT IS SCHEDULED BY XUDMP TO PRINT
CD2  A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD1  ********
CD2  INPUT
CD3  ICNTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
CD4  LU NUMBER OF WHERE DUMP IS ON DISK
CD5  TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
CD6  CONTAINING THE DUMP
CD1  ********
CD2  OUTPUT
CD3  FORMATTED DUMP TO LU 6
CD1  ********
CD2  EXTERNAL REFERENCES
CD3  EXEC
CD5  RMPAR
CD6  XRCPR
CD7  XMOV
CD8  XUDPL
CD9  BEGIN XUFNT
CD1  CALL RMPAR TO GET LU AND STARTING TRACK NOS.
CD2  READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
CD3  CNT OF ID-SEGS TO BE DUMPED
CD4  UP TO 7 ID-SEGMENT ADDRESSES
CD5  LOW AND HIGH BASE PAGE ADDRESSES
CD6  LOW AND HIGH MAIN MEMORY ADDRESSES
CD7  POINT TO 2ND SECTOR OF DUMP DATA
CD8  DO UNTIL ALL ID-SEGS PRINTED
CD9  DO UNTIL A EIGHT-WORD LINES PRINTED
CD10  PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
CD11  ENDDO
CD12  INCREMENT TO NEXT SECTOR OF DUMP DATA
CD13  ENDDO
CD14  COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
CD15  PERFORM COMPAIR AND PRINT FUNCTION
CD16  COMPUTE HTKS, NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ
CD17  DO UNTIL HTKS ARE READ
CD18  READ NEXT TRACK FROM DISK
CD19  COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP
CD20  PERFORM COMPAIR AND PRINT FUNCTION
CD21  ENDDO
CD22  RELEASE THE GLOBALLY ALLOCATED TRACKS
CD23  EXIT XUFNT
CD24  BEGIN COMPAIR AND PRINT FUNCTION
CD25  DO UNTIL N LINES PROCESSED
CD26  IF NOT 1ST LINE, THEN
CD27  CALL XRCPR TO COMPARE WITH PREVIOUS LINE
CD28  IF LINES ARE IDENTICAL, THEN
CD29  IF THIS IS 1ST OF A SERIES, THEN
CD30  WRITE 'DUPLICATE LINE'
CD31  ENDDO
CD32  ELSE
CD33  CALL XUDPL TO FORMAT THE DUMP LINE
CD34  WRITE FORMATTED DUMP LINE
CD35  ENDDO
CD36  ELSE
CD37  CALL XUDPL TO FORMAT THE DUMP LINE
CD38  WRITE FORMATTED DUMP LINE
CD39  ENDDO
CD40  ENDDO
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 *01 ENTRY XVABN
2 *02* TYPE 1A ROUTINE TO ABORT CURRENT ID AND
3 *02* REQUEST A DUMP FOR THE ID.
4 *02* ENTRY XVABN
5 *02* TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
6 *02* THE REQUESTED DATA AREAS AND THE CURRENT
7 *02* ID SEGMENT AND ITS FATHER CHAIN SEGMENTS.
8 *02* BEGIN XVABN ABEND
9 *02* INPUTS: NONE
10 *03 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
11 *03* EXTERNALS: SABRT,SABRE,SXEN,SLIBR
12 *03* CALL SLIBR - REENTRANT ROUTINE CALL
13 *03* GET CURRENT ID FROM XERT (LOCATION 1717)
14 *03* INCREMENT TO ADDRESS BOUNDARIES
15 *03* GET ADDRESS BOUNDARIES
16 *03* STORE := XUDMP PARAMETER LIST
17 *03* PERFORM XUDMP - DUMP THIS PARTITION
18 *03* CALL SABRE - ABORT THIS PARTITION
19 *03* CALL SABRE - FREE CURRENT REENTRANT DATA BLOCK
20 *03* EXIT TO :SXEN - GO TO DISPATCHER
21 *03* END XVABN
22 *03* END XVABN
23 *03* BEGIN XUDMP
24 *03* ENTRY XUDMP
25 *03* INPUTS: ADDRESS OF DOUBLE WORD CONTAINING START AND END ADDRESSES
26 *03* OF DEFINING AREA TO BE DUMPED.
27 *03* OUTPUTS: OUTPUTS TO DISK.
28 *03* .HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW
29 *03* .CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS
30 *03* .MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION
31 *03* .MEMORY IMAGE AS SPECIFIED BY INPUT ADDRESS RANGE
32 *03* SCHEDULES XUFTM TO FORMAT DATA FROM DISK TO PRINT
33 *03* EXTERNALS: SLIBR, SLIBX, EXEC, XUFTM
34 *03* CALL SLIBR - NOTIFY OF REENTRANCY AND BECOME PRIVILEGED
35 *03* RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH
36 *03* CALL SLIBX - TO BECOME NOT-PRIVILEGED.
37 *03* CALL EXEC FOR A GLOBAL ALLOCATION OF DISK
38 *03* CALL EXEC TO WRITE HEADER DATA USING CLASS I/O
39 *03* DO UNTIL ALL ID SEGMENTS PROCESSED
40 *03* WRITE ID SEGMENT TO DISK
41 *03* ENDDO
42 *03* WRITE BASE PAGE SEGMENT TO REMAINING PORTION OF THIS TRACK
43 *03* DO UNTIL END ADDRESS REACHED
44 *03* WRITE ONE SECTOR OF DATA
45 *03* ENDDO
46 *03* SCHEDULE XUFTM WITH LU AND STARTING TRACK NOS.
47 *03* CALL SLIBX TO RETURN FROM REENTRANT ROUTINE
48 *03* END XUDMP
**Assembly Language Code**

```
58  1  *01 TYPE 14 ROUTINE TO CONTROL COMMUNICATION BETWEEN AND
59  1  EXECUTION OF FDS MANAGER AND ITS ASSOCIATED TASKS
60  1  (EXECUTIVE, PROCESSORS, AND UTILITIES).
61  1  * ENTRY XVPAN AND XVSBO
62  1  * INPUTS
63  1  * FROM AN ASSOCIATED TASK
64  1  * CALL XVPAN(PARMS)
65  1  * ASSEMBLY FORM
66  1  * JSB XVPAN
67  1  * DEF =+2 = RETURN ADDRESS
68  1  * DEF PARMS A(PARMS)
69  1  * WHERE PARMS ARE P1,P2,P3,P4,P5
70  1  * P1 IS THE SERVICE REQUEST
71  1  * P2 = NORMAL TERMINATION (P2-P5 NOT USED)
72  1  * P3 = WORK AREA REQUEST (P2-P5 NOT USED)
73  1  * P4 = EXECUTE A SEQUENCE TABLE
74  1  * P5 = TERMINATE SEQUENCE (P2-P5 NOT USED)
75  1  * 3= TERMINATE SEQUENCE (P2-P5 NOT USED)
76  1  * 9= TERMINATE FDS FUNCTION (P2-P5 NOT USED)
77  1  * 32767= ABNORMAL TERMINATION OF ASSOCIATED TASK
78  1  * FROM AN FDS MANAGER
79  1  * ASSEMBLY FORM
80  1  * JSB XVPAN
81  1  * DEF (RETURN POINT)
82  1  * OCT 0
83  1  * DEF PARMS A(FDS MANAGER RESPONSE)
84  1  * OUTPUTS
85  1  * TO AN FDS MANAGER
86  1  * REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
87  1  * RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
88  1  * CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA BLIST
89  1  * MANAGER IS ACTIVATED VIA BLIST
90  1  * TO AN ASSOCIATED TASK
91  1  * RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
92  1  * RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
93  1  * CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA BLIST
94  1  * ASSOCIATE TASK IS ACTIVATED VIA BLIST
```
103 1 BEGIN XVPAM
104 2 CALL BLIBM BECOME PRIVILEGED
105 2 SET STOP-ID FROM XRT (OCT 1777)
106 2 IF THIS IS A MANAGER RESPONSE
107 2 THEN SET UP TO ACTIVATE ASSOCIATED TASK AND SUSPEND MANAGER
108 3 * CALL SEQUENCE IS RETURN,3,A(PARMS)
109 3 PERFORM XVPAM POST REQUESTOR AND WAIT
110 2 ELSE SET UP TO ACTIVATE MANAGER AND SUSPEND ASSOCIATED TASK
111 3 * CALL SEQUENCE IS RETURN,3,A(PARMS)
112 3 PERFORM XVPAM POST MANAGER AND WAIT
113 2 EMDIF
114 2 * STOP-ID HAS ID-SEGMENT TO BE SUSPENDED,
115 2 * AWAKEN-ID HAS ID-SEGMENT TO BE ACTIVATED,
116 2 CALL SLIST (SCHEDULE,AWAKEN-ID)
117 2 MOVE RETURN ADDRESS TO XSUSP OF STOP-ID.
118 2 CALL SLIST (WAIT,STOP-ID)
119 1 EXIT TO :$XEN RTE DISPATCHER
120 1 END XVPAM
122 1 BEGIN XVPMAW
123 2 * DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
124 2 * FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
125 2 * SET TARGET-ID FROM CURRENT-ID-SEGMENT
126 2 * compute father-id-number .NE. 0 or father is waiting
127 2 * PERFORM MGRFND(father-id-segment,count)
128 2 * EXITS COUNT .GT. 0
129 2 * SET TARGET-ID TO FATHER-ID-SEGMENT
130 2 ENDDO
131 3 IF father-id-number .EQ. 0, or father not waiting then
132 3 call s sym (12,*XV03,segment-name) *XV03,NAME* REQUESTING PROG
133 3 PERFORM PUMP
134 3 EXIT TO 2.LIBX TO ENABLE AND REDISPATCH
135 2 ENDF
136 3 SET AWAKEN-ID FROM FSD-ENTRY SBTAG
137 3 GET REQUEST PARNs MOVE INTO ID-SEGMENT
138 3 SET STBAT FROM CURRENT-ID
139 2 FND XVPMAW
140 1 *
141 1 *
142 1 *
143 1 *
144 1 *
145 1 DEF: XVPMAW
146 2 * DETERMINE IF CALLER IS A VALID FDS MANAGER
147 2 * PERFORM MGRFND (CURRENT-ID,COUNT)
148 2 * COUNT WILL BE 0 FOR NO MATCH.
149 3 * COUNT NOT EQUAL ZERO IMPLIES A MATCH
150 3 * AND FSD-ENTRY HAS MATCHING FSDTAG ENTRY ADDRESS
151 3 * IF COUNT .LT. 0 THEN CALLER IS NOT A FDS MANAGER
152 3 call s sym (12,*XV01,segment-name) *XV01 PROGRAM* REQUESTING PROG.
153 3 PERFORM PUMP
154 3 EXIT TO :SSeek THE DISPATCHER
155 2 ENDF
156 2 * FSDTAG-ENTRY HAS ENTRY FOR RESPONDING MANAGER
157 2 * SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
158 2 IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
159 2 call s sym (12,*XV02,segment-name) *XV02 PROGRAM* ASSOCIATED PROG.
160 3 PERFORM PUMP
161 3 EXIT TO :SSeek THE DISPATCHER
162 2 ENDF
163 2 * MOVE FDS MANAGERS INPUT PARMbS TO ASSOCIATED TASK ID SEGMENT
164 2 * MOVE PARMs TO ID-SEGMENT WORDS 2-6
165 1 END XVPMAW
172 1 BEGIN PUMP
173 2 * SET A CALL TO XVABN AT CALLER'S SUSPEND POINT
175 2 * END PUMP
177 1 *
178 1 *
180 1 *
181 1 *
182 1 BEGIN MGIFND INPUT IS TARGET-ID
183 2 GET XVSTO TABLE OF ACTIVE FDS MANAGERS
184 2 SET COUNT TO NUMBER IN TABLE NUMBER MANAGERS
185 2 SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
186 2 DO WHILE COUNT .GT. 0
187 2 EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMENT
188 3 SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
189 3 SET COUNT = COUNT - 1
190 2 ENDDO
191 2 * FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
192 2 COUNT = 0 MEANS NO MATCH
193 1 END MGIFND
194 1 *
195 1 *
197 1 *
198 1 BEGIN XVSTO
199 2 N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSTO
200 2 STBNM DEF N NUMBER-IN-TABLE
201 2 STBCF DEF 0 NUMBER-ACTIVE
202 2 STBDN DEF 0 STD RESOURCE NUMBER
203 2 STAES EQU * ENTRY START
204 2 STBLU DEF 0 L(UCCAL)
205 2 STBALA DEF 0 L(CASCII)
206 2 STBMG DEF 0 A(FDS-MANAGER-ID-SEGMENT)
207 2 STBEX DEF 0 A(FDS-EXECUTIVE-ID-SEGMENT)
208 2 STBEC DEF 0 A(CLASS-NUMBER FOR EXEC)
209 2 STBAT DEF 0 A(CURRENT-ASSOCIATED-TASK)
210 2 STBCF DEF 0 A(CLASS-NUMBER FOR PROCESSOR)
212 2 STBEG DEF 0 ENTRY "END"
213 2 STBEE EQU STBEE-STBES ENTRY LENGTH
214 2 STBES EQU N NUMBER-OF-ENTRIES
215 2 ORG STBES
216 2 REP N DO FOR N ENTRIES
217 2 REP STBEL DEFINE STORAGE WORDS FOR AN ENTRY
219 1 END XVSTO
**FORTRAN CALLING PROCEDURE:**

CALL IXAUT

**IXAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE**

**INPUTS IN COMMON:**
- XE(5) MASSTA, XE(10) SERRSTR, XE(11) SEGEND, XE(12) SEGPR,
- XB(140) TABEND, XB(1) NOPROC, XB(2) LIBD,
- XB(249) SEENO, XB(250) SNLEN, XB(251) SENTAB

**OUTPUTS IN COMMON:**
- XE(5) MASSTA, XE(1) FLGTAB

**COMMON USED:**
- EQUVALENCE (XE(5), MASSTA)
- +(XE(10), SERRSTR), (XE(11), SEGEND)
- +(XE(12), SEGPR), (XE(140), TABEND),
- +(XB(1), NOPROC), (XB(2), LIBD),
- +(XB(235), RESIND), (XB(249), SEENO),
- +(XB(250), SNLEN), (XB(251), SENTAB),
- +(XE(13), FLGTAB)

**FBS ROUTINES CALLED:**
- XCRPR, XEXT, XMMOV, XMSP,
- XEXE, XISTO, XITMP

**RTE ROUTINES CALLED:**
- IOE
49 1 BEGIN XXAUT
50 2 IF ENTRY IS FROM A DIRECTIVE THEN
51 3 SET MASTER STATE TO INDICATE REENTRY
52 4 DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
53 5 SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
54 6 IF EXIT IF PROCESSOR NOT FOUND TO :ERR1:
55 7 STUFF INTERFACE TABLE 677A AND VERSION INTO SEQUENCE TABLE ENTRY
56 8 ENDIF
57 9 CALL XXSTO TO STORE REVISED SEQUENCE TABLE IN AWA AS &SEQTB
58 10 ELSE (I AM BEING REENTERED FROM INT.)
59 11 CALL XXTP TO SET UP TEMPORARY EN-HY W71N &INTAB
60 12 CALL XXEXE TO EXECUTE FROM TEMPORARY ENTRY
61 13 IF RESET SEQUENCE NUMBER IS NOT REQUESTED THEN
62 14 EXIT XXAUT IF TERMINAL ENTRY WAS JUST EXECUTED
63 15 SET STARTING ENTRY TO NEXT ENTRY
64 16 ENDIF
65 17 ENDIF
66 18 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
67 19 CALL XXEXE TO EXECUTE REMAINDER OF TABLE
68 20 ENDDO
69 1 EXIT XXAUT
70 2 :ERR1:
71 2 CALL XXMSG TO DISPLAY INVALID PROCESSOR NAME
72 1 END XXAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELDS (XICNT)

XICNT IS THE MAIN PROGRAM FOR THE EXECUTION CONTROLLER.
IT GIVES CONTROL TO THE APPROPRIATE SUBROUTINE DEPENDING ON THE MODE AND RETURNS TO DIRECTIVE LEVEL.

INPUTS FROM CALLING SEQUENCE:

MODE - (INTEGER, 1 WORD) MODE IN WHICH THE EXECUTION CONTROLLER WAS CALLED
0 - MANUAL
1 - SEMI AUTOMATIC
2 - AUTOMATIC WITH TRACE
3 - AUTOMATIC

INTERNAL VARIABLES:

COMMON USED:

EQUIVALENCE (XEXP, NASTA)

FDS ROUTINES USED
XERMT, XEXPT, XERMSG, XERMAN

NOTE: CONTAINS DUMMY CALL TO XEXEC
119 1 BEGIN XICNT
120 2 SET MODE TO XREXT OF BITS 12 AND 13 OF WASSTA
121 2 CASE MODE (:MANU:, :SENI:, :AUIT:, :AUTO:)
122 3 :MANU: CALL XXMAN
123 3 :SENI: CALL XXSEN
124 3 :AUIT: CALL XXMSG TO DISPLAY INVALID OPTION
125 3 :AUTO: CALL XXAUT
126 2 ENDCASE
127 2 SET MASTER STATE TO DIRECTIVE LEVEL
128 2 CALL XERTH TO RETURN TO XEXEC **NO RETURN**
129 2 DUMMY CALL XEXEC
130 1 END XICNT
132 1 CD************
133 1 CD0
134 1 CD0  FORTRAN CALLING SEQUENCE:
135 1 CD0  CALL XXDEC (RETC)
136 1 CD0
137 1 CD0************
138 1 CD0
139 1 CD1  XXDEC DECODES A RESPONSE OF PROCESSOR NAME (INT TABLE NAME)
140 1 CD1  INTO A SEQUENCE TABLE ENTRY.
141 1 CD1
142 1 CD1************
143 1 CD1
144 1 CD2  INPUTS IN COMMON:
145 1 CD2
146 1 CD2  XE(85) TOKENS, XE(145) COMBUF, XE(1) NOPROC, XE(2) LIBD
147 1 CD2
148 1 CD2************
149 1 CD2
150 1 CD3
151 1 CD3  OUTPUTS IN CALLING SEQUENCE:
152 1 CD3
153 1 CD3  RETC - RETURN CODE (O IS NORMAL RETURN)
154 1 CD3
155 1 CD3
156 1 CD3  OUTPUTS IN COMMON:
157 1 CD3
158 1 CD3  XE(16) PRCNAM, XE(251) SEQTAB
159 1 CD3************
160 1 CD3
161 1 CD3  COMMON USED:
162 1 CD3
163 1 CD5  EQUIVANCE
164 1 CD5
165 1 CD5  + (XE(85), TOKEHS), (XE(145), T0KPTR),
166 1 CD5  + (XE(85), COMBUF),
167 1 CD5  + (XE(1), NOPROC), (XE(2), LIBD ),
168 1 CD5
169 1 CD5  FDS ROUTINES USED:
170 1 CD5
171 1 CD5
172 1 CD5
173 1 CD5
174 1 CD5
175 1 CD5
176 1 CD5
177 1 CD5************
179 1 BEGIN XXDEC
180 2 INITIALIZE RETURN CODE TO ZERO
181 2 SET SEQUENCE ENTRY TO ZEROS
182 2 ERREXIT IF FIRST TOKEN IS NOT A PROCESSOR NAME TO :ERR1:
183 2 SEARCH LIBRARY DIRECTORY FOR PROCESSOR
184 2 ERREXIT IF NAME NOT FOUND TO :ERR1:
185 2 MOVE PROCESSOR NAME, IT BIT AND VERSION INTO SEQUENCE ENTRY
186 2 IF AN INTERFACE TABLE NAME WAS ENTERED THEN
187 3 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
188 2 ENDIF
189 2 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
190 2 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
191 2 IF AN INTERFACE TABLE IS REQUIRED BUT NOT SPECIFIED THEN
192 3 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'SINTAB'
193 2 ENDIF
194 1 EXIT XXDEC
195 2 :ERR1:
196 2 CALL XRNUG TO DISPLAY ERROR
197 2 SET RETURN CODE TO SAY ERROR
198 1 END XXDEC
200 CD************
201 CD0  
202 CD0 FORTRAN CALLING PROCEDURE:
203 CD0  
204 CD0 CALL XXDEF
205 CD0  
206 CD************
207 CD1 XXDEF READS IN THE DEFAULT INTERFACE TABLE FOR A PROCESSOR
208 CD1 AND STORES IT IN THE AWA AS GINTAB
209 CD1  
210 CD1  
211 CD************
212 CD2 INPUTS FROM COMMON:
213 CD2  
214 CD2 PRCHNAM - (INTEGER, 3 WORDS) NAME OF PROCESSOR IN SERTAB
215 CD2 FOR WHICH NO INTERFACE TABLE WAS SUPPLIED
216 CD2  
217 CD2  
218 CD************
219 CD4 INTERNAL VARIABLES:
220 CD4  
221 CD4  
222 CD4 DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE
223 CD4 DEFAULT INTERFACE TABLE CAN BE READ INTO
224 CD4 DEFHAN - (INTEGER, 3 WORDS) ARRAY WHERE INTERFACE TABLE
225 CD4 NAME IS CREATED FROM PROCESSOR NAME
226 CD4  
227 CD************
228 CD5 COMMON USED:
229 CD5  
230 CD5  
231 CD5 EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA),
232 CD5 + (XE(13), INTNAM), (XE(16), PRCHN),
233 CD5 + (XE(19), REPTR), (XE(20), REBUF),
234 CD5 + (XE(142),ICR ),
235 CD5 + (XS(6), TMPTAB), (XS(14), DEFHAN),
236 CD5 + (XS(18), LEN1 ), (XS(19), LEN2 ),
237 CD5 + (XS(20), IDCB ), (XS(200),RETC )
238 CD5  
239 CD5 RTE ROUTINES USED:
240 CD5  
241 CD5 CLOSE, EXEC, KCVT, OPEN, READF
242 CD5  
243 CD5  
244 CD5 FDS ROUTINES USED:
245 CD5  
246 CD5 XERTH, XREQ, XREXT, XMNOV, XRMNG, XRCK, XRUPE
247 CD5  
248 CD************
249  1 BEGIN XXDEF
250   2 CREATE DEFAULT INTERFACE TABLE NAME FROM PROCESSOR NAME
251   2 CALL OPEN TO OPEN FILE
252   2 ERREXIT IF OPEN ERROR TO :FILERR:
253   2 CALL READ TO READ RECORD 1
254   2 ERREXIT IF READ ERROR TO :FILERR:
255   3 IF THERE IS LITERAL DATA THEN
256   3 CALL READ TO READ RECORD 2
257   3 ERREXIT IF READ ERROR TO :FILERR:
258  2 ENDF
259  2 CALL CLOSE TO CLOSE FILE
260  2 ERREXIT IF CLOSE ERROR TO :FILERR:
261  2 CALL XREG TO ALLOCATE & STORE XINTAB
262  2 ERREXIT IF RETURN CODE IS NOT ZERO TO :MGERR:
263  3 IF DEFAULT TABLE IS INCOMPLETE THEN
264  3 SET INTERFACE TABLE NAME IN XE TO XINTAB
265  3 CALL XSUSTATE TO INTERFACE TABLE EDITOR **NO RETURN**
266  3 ENDIF
267  2 ENDF
268  1 EXIT XXDEF
269  2 :FILERR:
270  2 CALL CLOSE TO CLOSE FILE
271  2 CALL XRMG TO DISPLAY FILE ACCESS ERROR
272  1 EXIT XXDEF
273  2 :MGERR:
274  2 CALL XRMG TO DISPLAY SPACE ERROR
275  2 CALL EXEC TO FREE CLASS NUMBER
276  2 SET MASTER STATE TO DIRECTIVE LEVEL
277  2 CALL XERTN TO RETURN TO EXEC **NO RETURN**
278  1 END XXDEF
FORTRAN CALLING PROCEDURE:

CALL XXEXE

XXEXE REQUESTS THE MANAGER TO EXECUTE A SEQUENCE AND
HANDLES ALL RETURN CODES FROM THE MANAGER

INPUTS FROM COMMON:

XE(10) SERSTA, XE(11) SEREND, XE(12) SERPTA,
XB(250) SERLEN, XB(251) SECTAB, XE(13) FLGTAB

OUTPUTS TO COMMON:

XE(6) SUBSTA, XE(10) SERSTA, XE(12) SERPTA,
XE(13) INTNAM, XE(16) PRCHAM, XB(235) REIND

COMMON USED:

EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA),
+ (XE(10), SERSTA), (XE(11), SEREND),
+ (XE(12), SERPTA), (XE(13), INTNAM),
+ (XE(16), PRCHAM), (XE(23), REBUFP),
+ (XB(235), REIND), (XB(249), SERNO),
+ (XB(250), SERLEN), (XB(251), SECTAB),
+ (XS(1), IPARM), (XS(13), FLGTAB)

FDS ROUTINES USED:

XREQ, XREPORT, XRIG, XRMV, XRMSG,

RSET, XVPAM, XXDEF, XXTMP

RTE ROUTINES USED:

AMPAR
1 BEGIN XXXE
2 INITIALIZE RESET INDEX TO ZERO
3 IF TABLE FLAG SAYS SETTAB IN XE THEN
4 CALL XVPM WITH SEgb # START AND END
5 ELSE - AM BEING CALLED TO EXECUTE A TEMPORARY SEQUENCE TABLE
6 CALL XVPM WITH SEQUENCE POINTER AS START AND END
7 ENDIF
8 CALL RMPAR TO RETRIEVE RETURN PARAMETERS
9 IF THE MANAGER DETECTED A \1 ERROR TRYING TO EXECUTE THE SEQUENCE THEN
10 SET SEQUENCE POINTER TO SEQUENCE \1 IN ERROR
11 FIND ENTRY IN ERROR AND SAVE IN XE
13 :INT1: INTERFACE TABLE NOT SPECIFIED
14 CALL XDEF TO READ UP DEFAULT TABLE
15 CALL XTMP TO SET UP A TEMPORARY ENTRY
16 PERFORM XXXE TO EXECUTE **NO RETURN**
17 :ERR1: SET MESSAGE TO INTERFACE TABLE NOT FOUND
18 :INT2: INTERFACE TABLE NOT COMPLETE
19 SET SUBSTATE TO INTERFACE TABLE EDITOR
20 CALL XRTN TO RETURN TO EXEC **NO RETURN**
21 :ERR2: SET MESSAGE TO INT TABLE NOT FOR PROCESSOR
22 :ERR3: SET MESSAGE TO VERSIONS DO NOT MATCH
23 :RESET: RESET REQUESTED ON TERMINATION
24 CONVERT SEQUENCE \# TO INDEX
25 SAVE INDEX AND SEQUENCE NUMBER IN COMMON
26 EXIT XXXE
27 :ERR4: SET MESSAGE TO RESET SEGB NOT FOUND
28 :ERR5: SET MESSAGE TO PROCESSOR ABENDED
29 :ERR6: SET MESSAGE TO XMA OVERFLOW
30 ENDCASE
31 CALL XRMSG TO DISPLAY ERROR
32 FORMAT SEQUENCE ENTRY INTO ASCII
33 CALL XRMSG TO DISPLAY SEQUENCE TABLE ENTRY IN ERROR
34 IF MODE IS SEMI-AUTO AND ENTRY IS NOT AN OVERRIDE THEN
35 RESET OLD INDEX TO RE-EXECUTE THIS ENTRY
36 ENDIF
37 IF MODE IS AUTO THEN
38 SET MASTER STATE TO ZERO
39 CALL XRTN TO ABORT SEQUENCE *** NO RETURN ***
40 ENDIF
41 ENDIF
42 RETURN
43 END XXXE
371 1 C0************
372 1 C00
373 1 C00  PROCEDURE FOR CALLING XXMAN:
374 1 C00
375 1 C00     CALL XXMAN
376 1 C00
377 1 C00************
378 1 C01
379 1 C01  XXMAN IS THE MANUAL EXECUTION CONTROLLER
380 1 C01
381 1 C00************
382 1 C02
383 1 C02  INPUTS FROM COMMON:
384 1 C02
385 1 C02     XE(5) MASSTA, XB(249) SEENO
386 1 C02
387 1 C00************
388 1 C03
389 1 C03  OUTPUTS TO COMMON:
390 1 C03
391 1 C03     XE(10) SERSTR, XE(11) SEGEND, XB(249) SEENO,
392 1 C03     XB(250) SERLEN, XB(251) SEQTAB, XS(13) FLGTAB
393 1 C03
394 1 C00************
395 1 C05
396 1 C05     COMMON USED:
397 1 C05
398 1 C05     EQUIVALENCE (XE(5), MASSTA), (XE(10), SERSTR),
399 1 C05     + (XE(11), SEGEND),
400 1 C05     + (XB(249), SEENO), (XB(250), SELEN),
401 1 C05     + (XB(251), SEQTAB), (XS(13), FLGTAB)
402 1 C05
403 1 C05  FDS Routines Used:
404 1 C05
405 1 C05     XREX, XMNSG, XTCOM, XXE,
406 1 C05     XDXEC, XXDEF, XXTST, XXTMP
407 1 C05
408 1 C05
409 1 C05  RTE Routines Used:
410 1 C05
411 1 C05     IOR
412 1 C0************
BEGIN XXMAN
  IF ENTRY IS FROM A DIRECTIVE THEN
    SET HASSTA TO INDICATE RE-ENTRY
    DO UNTIL PERCENT IS ENTERED
      :PROMPT: CALL XTCOM TO PROMPT FOR PRNAME, "ITNAME"
      IF PERCENT IS NOT ENTERED THEN
        ERROR IF CR ENTERED TO :PROMPT:
        CALL XXDEC TO DECODE RESPONSE
        ERROR IF INVALID RESPONSE TO :PROMPT:
        SET SER #5 IN ZE TO 0 ZEROS
        SET #ENTRIES IN SERTAC IN XE TO 1
        CALL XXSTO TO STORE SEQUENCE TABLE
        IF IT NAME IN SERTAB IS initab THEN
          PROCESSOR REQUIRES AN IT THEN
            CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
          ENDEF
          CALL XXxEZ TO EXECUTE SECTAB
        ENDEDO
        ELSE_ AM BEING REENTERED FROM INTERFACE TABLE EDITOR
          CALL XXMP TG SET UP TO EXECUTE A TEMPORARY TABLE
          PERFORM XXMAN **NO RETURN**
          ENDEDF
          RETURN
          :PRMERR: CALL XRMSG TO DISPLAY ERROR
        PERFORM XXMAN **NO RETURN**
      ENDEDF
    ENDF
  END XXMAN
**FORTRAN CALLING SEQUENCE:**

```
CALL XXSEM
```

**INPUTS IN COMMON:**

```
XE(5) NASSTA, XE(10) SERSTA, XE(11) SEGEND, XE(12) SEQ'TR,
XE(140) TATEND, XB(1) NOPROC, XB(2) LIBD,
XB(245) SERNO, XB(230) SERLEN, XB(231) SERTAB
```

**OUTPUTS IN COMMON:**

```
XE(5) NASSTA, XE(13) FLGTAB
```

**INTERNAL COMMON USED:**

```
XE(139) EXEND - ENDING SEQUENCE NUMBER USED TO
TERMENATE SEQUENCE
XE(141) CURIND - INDEX TO THE CURRENT ENTRY BEING EXECUTED
```

**COMMON USED:**

```
+ (XE(10), SERSTA), (XE(11), SEGEND),
+ (XE(12), SERSTA), (XE(13), SEGEND),
+ (XE(141), CURIND), (XE(140), TATEND),
+ (XE(145), COMPS), (XB(245), SERNO),
+ (XB(235), RESIND), (XB(236), ASCENT), (XB(240), SERNO),
+ (XB(250), SERTAB), (XE(13), FLGTAB)
```

**FDSC ROUTINES USED:**

```
XRCPR, XREX, XRSI, XXMDV,
XXMSG, XXSET, XXDEC, XXDEF, XXEXF,
XXSTO, XXTRP, XXCM
```
1 BEGIN XXSEM
2 IF ENTRY IS FROM A DIRECTIVE THEN
3 SET MASTER STATE TO SAT REENTRY
4 DO FOR # ENTRY, IN SEQUENCE TABLE
5 SEARCH LIST FOR PROCESSOR NAME
6 ERROR IF NOT FOUND TO :ERR1:
7 MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
8 ENDIF
9 IF SAVE ORIGINAL ENDING SEQUENCE #
10 SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
11 SET PLIST TO SAT EXECUTE ENTIRE SEQTAB
12 CALL XISTO TO STORE SEQTAB IN AWAY
13 ELSE (I AM BEING REENTERED FROM INTE)
14 CALL XXTRP TO SET UP TEMPORARY EXECUTIN
15 CALL XERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQTAB
16 IF RESET WAS REQUESTED THEN
17 RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
18 ELSE
19 EXIT XXEMT IF TERMINAL ENTRY WAS JUST EXECUTED
20 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
21 ENDIF
22 ENDIF
23 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
24 CALL XTCOM TO PROMPT USER WITH CURRENT ENTRY
25 EXIT XXEMT IF RESPONSE IS X
26 IF RESPONSE IS CR THEN
27 IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
28 CALL XXDEF TO READ UP DEFAULT TABLE
29 ENDIF
30 SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
31 SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
32 CALL XERE TO EXECUTE
33 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
34 ELSE
35 IF RESPONSE WAS A SEQUENCE # THEN
36 ERROR IF NUMBER IS ZERO TO :ERR1:
37 SEARCH SEQUENCE TABLE FOR SEQUENCE #
38 ERROR IF NUMBER IS NOT FOUND TO :ERR1:
39 SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
40 ELSE
41 IF RESPONSE IS AN AMPSAND THEN
42 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
43 ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
44 CALL XXDEC TO DECODE PROCESSOR NAME, IT NAME RESPONSE
45 IF R. RESPONSE IS VALID THEN
46 CALL XISTO TO STORE OVERRIDING ENTRY
47 SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
48 SET CURRENT SEQUENCE # TO FIRST ENTRY
49 ENDIF
50 ENDIF
51 ENDIF
52 ENDIF
53 ENDXXSEM
54 EXIT XXSEM
55 EXIT XXSEM
56 EXIT XXSEM
57 EXIT XXSEM
58 EXIT XXSEM
59 EXIT XXSEM
60 EXIT XXSEM
61 EXIT XXSEM
600 1 BEGIN IXST0
610 2 SET CLASS NUMBER TO ZERO
611 3 IF TABLE FLAG SAYS STORE ENTIRE TABLE THEN
612 4 CALL EXEC TO WRITE ENTIRE TABLE
613 5 SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN XR
614 6 CALL IXMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER
615 7 ELSE
616 8 CALL EXEC TO WRITE ONLY FIRST ENTRY
617 9 SET LENGTHS IN REQUEST BUFFER TO 7 WORDS
618 10 CALL IXMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER
619 11 ENSIF
620 12 CALL USER TO REQUEST MANAGER TO STORE .SORKB
621 13 IF RETURN CODE IS NOT ZERO THEN
622 14 CALL IXMSG TO WRITE SPACE ERROR
623 15 CALL EXEC TO RELEASE CLASS &
624 16 SET MASTER STATE TO ZERO
625 17 CALL XERTH TO RETURN TO EXEC **NO RETURN**
626 18 ENSIF
627 19 RETURN
628 20 END IXST0
630 1 C8***********
631 1 CODFORTAM CALLING SEQUENCE:
632 1 CD0CALL XHTMP
633 1 CDO
634 1 CD0
635 1 CD0
636 1 CD0
637 1 CD0XHTMP SETS UP A ONE ENTRY SEQUENCE TABLE USING BINTAB AND
638 1 CD0STORES IT IN THE AMA TO EXECUTE WHEN EXECUTION WITH A
639 1 CD0TEMPORARY ENTRY IS NECESSARY
640 1 CD0
641 1 CD0
642 1 CD0
643 1 CD0
644 1 CD0
645 1 CD0INPUTS FROM COMMON:
646 1 CD0XE(12) SEPRTR, XB(250) SELEN, XB(251) SERTAB
647 1 CD0
648 1 CD0
649 1 CD0OUTPUTS TO COMMON:
650 1 CD0XE(6) SUBSTA, XB(249) SEMSNO, XB(250) SELEN,
651 1 CD0XB(251) SERTAB, X5(13), FLG T A B
652 1 CD0
653 1 CD0
654 1 CD0
655 1 CD0
656 1 CD0COMMON USED:
657 1 CD0
658 1 CD0
659 1 CD0EQUIVALENCE (XE(6), SUBSTA), (XE(12), SEPRTR),
660 1 CD0(XE(19), SEPRTR), (XE(20), REBUF),
661 1 CD0(XE(249), SEMSNO), (XE(250), SELEN),
662 1 CD0(XE(251), SERTAB), (XE(6), TMPTAB),
663 1 CD0(X5(13), FLG T A B)
664 1 CD0
665 1 CD0
666 1 CD0FDS ROUTINES USED:
667 1 CD0XRD, XRMOV, XRST0
668 1 CD0
669 1 CD0
670 1 CD0RTE ROUTINES USED:
671 1 CD0EXEC
672 1 CD0
673 1 CD0***********
ASSGN - DATA ASSIGNMENT PROCESSOR
- SCHEDULED BY FDS

ASSGN ALLOWS THE FDS USER TO COMPUTE VALUES AND STORE THEM IN
AN EXISTING DATA ELEMENT IN THE AWA. ASSGN SUPPORTS EXTENDED
FORTRAN TYPE MIXED-MODE EXPRESSIONS AND FUNCTIONS AND ALLOWS
REPETITIVE EVALUATIONS IN ORDER TO COMPUTE AND STORE MULTIPLE
VALUES

INPUTS FROM THE MANAGER:

LU - LOGICAL UNIT OF USER'S TERMINAL
DEBUG - FLAGS FOR DEBUG

INPUTS FROM THE INTERFACE TABLE:

EXP - SYMBOLIC STRING CONTAINING DATA ASSIGNMENT
(SEE BELOW FOR BACKUS-NAUR DEFINITION OF VALID
SYNTAX)

OUTPUTS TO THE AWA:

THE COMPUTED VALUE(S) IS STORED INTO THE SPECIFIED DATA
ELEMENT

INTERNAL VARIABLES:

BLANK COMMON - ASGCMM DIMENSIONED BY 2300 WORDS DEFINED AS
FOLLOWS:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIMENSION</th>
<th>START</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARMS</td>
<td>5, 1</td>
<td></td>
<td>PARMS(1)= LU, PARMS(7)= DEBUG FLAGS</td>
</tr>
<tr>
<td>TOKERS</td>
<td>32, 6</td>
<td></td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STWIDE</td>
<td>1, 38</td>
<td></td>
<td>SYMBOL TABLE WIDTH</td>
</tr>
<tr>
<td>STLNG</td>
<td>1, 39</td>
<td></td>
<td>SYMBOL TABLE LENGTH</td>
</tr>
<tr>
<td>LASTSY</td>
<td>1, 40</td>
<td></td>
<td>LAST SYMBOL TABLE ENTRY DEFINED</td>
</tr>
</tbody>
</table>
| SYNTAX| 12, 81, 41 |       | SYMBOL TABLE (WORDS 1-8-TOC ENTRY OR
APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR
SUBSCRIPTED DATA ELEMENT)
| SUBC | 1, 24      |       | SUBSCRIPTED STRING (EXP)                         |
| SSTREG| 247, 1013  |       | RESULT STACK USED DURING POST-FIX STRING EVALUATION (EACH EM-
FIX STRING EVALUATION (EACH EM-
<p>| RESULT| 4, 35, 1260|       | 1, 2, 3 FIXED DATA                               |
| -1 Symbol Table Index |</p>
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>NAME</th>
<th>DESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>OPRND5</td>
<td>4,9</td>
</tr>
<tr>
<td>1400</td>
<td>OPRND3</td>
<td>4,9</td>
</tr>
<tr>
<td>1436</td>
<td>SCRTCH</td>
<td>4,9</td>
</tr>
<tr>
<td>1472</td>
<td>REGRST</td>
<td>8</td>
</tr>
<tr>
<td>1480</td>
<td>OPINFO</td>
<td>7</td>
</tr>
<tr>
<td>1487</td>
<td>NUMWDS</td>
<td>1</td>
</tr>
<tr>
<td>1488</td>
<td>DATTPS</td>
<td>9</td>
</tr>
<tr>
<td>1497</td>
<td>RSLTPY</td>
<td>1</td>
</tr>
<tr>
<td>1498</td>
<td>EXPRTR</td>
<td>1</td>
</tr>
<tr>
<td>1499</td>
<td>CLSRN</td>
<td>1</td>
</tr>
<tr>
<td>1500</td>
<td>CLSRN</td>
<td>1</td>
</tr>
<tr>
<td>1501</td>
<td>MAPWDS</td>
<td>9</td>
</tr>
<tr>
<td>1510</td>
<td>DECLAS</td>
<td>1</td>
</tr>
<tr>
<td>1511</td>
<td>RNGKR</td>
<td>4,4</td>
</tr>
<tr>
<td>1527</td>
<td>POLISH</td>
<td>161</td>
</tr>
<tr>
<td>1688</td>
<td>SYNTAX</td>
<td>7,40</td>
</tr>
<tr>
<td>1968</td>
<td>FNCDBL</td>
<td>7,36</td>
</tr>
<tr>
<td>2220</td>
<td>MBBUFF</td>
<td>64</td>
</tr>
</tbody>
</table>

**NOTE:** STACKS USED IN THE ASSGN PROCESSOR ARE SIZED FOR

-2 DISPLACEMENT
-3 CHARACTER STRING INDEX

STACK CONTAINING OPERANDS FOR
FUNCTIONS AND ARITHMETIC OPERATIONS AND RESULTS FOR STORING
(EA. ENTRY: WORDS 1-3 CONTAIN VALUE; WORD 4 = DATA TYPE)
SIZED FOR C72 CHARACTER STRING

MARY REQUEST FOR XPRE

CONTENTS OF FNCDBL OR SYNTAX FOR
FUNCTION OR MATHEMATICAL OPERATIONS BEING EVALUATED
NUMBER OF WORDS TO BE STORED IN
OBJECT DATA ELEMENT
DATA TYPES
RESULT STACK POINTER
POST-FIX STRING (POLISH) POINTER
XPRE OPTION WORD FOR QUEUE REQUEST AND CLOSE BUFFER - NO
DATA TRANSFERED
XPRE OPTION WORD FOR QUEUE REQUEST, CLOSE BUFFER AND TRANSFER DATA
NUMBER OF WORDS PER LOGICAL UNIT OF DATA FOR EACH DATA TYPE
DATA ELEMENT CLASS
DATA ELEMENT, END RANGE, INCREMENT AND SYMBOL INDEX FOR EACH RANGE SPECIFICATION
POST-FIX REPRESENTATION OF EXPRESSION
SYNTAX TABLE FOR VALIDITY TESTS ON EXPRESSION (SEE BELOW)
FUNCTION TABLE CONTAINING DATA REQUIREMENTS FOR EACH FUNCTION (SEE BELOW)
XPRE BUFFER

THE MAXIMUM POSSIBLE AND OVERFLOW IS NOT TESTED

**TABLE DEFINITIONS:**
<table>
<thead>
<tr>
<th>Input Token</th>
<th>Output Token</th>
<th>Input Priority</th>
<th>Output Priority</th>
<th>Length</th>
<th>Requirement</th>
<th>Type</th>
<th>Precedent</th>
<th>Valid Precedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note:**
- Input Priority: 1 indicates a higher priority than 0.
- Output Priority: 1 indicates a higher priority than 0.
- Length: 1 indicates a single character, 2 indicates two characters, and so on.
- Requirement: 1 indicates a requirement, 0 indicates no requirement.
- Type: 1 indicates the type of token.
- Precedent: 1 indicates precedence, 0 indicates no precedence.

**Reproducibility of the original page is poor.**
201 1 C05 Routines Referenced - RMPAR, XPGET, XRMOV, XI?51, XIPS2, XPXIT
202 1 C05
203 1 C05 BACKUS-NAUR LANGUAGE DEFINITION
204 1 C05
205 1 C05 <assignment> ::= <replacement> <range> //
206 1 C05 <replacement> ::= <non-numeric-de> = <non-numeric-de> //
207 1 C05 <non-numeric-de> ::= "character string" //
208 1 C05 <variable> ::= free data element
209 1 C05 <non-numeric-de> ::= free data element<subscript list> //
210 1 C05 character data element //
211 1 C05 character data element<subscript list>
212 1 C05 <evaluation> ::= <variable> = <expression> //
213 1 C05 free data element = <expression> //
214 1 C05 free data element<subscript list> = <expression>
215 1 C05 <variable> ::= fixed data element //
216 1 C05 fixed data element<subscript list>
217 1 C05 <subscript list> ::= <subscript list>,<expression> //
218 1 C05 <expression> ::= <expression> + <additive-operator> <term> //
219 1 C05 <additive-operator> ::= \+ \- 
220 1 C05 <term> ::= <term> * <multiplicative-operator> <factor> //
221 1 C05 <factor> ::= <factor> ** <power> /<expression> //
222 1 C05 <power> ::= <expression> /<operator>
223 1 C05 <operator> ::= <function-name> <function-list> <rb> //
224 1 C05 <function-list> ::= <function-list>,<expression> //
225 1 C05
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>260</td>
<td>CD5</td>
<td><code>CONSTANT</code> ::= <code>INTEGER</code> //</td>
</tr>
<tr>
<td>261</td>
<td>CD5</td>
<td><code>SINGLE PRECISION REAL</code> //</td>
</tr>
<tr>
<td>262</td>
<td>CD5</td>
<td><code>DOUBLE PRECISION REAL</code></td>
</tr>
<tr>
<td>263</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>RANGE</code> <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>264</td>
<td>CD5</td>
<td><code>&lt;LIMITS&gt;</code> ::= <code>INDEX</code>=<code>INTEGER</code>,<code>INTEGER</code></td>
</tr>
<tr>
<td>265</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>266</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>267</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>268</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>269</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
<tr>
<td>270</td>
<td>CD5</td>
<td><code>RANGE</code> ::= <code>&lt;LIMITS&gt;</code> //</td>
</tr>
</tbody>
</table>
1 BEGIN ASSGN
2 INITIALIZE COMMON
3 CALL XIPST TO BUILD POST-FIX STRING
4 CALL XPSS2 TO EVALUATE EXPRESSION AND STORE VALUE(S)
5 CALL XPRIT TO EXIT PROFESSOR
6 END ASSGN
| 279 | 1 CD  | **DDSP - DATA BOX DISPLAY PROCESSOR** |
| 280 | 1 CD 0 |
| 281 | 1 CD 0 |
| 282 | 1 CD 0 |
| 283 | 1 CD 0 |
| 284 | 1 CD 0 |
| 285 | 1 CD 0 **- SCHEDULED BY FDS** |
| 286 | 1 CD 1 |
| 287 | 1 CD 1 |
| 288 | 1 CD 1 |
| 289 | 1 CD 1 |
| 290 | 1 CD 1 |
| 291 | 1 CD 1 |
| 292 | 1 CD 1 |
| 293 | 1 CD 1 |
| 294 | 1 CD 1 |
| 295 | 1 CD 1 |
| 296 | 1 CD 1 |
| 297 | 1 CD 1 |
| 298 | 1 CD 1 |
| 299 | 1 CD 1 |
| 300 | 1 CD 1 |
| 301 | 1 CD 1 |
| 302 | 1 CD 1 |
| 303 | 1 CD 1 |
| 304 | 1 CD 1 |
| 305 | 1 CD 1 |
| 306 | 1 CD 1 |
| 307 | 1 CD 1 |
| 308 | 1 CD 1 |
| 309 | 1 CD 1 |
| 310 | 1 CD 1 |
| 311 | 1 CD 1 |
| 312 | 1 CD 1 **- SCHEDULED BY FDS** |
| 313 | 1 CD 2 |
| 314 | 1 CD 2 |
| 315 | 1 CD 2 |
| 316 | 1 CD 2 |
| 317 | 1 CD 2 |
| 318 | 1 CD 2 |
| 319 | 1 CD 2 |
| 320 | 1 CD 2 |
| 321 | 1 CD 2 **- SCHEDULED BY FDS** |
| 322 | 1 CD 2 |
| 323 | 1 CD 2 |
| 324 | 1 CD 2 |
| 325 | 1 CD 2 |
| 326 | 1 CD 2 |
| 327 | 1 CD 2 |
| 328 | 1 CD 2 |
| 329 | 1 CD 2 |
| 330 | 1 CD 2 |
| 331 | 1 CD 2 |
| 332 | 1 CD 2 |
| 333 | 1 CD 2 |
| 334 | 1 CD 2 |
| 335 | 1 CD 2 |
| 336 | 1 CD 2 |
| 337 | 1 CD 2 **- SCHEDULED BY FDS** |

**DDSP PROCESSES THE DATA PRODUCED BY SCANNING FUNCTIONAL PROCESSORS. IT GENERATES A DIGITAL MATRIX DISPLAY CONTAINING THE VALUES OF ANY PARAMETER OR PAIR OF PARAMETERS OVER THE SCANNED SPACE.**

**BECAUSE OF THE SIZE OF THIS PROCESSOR, IT WAS DIVIDED INTO THREE OVERLAYS AS FOLLOW**

- **XDIM - READS AND EDITS INTERFACE TABLE**
- **XDIP1 - READS ORdre A, VALIDATES NAMES IN INTERFACE TABLE AGAINST NAMES IN DATABOX SCAN. WITH SUBROUTINE XDOM**
- **IT DEVELOPS CONSTRAINT MADS FOR ARRAYS**
- **XDIP2 - PROMPTS USER IF REQUIRED AND OUTPUTS REQUESTED PAGE ARRAYS OF UP TO TWO VARIABLES TO THE SPECIFIED LU DEBSITE WITH SUBROUTINE XDOO**

**DDSP MERELY CALLS THESE OVERLAYS IN THE PROPER ORDER AND EXITS**

**INPUTS TO DDSP FROM INTERFACE TABLE**

- **DATBOX - DATA BOX FILE NAME**
- **NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER**
- **KEPS - DISPLAY VARIABLE SCALE LIST SET UP BY USER**
- **VIODEF - CONSTRAINT VARIABLE DEFINITION LIST**

**INPUTS TO DDSP FROM ORDE FILE**

**RECORD 1**

- (1) - NAME OF FDS PROCESSOR CREATING FILE
- (4) - INTERFACE TABLE VARIABLE NAME FOR THIS FILE
- (7) - NAME OF FDS PROCESSOR UPDATING FILE (3 ASCII WORDS OF BLANKS)
- (10) - INTERFACE TABLE VARIABLE NAME FOR THIS U-DATE (3 ASCII WORDS OF BLANKS)

**RECORD 2**

- (1) - NO OF ENTRIES IN SUMMARY TABLE
- (2) - X SCAN VARIABLE (6 CHAR)
- (5) - X FI. ST SUBSCRIPT (INT OR ZERO)
- (6) - XSECOND SUBSCRIPT (INT OR ZERO)
- (7) - X UNITS (6 CHAR)
- (10) - X CENTROID (REAL)
DEPANDANT VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
- DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES)
- INCLUDING ERROR FLAG WHICH IS FIRST VALUE
- IN SUMMARY TABLE

OUTPUT FROM DBDSP
- DISPLAY IS OUTPUT TO LU IDENTIFIED IN THE INTERFACE TABLE

DEPANDANT VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
- DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES)
- INCLUDING ERROR FLAG WHICH IS FIRST VALUE
- IN SUMMARY TABLE

OUTPUT FROM DBDSP
- DISPLAY IS OUTPUT TO LU IDENTIFIED IN THE INTERFACE TABLE
397   1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)   DDSGP
398   1 CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)  DDSGP
399   1 CD 4 YSCANN - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR) DDSGP
400   1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)  DDSGP
401   1 CD 4 ZTABLE - TABLE IN COMMON FOR SNTAB VARIABLE NAMES AND UNITS DDSGP
402   1 CD 4 NAMVUL - UNITS LIST FOR VARIABLES SCANNED BY SIAN/ENDSCH DDSGP
403   1 CD 4 SUNTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD DDSGP
404   1 CD 4 PRMS - COMMUNICATION BUFFER FOR RMPAR - LU, USER ID, FLAGS DDSGP
405   1 CD 4 LU - LOGICAL UNIT # FOR SPROM CALLING SEQUENCE - USER LOCATNM DDSGP
406   1 CD 4 LURSP - DDS/P WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU DDSGP
407   1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH SPROM DDSGP
408   1 CD 4 DEBUG DDSGP
409   1 CD 4 SELECT - SELECT =0 PROMPT ; SELECT NOT 0 RUN ALL DISPLAYS TO O/P DDSGP
410   1 CD 4 WITHOUT PROMPT DDSGP
411   1 CD 4 CARTAG - CARTRAG USED TO LOCATE DATA BOX DDSGP
412   1 CD 4 CVLUE DDSGP
413   1 CD 4 INCB DDSGP
414   1 CD 4 ****** USES ROUTINES DDSGP
415   1 CD 5 DDSGP
416   1 CD 5 DDSGP
417   1 CD 5 DDSGP
418   1 CD 5 DDSGP
419   1 CD 5 DDSGP
420   1 CD 5 DDSGP
421   1 CD ****** DDSGP
CO************
DEFIN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

CO************
DEFIN ALLOCATES DATA ELEMENTS IN THE AWA THAT WERE SPECIFIED
BY THE PARAMETER KEYWORD DEFINE. IF THE DATA ELEMENT ALREADY
EXISTS, IT IS DELETED AND REALLOCATED. DATA ELEMENTS ARE
INITIALIZED TO ZERO (CHARACTER STRINGS TO BLANKS).

CO************
INPUTS FROM THE MANAGER:
LD - LOGICAL UNIT OF THE USER'S TERMINAL
DEBO - FLAGS FOR DEBUG

CO************
INPUTS FROM THE INTERFACE TABLE:
DEFINE - SYMBOLIC STRING CONTAINING DATA ELEMENT NAME(S),
OPTIONAL I AND J DIMENSIONS AND A REQUIRED TYPE

CO************
OUTPUTS TO THE AWA:
SET OF DATA ELEMENT(S) REQUESTED

CO************
INTERNAL VARIABLES:
INBUF - INTERFACE TABLE HEADER
ISLENG - LENGTH OF SYMBOLIC STRING
ITOKEH - POSITION WITHIN THE SYMBOLIC STRING
MAWARK - BUFFER FOR VALD NAMES TO BE ALLOCATED
MBUFF - BUFFER AREA FOR APET AND XPAR USE
NEKT - NUMBER OF ENTRIES IN THIS AWA REQUEST
NAME - NUMBER OF NAMES IN SYMBOLIC STRING
NEXTPA - TO ELEMENTAL NAME FOR NEXT NAME
SITOKEH - SYMBOLIC STRING INPUT TO DEFINE

CO************
EXTERNAL ROUTINES USED:
EXEC, IANM, KCVT, XPARF, XPAR,
XPREB, XPAR, XPAR, XPAR,
XPREF, XPAR, XPAR, XPAR

CO************
1 BEGIN DEF
2   CALL RPMPAR TO GET LU AND DEBUG FLAGS
3   CALL APGET TO RETRIEVE SYMBOLIC STRING
4   GET STRING LENGTH FROM INTERFACE TABLE HEADER
5   STARTSEARCH WHILE Token-Position .LT. string-Length, OR
6      WHILE Current-Token NE. END-OF-STRING
7   PERFORM ZOMP TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
8   PERFORM ZIDPM TO PROCESS THE CURRENT NAME
9   EXIT IF THERE WAS AN ERROR RETURN FROM ZIDPM
10  CALL ZIMSG TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
11  SET PROCESSOR RETURN CODE TO ABEND
12  ELSE
13      INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
14 ENDLOOP
15   SET PROCESSOR RETURN FOR NORMAL EXIT
16   ENDSrch
17   SET OPTION SO XPR# WILL DO A QUEUE REQUEST
18   DO FOR NUMBER OF REQUESTS IN REQUEST BUFFER (NANAB)
19      IF THIS IS LAST REQUEST, THEN
20         SET OPTION TO CLOSE REQUEST BUFFER
21         ENDIF
22         COMPUTE INDEX TO THIS REQUEST
23         CALL XPR# TO QUEUE THIS REQUEST
24         ENDDO
25         CALL XPR# TO RETURN TO THE MANAGER
26 1 END DEF
BEGIN XIDNN

1. SET NEXT NAME TO ZERO
2. DO WHILE NEXT NAME IS ZERO
3. CALL XZDF2 TO FIND THE NEXT LEFT PARENTHESE
4. IF THERE ARE NO SUBSCRIPTS (COMMA PRECEDES LEFT PARENTHESIS) THEN
5. SET NEXT NAME TO COMMA POSITION + 1
6. ELSE
7. CALL XZDF2 TO FIND THE NEXT RIGHT PARENTHESIS
8. SET STARTING POSITION TO RIGHT PARENTHESIS POSITION
9. IF STARTING POSITION > END OF STRING THEN
10. SET STARTING POSITION = END OF STRING

11. ENDIF
12. ENDIF
13. ENDDO
14. END XIDNN
527 1 BEGIN XIDPM
528 2 SET IDIM AND JDIM TO 1
529 3 IF TOKEN IS NOT A NAME THEN
530 4 SET ERROR CODE
531 5 ELSE
532 6 MOVE NAME INTO REQUEST
533 7 INCREMENT TO NEXT TOKEN
534 8 IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREN) THEN
535 9 INCREMENT TO NEXT TOKEN
536 10 IF TOKEN IS NOT AN INTEGER OR
537 11 TOKEN IS NOT ZERO THEN
538 12 CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
539 13 EXIT TO :PNERR1:
540 14 ENDIF
541 15 SET IDIM TO THIS TOKEN
542 16 INCREMENT TO NEXT TOKEN
543 17 IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
544 18 INCREMENT TO NEXT TOKEN
545 19 IF TOKEN IS NOT AN INTEGER OR
546 20 TOKEN IS NOT ZERO THEN
547 21 CALL XIMSG TO DISPLAY ERROR "INVALID JOIN"
548 22 EXIT TO :PNERR1:
549 23 ENDIF
550 24 SET JDIM TO THIS TOKEN
551 25 INCREMENT TO NEXT TOKEN
552 26 ENIF
553 27 IF TOKEN IS NOT A RIGHT PAREN THEN
554 28 CALL XIMSG TO DISPLAY ERROR "INVALID SUBSCRIPT DELIMITER"
555 29 EXIT TO :PNERR1:
556 30 ENDIF
557 31 INCREMENT TO NEXT TOKEN
558 32 IF TOKEN IS NOT A BEGIN TYPE FIELD SLASH THEN
559 33 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
560 34 EXIT TO :PNERR1:
561 35 ENDIF
562 36 INCREMENT TO NEXT TOKEN
563 37 IF TOKEN IS NOT A NAME THEN
564 38 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
565 39 EXIT TO :PNERR1:
566 40 ENDIF
567 41 INCREMENT TO NEXT TOKEN
568 42 STARTSEARCH FOR ALL VALID TYPES
569 43 EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
570 44 SET TYPE AND CLASS IN REQUEST
571 45 COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
572 46 IF SIZE IS TOO LARGE (>1200) WORDS THEN
573 47 CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
574 48 EXIT TO :PNERR1:
575 49 ENDIF
576 50 ENDDO
577 51 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
578 52 EXIT TO :PNERR1:
579 53 ENDSEARCH
580 54 INCREMENT TO NEXT TOKEN
581 55 IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
582 56 NEXT TOKEN IS NOT A COMMA THEN
583 57 CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
BUILD REQUEST TO DELETE/VERIFY ABSENT THIS DATA ELEMENT

BUILD REQUEST TO ALLOCATE THIS DATA ELEMENT

END

EXIT XIDPN

;PMEPR1:
CALL X2MSG TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING

END XIDPN
ENDSC is an FDS processor scheduled by the manager.

ENDSC processor is used in conjunction with SCAN only. It marks the end of the series of processors to be scanned.

It has no interface table.

Inputs from the Manager:
- LU - Logical Unit of the User's Terminal
- USEID - User ID Code
- FLAGS - Flags for Debug

Inputs from AWA:
- &SCNTB - Scan Control Table Created by SCAN

Outputs to Manager:
- XZRET - Return Code to Manager

Outputs to AWA:
- XSCAN - New Value for X Scan Variable
- YSCAN - New Value for Y Scan Variable

RTE Routines Used:
- CLOSE, EXEC, FLOAT, KCVT, POINT,
- READ, RMPAR, WRIT

FDS Routines Used:
- XPEIT, XREXT, XRMOV, XUDBG, XVPAM,
641 1 BEGIN ENDSC
642 2 CALL RMPIAR TO GET INPUTS FROM MANAGER
643 2 SAVE REQUEST TO RETRIEVE $SCNTB FROM ANA
644 2 CALL XVPAM TO REQUEST ANA MANAGEMENT
645 2 ERREXIT IF $SCNTB NOT FOUND TO :ERR4:
646 2 CALL EXEC TO READ IN $SCNTB
647 2 SAVE REQUEST TO RETRIEVE SUMTAB IN REGBUF
648 2 CALL XVPAM TO REQUEST ANA MANAGEMENT
649 2 ERREXIT IF NOT FOUND TO :ERR4:
650 2 CALL EXEC TO READ IN SUMTAB
651 2 IF SUMMARY TABLE IS LARGER THAN 32 ENTRIES THEN
652 2 SET SIZE OF SUMTAB TO 32 ENTRIES
653 2 ENDF
654 2 CALL WRTSF TO WRITE SUMTAB TO DATBOX
655 2 ERREXIT IF WRTSF ERROR TO :ERR4
656 2 IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
657 2 THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
658 2 CALL READ TO READ HEADER RECORD
659 2 ERREXIT IF READF ERROR TO :ERR4.
660 2 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
661 2 CALL WRTSF TO WRITE UPDATED HEADER
662 2 ERREXIT IF WRTSF ERROR TO :ERR4:
663 2 CALL CLOSE TO CLOSE DATBOX
664 2 SAVE REQUEST TO DEL/VER ABS $SCNTB IN REGBUF
665 2 IF THERE ARE REMAINING SCANS I.HI $SCNTB THEN
666 2 CALL EXEC TO WRITE REMAINING $SCNTB
667 2 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCNTB
668 2 ENDF
669 3 SET RETURN PARAMETER TO NORMAL RETURN
670 3 ELSE
671 3 PERFORM SETXY
672 2 ENDF
673 2 CALL XVPAM TO REQUEST ANA MANAGEMENT
674 2 CALL XPSTI TO TERMINATE WITH RETURN PARAMETERS
675 1 EXIT ENDSC
676 2 :ERR4:
677 2 CALL XMSG TO DISPLAY ERROR
678 2 CALL XPSTI TO ABEND PROCESSOR
679 1 END ENDSC
SCAN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

SCAN PROCESSOR EXECUTES A SERIES OF PROCESSORS ITERATING ON XSCAN AND YSCAN (IF ENTERED) VALUES COMPUTED USING THE CENTROID, THE INCREMENT, AND THE CURRENT STEP NUMBER, CREATING A DATA BOX FILE.

INPUTS FROM MANAGER:
- LU - LOGICAL UNIT OF THE USER'S TERMINAL
- USERID - USER ID CODE
- FLAGS - FLAGS FOR DEBUG
- ENTSUB-DISPLACEMENT OF THIS SCAN ENTRY IN BSEGTB

INPUTS FROM INTERFACE TABLE:
- PROCON-CARTRIDGE # FOR DATA BOX FILE
- SUMTAB-SUMMARY TABLE
- DATBOX-NAME OF DATA BOX ENTERED
- NVAR - NUMBER OF SCAN VARIABLES
- *SCANNAME OF X SCAN VARIABLE
- XUNIT - UNIT OF X VARIABLE
- XCENTR-CENTROID OF X VARIABLE
- ZINC - INCREMENT FOR X
- XSTEPS-NUMBER OF STEPS FOR X
- TSCANNAME OF Y SCAN VARIABLE
- YUNIT - UNIT OF Y VARIABLE
- YCENTR-CENTROID OF Y VARIABLE
- TINCR - INCREMENT FOR Y
- YSTEPS-NUMBER OF STEPS FOR Y

* - ACTUALLY OUTPUTS, ONLY NAME ENTERED IS AN INPUT TO SCAN

OUTPUTS TO MANAGER:
- XRET - RETURN CODE TO MANAGER

OUTPUTS TO WORK AREA (CWA):
- DATBOX-FILE WHERE SUMMARY TABLE IS WRITTEN
- XSCAN - X SCAN VARIABLE
- YSCAN - Y SCAN VARIABLE
- BSCNTB - SCAN CONTROL TABLE (173 WORDS PER ACTIVE SCAN)
- XCHAR NAME OF SUMMARY TABLE
- DATBOX & CHAR QUALIFIED NAME OF DATA BOX FILE
- RESET SEQUENCE NUMBER
- CENTRE CENTROID RECORD NUMBER
- XSCAN NAME OF X VARIABLE
- XDISPLAY DISPLACEMENT FOR X
- XCENTR - CENTROID FOR X
- ZINC - INCREMENT FOR X
BEGIN SCAN
CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XIPAR TO GET PROCON AND @ SCAN VARIABLES
ERREXIT IF @ SCAN VARIABLES < 1 OR > 2 TO :ERR3:
GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
ERREXIT IF SUMMARY IS A LITERAL TO :ERR3:
ERREXIT IF DISPLACEMENT IS NOT AN ELEMENT BOUNDARY TO :ERR3:
GET DATA BOX NAME FROM INTERFACE TABLE
DO FOR # SCAN VARIABLES
CALL XPATH TO GET NAME AND DISPLACEMENT
COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND IDIM
CALL XIPET TO GET UNITS, CENTROID, IMER, # STEPS
ERREXIT IT # STEPS < 0 OR > 5 TO :ERR3:
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR $SERTB AND $SCNTB
CALL XVPAW TO REQUEST AWA MANAGEMENT
CALL EXEC TO READ IN $SERTB
IF $SCNTB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF $SCNTB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN $SCNTB
ERREXIT IF # SCANS = MAXIMUM ALLOWED (4) TO :ERR3:
ERREXIT IF THIS DATABASE NAME IS ALREADY IN USE TO :ERR3:
ENDIF
ERREXIT IF THIS IS THE LAST ENTRY IN $SERTB TO :ERR3:
GET THE SEQUENCE NUMBER OF THIS SCAN FROM $SERTB
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH $SERTB FOR THE SEQUENCE NUMBER
ERREXIT IF THIS SCAN IS THE LAST ENTRY IN $SERTB TO :ERR3:
IF THIS IS A SEMI-OVERRIDE (2 PROCESSOR NAMES NOT EQUAL) THEN
SET RESET NUMBER TO THIS ENTRY SEQUENCE NUMBER
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
COMPUTE SIZE OF DATBOX FILE = (2 + (2 * XSTEPS + 1) * (2 * YSTEPS + 1) + 1) / 2
COMPUTE CENTROID RECORD NUMBER = SIZE + 3
DO FOR # SCAN VARIABLES
COMPUTE BEGINNING VALUE = (CENT + INCR + FLOAT (CUR STEP))
ENDDO
CALL XIPUT TO STORE DATABOX AND SCAN VARIABLES
CALL CREATE TO CREATE DATABOX FILE
IF FILE ALREADY EXISTS THEN
CALL PURGE TO PURGE FILE
ERREXIT IF PURGE ERROR TO :ERR2:
CALL CREATE TO CREATE FILE
ENDIF
ERREXIT IF CREATE ERROR TO :ERR2:
CALL WRITF TO WRITE HEADER RECORD TO DATBOX
ERREXIT IF WRITE ERROR TO :ERR2:
CALL POSMT TO POSITION FILE TO FIRST DATA RECORD
ERREXIT IF POSMT ERROR TO :ERR2:
CALL EXEC TO WRITE $SCNTB
SAVE REQUEST TO DELETE/VERIFY ABSENT $SCNTB IN REQBUF
SAVE REQUESTS TO ALLOC AND STORE VALUES FOR NEW $SCNTB
CALL XVPAW TO REQUEST AWA MANAGEMENT
ERREXIT IF NO AWA SPACE TO :ERR1:
874 2 CALL A?EXIT TO EXIT NORMALLY
875 1 EXIT SCAN
876 2 :ERR1:
877 2 IF THERE ARE MORE ACTIVE SCANS (# SCANS > 0) THEN
878 3 CALL EXEC TO READ IN NEW BSCNTB
879 3 CALL EXEC TO WRITE OUT ORIGINAL BSCNTB
880 3 SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL BSCNTB
881 2 ENDIF
882 2 :ERR2:
883 2 CALL CLOSE TO CLOSE DATBOX
884 2 CALL PURGE TO PURGE DATBOX
885 2 SET VALUE FOR XPPUT
886 2 SAVE REQUEST TO DELETE DATBOX FROM AWA
887 2 CALL XPNAV TO REQUEST AWA MANAGEMENT
888 2 :ERR3:
889 2 CALL XMSG TO DISPLAY ERROR
890 2 CALL XPDIT TO ABEND SCAN
891 1 END SCAN
FORTRAN CALLING PROCEDURE:

CALL XICHR

XICHR IS USED BY THE ASSGN ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS
FOR CHARACTER-TYPE OBJECT DATA ELEMENTS

INPUTS FROM ASSCOC

LU, SYMTAB, ESTRNG, DATYPS, RSLTPT, CLSTRM, MAPWDS, RESULT

OUTPUTS TO ASSCOC

RESTR, NUMWDS, RSLTPT, OFRND5

EXTERNAL REFERENCES

FD5 - XPREP, XPXIT, XMMOV, XLSSA, XIMSG

RTE - IAMO
925 1 BEGIN XICHR
926 2 SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
927 3 * CASE (RESULT DATA TYPE) :CHRSTR:, :CHRFREE:, :CHRFREE:, :FIXERR:, :FIXERR:
928 2 3 :FIXERR:, :FIXERR:
929 3 :CHRSTR:
930 3 3 DETERMINE # WORDS IN CHARACTER STRING
931 3 3 MOVE CHARACTER STRING TO RESULT LOCATION
932 3 3 IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
933 4 3 BLANK FILL AFTER CHARACTER STRING
934 4 3 ENDIF
935 4 :CHRFREE:
936 3 3 IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
937 4 3 POP DISPLACEMENT FROM RESULT STACK
938 4 3 ELSE
939 4 4 SET DISPLACEMENT = 0
940 4 4 ENDIF
941 4 4 IF RESULT OPERAND IS FREE, THEN
942 4 4 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
943 4 4 ELSE CHARACTER = CHARACTER
944 4 4 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
945 4 4 ENDIF
946 4 4 CALL XPER TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
947 4 4 IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
948 4 4 BLANK FILL REMAINING DATA
949 4 4 ENDIF
950 4 4 ENDIF
951 4 4 ENDIF
952 4 4 ENDIF
953 2 END CASE
954 1 EXIT XICHR
955 2 :FIXERR:
956 2 SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO
957 2 NUMERICAL DATA"
958 2 CALL XING TO OUTPUT MESSAGE TO USER
959 2 CALL XLIST TO LIST SYMBOLIC STRING
960 2 CALL XEXIT TO EXIT PROCESSOR
961 1 END XICHR
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1053</td>
<td><strong>FUNCTION CALLINGSEQUENCE:</strong></td>
</tr>
<tr>
<td>1054</td>
<td>CALL XIDFT (ARRAY, START, END, TOKEN, INDEX)</td>
</tr>
<tr>
<td>1055</td>
<td>XIDFT SEARCHES 'ARRAY' BEGINNING AT 'START' AND ENDING AT 'END' FOR A SPECIFIED 'TOKEN'. IT RETURNS EITHER THE POSITION OF 'TOKEN' OR THE 'END' VALUE IN 'INDEX'.</td>
</tr>
<tr>
<td>1056</td>
<td><strong>INPUTS FROM CALLING SEQUENCE:</strong></td>
</tr>
<tr>
<td>1057</td>
<td>ARRAY - SYMBOLIC STRING TO BE SEARCHED</td>
</tr>
<tr>
<td>1058</td>
<td>END - ENDING INDEX INTO 'ARRAY'</td>
</tr>
<tr>
<td>1059</td>
<td>START - BEGINNING INDEX INTO 'ARRAY'</td>
</tr>
<tr>
<td>1060</td>
<td>TOKEN - TOKEN TO BE SEARCHED FOR</td>
</tr>
<tr>
<td>1061</td>
<td><strong>OUTPUTS TO CALLING SEQUENCE:</strong></td>
</tr>
<tr>
<td>1062</td>
<td>INDEX - POSITION OF 'TOKEN' IN SYMBOLIC STRING IF FOUND, OR 'END' IF NOT FOUND</td>
</tr>
</tbody>
</table>
1083  1 BEGIN  XZDFT
1084  2   SET INDEX TO START
1085  2   DO UNTIL INDEX >= END
1086  3   IF ARRAY(INDEX) = EQ. TOKEN, THEN
1087  4   SET END TO INDEX
1088  3   ELSE
1089  4   CASE ARRAY(INDEX) + 1 (
1090  5   :ADD1: = 0 END OF STRING
1091  5   :ADD2: = 1 INTEGER
1092  5   :ADD3: = 2 REAL
1093  5   :ADD4: = 3 DOUBLE
1094  5   :ADD1: = 4 UNKNOWN
1095  5   :ADD4: = 5 NAME
1096  5   :ADD1: = 6 UNKNOWN
1097  5   :ADD1: = 7 UNKNOWN
1098  5   :CALST: = 8 CHARACTER STRING
1099  5   :ADD1: = 9 UNKNOWN
1100  5   :ADD1: = 10 +
1101  5   :ADD1: = 11 -
1102  5   :ADD1: = 12 *
1103  5   :ADD1: = 13 /
1104  5   :ADD1: = 14 <
1105  5   :ADD1: = 15 >
1106  5   :ADD1: = 16 =
1107  5   :ADD1: = 17 %
1108  5   :ADD1: = 18 \\
1109  5   :ADD1: = 19 \\
1110  5   :ADD1: = 20 ?
1111  5   :ADD1: = 21 \\
1112  5   :ADD1: = 22 )
1113  5   :ADD2: = 23 START OF SYMBOLIC STRING
1114  5   :ADD1: = 24 END OF STRING
1115  5   :ADD1: = 25 %
XDP1 - DATA BOX DISPLAY OVERLAY - PREPARES DATA FOR DISPLAY
- SCHEDULED BY DBDSP

XDP1 READS IN THE SCAN CONTROL INFORMATION AND THE SUMMARY
TABLES PRODUCED BY SCAN/ENDSCAN FOR FURTHER PROCESSING.
IN ADDITION, XDP1 VERIFIES THE DISPLAY DEPENDANT VARIABLE
NAMES AND THE CONSTRAINT VARIABLE NAMES ARE CONTAINED IN THE
DEPENDANT VARIABLE NAME LIST GENERATED DURING THE SCAN.
XDP1 ALSO GENERATES CONSTRAINT MASKS WITH SUBROUTINE XDMK
FOR SUBSEQUENT DISPLAY BY XDP2 AND XDP3.

INPUTS FROM THE DATA BOX

RECORD 1
(1) - NAME OF FDS PROCESSOR CREATING FILE
(2) - NAME OF FDS PROCESSOR UPDATING FILE
(3) - INTERFACE TABLE VARIABLE NAME FOR THIS FILE
(4) - INTERFACE TABLE VARIABLE NAME FOR TIS U-DATE
(5) - 3 ASCII WORDS OF BLANKS
(6) - XSCAN VARIABLE (INT OR ZERO)
(7) - XUNITS (INT OR ZERO)
(8) - XCENTROID (REAL)
(9) - XINCREMENT (REAL)
(10) - NO OF ENTRIES IN SUMMARY TABLE
(11) - X NUMBER OF STEPS (INTEGER 1-5)
(12) - YSCAN VARIABLE (INT OR ZERO)
(13) - YUNITS (INT OR ZERO)
(14) - YCENTROID (REAL)
(15) - YINCREMENT (REAL)
(16) - Y SECOND SUBSCRIPT (INT OR ZERO)
(17) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPENDENT VARIABLE NAMES AND UNITS IN RECORDS 3, 4, & 5

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
DEPANDBY VARIABLE SCANNED (UP TO 32 VALUES
INCLUDING ERROR FLAG WHICH IS FIRST VALUE
IN SUMMARY TABLE)

OUTPUT FROM XZDP1

MASK TABLES CONTAINING THE CONSTRAINT MANS FOR EACH
CONSTRAINT WHICH WAS VIOLATED AND INDICATION OF
WHETHER OR NOT ANY CONSTRAINT WAS VIOLATED FOR EACH
ARRAY COORDINATE

LIST OF VALID DISPLAY DEPENDANT VARIABLES FOR EACH
PAGE (UP TO 16 PAGES)

LIST OF VALID CONSTRAINT VARIABLES WHICH WERE VIOLATED
(UP TO 8 CONSTRAINTS)

DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES

NAME OF DATA BOX TO BE DISPLAYED BY DBDSP

COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS

POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR

POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.

LIST OF VIOLATED CONSTRAINTS BUILT BY XZM8K (MAX OF 8)

FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)

SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)

NAME OF COMMON AREA USCD FOR INTERFACE TABLE

FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)

SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)

ARRAY CONTAINING CONSTRAINTS A THRU D

ARRAY CONTAINING CONSTRAINTS E THRU G

NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN

NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)

LIST OF CONSTRAINT RELATIONS INPUT BY USER

LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)

NUMBER OF DEP DISP VARIABLE PAIRS FOR PAGED OUT/P (1-16PR)

NUMBER OF DEP DISP VAR IN NDVARL LIST (INTEGER)

NDVRL - LIST OF DEP DISP VARIABLE PAIR CALF FACTORS FOR O/P

SET OF INDICATORS FOR CONSTRAINTS VIOLATED/0/NOT 0

NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)

NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)

X VALUES FOR X COORDINATES (1 - 11 REAL)

NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)

NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)

LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)

NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)

NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)

TABLE IN COMMON FOR SUMTAB VARIABLE NAMES AND UNITS

UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN

VALUES FOR SCAN VARIABLES(S) - 1 TO 32 VALUES/RECORD

COMMUNICATION BUFFER FOR RPAN - LU, USER ID, FLAG

LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATN

DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU

TABLE IN COMMON TO COMMUNICATE XPRM

SELECT = 0 PROMPT ; SELECT NOT O RUN ALL DISPLAYS TO O/P
**CD**********

**1306** **CD0** XIDMK - CONSTRAINT MASKING ROUTINE

**1307** **CD0** XIDMK IS CALLED ONCE BY XIDP1 TO BUILD THE CONSTRAINT

**1308** **CD0** VIOLATION MASKS FOR ALL VALUE POSITIONS OF THE DISPLAY GRID.

**1309** **CD0**

**1310** **CD**********

**1311** **CD1** XIDMK IS CALLED ONCE BY XIDP1 TO BUILD THE CONSTRAINT

**1312** **CD1** VIOLATION MASKS FOR ALL VALUE POSITIONS OF THE DISPLAY GRID.

**1313** **CD1**

**1314** **CD1**

**1315** **CD**********

**1316** **CD2** INPUTS

**1317** **CD2** COMMON - ATABLE, MCVARL, MCRELL, CVALUE

**1318** **CD2**

**1319** **CD2**

**1320** **CD3** OUTPUTS

**1321** **CD3** COMMON - MSKERR, MASK1, MASK2

**1322** **CD3** ISAVE

**1323** **CD3**

**1324** **CD3**

**1325** **CD3**

**1326** **CD**********

**1327** **CD4** NOTES

**1328** **CD4**

**1329** **CD4**

**1330** **CD4** USES Routines

**1331** **CD4**

**1332** **CD4** XRCFR

**1333** **CD4** XRM0V

**1334** **CD4** XRESET

**1335** **CD4**

**1336** **CD4**

**1337** **CD4**

**1338** **CD**********
**XZDP2**

1. **INPUTS TO XZDP2 FROM XZDP1**
   - Mask tables containing the constraint masks for each constraint which was violated and indication of whether or not any constraint was violated for each array coordinate.

2. **LIST OF VALID DISPLAY DEPENDANT VARIABLES FOR EACH PAGE (UP TO 16 PAGES)**
   - Data box display array values for all variables.

---

**OUTPUT FROM XZDP2**

- Display format shown in documentation is sent to name lu device.
1461 1 CD 4 MVARL - LIST OF DEP LISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
1462 1 CD 4 MVARCT - NUMBER OF DEP Disp VAR IN MVARL LIST (INTEGER)
1463 1 CD 4 MVARUL - LIST OF DEP Disp VARIABLE PAIR SCALE FACTORS FOR O/P
1464 1 CD 4 MVARREL - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT 0
1465 1 CD 4 MSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1466 1 CD 4 MSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1467 1 CD 4 REC - RTN CODE FM XPRMEX; 0=NRML, 1=RTN TO EXEC, 2=NULL BFR, 3=ERR
1468 1 CD 4 STRG - CHAR STRING CONTAINING USER PROMPT MESSAGE
1469 1 CD 4 TCORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
1470 1 CD 4 XCMNN - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1471 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1472 1 CD 4 TCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)
1473 1 CD 4 YCMNN - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1474 1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1475 1 CD 4 ZTABLE - TABLE IN COMMON FOR SUNTAB VARIABLE NAMES AND UNITS
1476 1 CD 4 VARUL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCM
1477 1 CD 4 SUNTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
1478 1 CD 4 PARM - COMMUNICATION BUFFER FOR RPR, RPR, USER ID, FLAGS
1479 1 CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATM
1480 1 CD 4 LUDBP - DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
1481 1 CD 4 FRMT - TABLE IN COMMON TO COMMUNICATE WITH XPRM
1482 1 CD 4 SELECT - SELECT =O PROMPT; SELECT NOT O RUN ALL DISPLAYS TO O/P
1483 1 CD 4 NO PROMPT
1484 1 CD 4 CARTC - CARTAGE USER TO LOCATE DATA BOX
1485 1 CD ********
1486 1 CD 5
1487 1 CD 5
1488 1 CD 5
1489 1 CD 5
1490 1 CD 5
1491 1 CD 5
1492 1 CD ********

USERS ROUTINES
XPRM, EXEC, XIDOT.
1591 1 CD******
1592 1 CD0
1593 1 CD0 FORTRAN CALLING PROCEDURE
1594 1 CD0 CALL XZISP (STRING, LEN)
1595 1 CD0
1596 1 CD******
1597 1 CD1 XZISP REMOVES DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1598 1 CD1 A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1599 1 CD1 WITH BLANKS
1600 1 CD1
1601 1 CD1
1602 1 CD******
1603 1 CD2 INPUT
1604 1 CD2
1605 1 CD2 CALLING SEQUENCE
1606 1 CD2
1607 1 CD2 STRING - INPUT CHARACTER STRING
1608 1 CD2 LEN - NUMBER OF WORDS IN STRING
1609 1 CD2
1610 1 CD2
1611 1 CD******
1612 1 CD3 OUTPUT
1613 1 CD3
1614 1 CD3 CALLING SEQUENCE
1615 1 CD3
1616 1 CD3 STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
1617 1 CD3 BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILL
1618 1 CD3 LEN - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
1619 1 CD3
1620 1 CD3
1621 1 CD******
1622 1 CD4 NOTES
1623 1 CD4
1624 1 CD4 USES Routines
1625 1 CD4
1626 1 CD4 XR1SP
1627 1 CD4
1628 1 CD4
1629 1 CD******
1630 1 BEGIN XZISP
1631 2 CALL XR1SP TO REMOVE DUPLICATE BLANKS FROM STRING
1632 2 DO WHILE THERE ARE TRAILING WORDS IN STRING
1633 2 SET THIS TRAILING WORD TO BLANKS
1634 2 ENDDO
1635 1 END XZISP
FORTRAN CALLING SEQUENCE:

CALL XIFCL (LU)

XIFCL SEARCHES XVSTB LOOKING FOR LU SO THAT THE
PROCESSOR'S CLASS NUMBER CAN BE EXTRACTED. IT
SAVES THE CLASS NUMBER IN EXTERNAL XPCLS.

INPUTS FROM CALLING SEQUENCE:

LU - LOGICAL UNIT OF TERMINAL BEING USED.

NOTE: CALLING PROGRAM MUST HAVE XPCLS DEFINED
AS AN EXTERNAL REFERENCE.

BEGIN XIFCL

CALL ETRN TO RESOLVE PARAMETER ADDRESS
START SEARCH FOR ALL XVSTB ENTRIES
EXIT IF THIS ENTRY'S LU IS MINE
ENDSEARCH
GET CLASS NUMBER FROM XVSTB
SAVE CLASS NUMBER IN XPCLS
END XIFCL
FORTRAN CALLING PROCEDURE:
CALL XIFNC(ENTRY)

XIFNC IS USED BY THE ASIGN ROUTINE XZPS2 TO EVALUATE FUNCTION
OPERATIONS

INPUTS
ENTRY - FUNCTION TOKEN CURRENTLY BEING PROCESSED

FROM ASGCOM - LU, STRNG, OPRNDS, OPINFO

OUTPUTS TO ASGCOM
RESULT, RSLTP, OPRNDS

EXTERNAL REFERENCES
FDS - XPXIT, XRMV, XILSS, XIMSG

RTE - ABS, AZNT, ALOG, ALGOT, AMOD, ATAN, ATAN2, COS, DABS, DATAN,
DATN2, DBLE, DCS, DDEPT, DLOG, DLCOT, DMOD, DEI-W, DSI, DSQRT,
EIP, FLEX, IABS, IDINT, IFIX, ISIGN, MOD, OVF, SIGN, SIM, SINGLE, SGRT,
TAN, TANH
1755 1 BEGIN XZFE
1756 2 * CASE (RESULT DATA TYPE ) :FRESTR:, :FREFE:, :FREFE:, :PREX:, :PREX:
1757 3 * :FRESTR:, :FREFE:, :FREFE:, :PREX:, :PREX:
1758 3 * :FRESTR:, :FREFE:, :FREFE:, :PREX:, :PREX:
1759 3 * :FRESTR:, :FREFE:, :FREFE:, :PREX:, :PREX:
1760 3 :FRESTR:
1761 3 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1762 3 MOVE CHARACTER STRING TO RESULT LOCATION
1763 3 :FREFE:
1764 3 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1765 4 POP DISPLACEMENT FROM RESULT STACK
1766 4 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIBED
1767 4 SET DISPLACEMENT = 0
1768 3 ENDIF
1769 3 POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1770 3 GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
1771 4 IF RESULT OPERAND IS FREE, THEN
1772 4 CALL XPREQ TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1773 4 SET # WORDS TO BE STORED IN OBJECT = 1
1774 4 ELSE FREE = CHARACTER DATA ELEMENT
1775 4 CALL XPREQ TO RETRIEVE LOGICAL ELEMENT OF CHARACTER DATA FROM RESULT OPERAND
1776 4 AT DISPLACEMENT DETERMINED
1777 4 SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR RESULT OPERAND
1778 3 ENDIF
1779 3 :PREX:
1780 3 CALL XZPCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781 3 SET # WORDS TO BE STORED = RESULT DATA TYPE
1782 2 END CASE
1783 1 END XZFE
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1785</td>
<td>1 CDO</td>
<td>FORTRAN CALLING PROCEDURE:</td>
</tr>
<tr>
<td>1786</td>
<td>1 CDO</td>
<td>CALL XIFXD</td>
</tr>
<tr>
<td>1787</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1788</td>
<td>1 CDO</td>
<td>FOR FIXED-TYPE OBJECT DATA ELEMENTS</td>
</tr>
<tr>
<td>1789</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1790</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1791</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1792</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1793</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1794</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1795</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1796</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1797</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1798</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1799</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1800</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1801</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1802</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1803</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1804</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1805</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1806</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1807</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1808</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1809</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1810</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1811</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1812</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1813</td>
<td>1 CDO</td>
<td>XIFXD IS USED BY THE ASG7N ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
</tbody>
</table>
1815 1 BEGIN XIFXD
1816 2  SET $ WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1818 4  2 3  :FIXFRE:
1819 5  3  ERREXIT TO :NOCHAR: IF TYPE IN SYMBOL TABLE FOR RESULT OPERAND IS NOT FREE
1820 6  3  IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1821 7  4  POP DISPLACEMENT FROM RESULT STACK
1822 8  3  ELSE FREE ELEMENT HAS NOT BEEN SUBSCRIPTED
1823 9  4  SET DISPLACEMENT = 0
1824 10  ENDIF
1825 11  3  POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1826 12  3  CALL XPREQ TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1827 13  3  (# WORDS RETRIEVED = OBJECT DATA TYPE)
1828 14  3  :FIXFRE:
1829 15  3  SET TARGET TYPE TO OBJECT DATA TYPE
1830 16  3  CALL XPCOS TO POP RESULT VALUE, CONVERT IF NECESSARY, AND SET UP FOR STORE
1831 17  END CASE
1832 18  2  EXIT XIFXD
1833 19  2  :NOCHAR:
1834 20  2  SET MESSAGE TO BE OUTPUT TO "NUMERICAL DATA ELEMENT CANNOT BE SET EQUAL TO
1835 21  2  CHARACTER DATA"
1836 22  2  CALL XIMSG TO OUTPUT MESSAGE TO USER
1837 23  2  CALL XLIST TO LIST SYMBOLIC STRING
1838 24  2  CALL XPXIT TO EXIT PROCESSER
1839 25  1  END XIFXD
INTERNAL VARIABLES

CONTROL = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING
FOR EACH OF THE TOKEN VALUES 1-32. EACH CONTROL TABLE
ENTRY IS 3 WORDS:

WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER

WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER

OR FLAG DESCRIBING HOW TO COMPUTE

THEN

WORD 3 (TOKSZ) = NO. OF WORDS IN SYMBOLIC STRINGS

CD4 ENTRY TOKEN WORD 1 WORD 2 WORD 3
CD4 NO. (SIZE) (FIELD) (TOKSZ)
CD4 1 INTEGER 3 1 CALL XRI6 2
CD4 2 REAL 7 2 CALL XRI14 3
CD4 3 DOUBLE 9 3 CALL XRI13 4
CD4 4 ---- 0 0 ERROR (INVALID) 0
CD4 5 NAME 3 4 USE 3 WORDS
CD4 6 ---- 0 0 ERROR (INVALID) 0
CD4 7 ---- 0 0 ERROR (INVALID) 0
CD4 8 CHAR. STR.-1 USE VALUE -5 USE SIZE WORDS -1 USE SIZE+2 WORDS
CD4 9 ---- 0 0 ERROR (INVALID) 0
CD4 10 * 1 * 1
CD4 11 / 1 / 1
CD4 12 < 1 < 1
CD4 13 > 1 > 1
CD4 14 ? 1 ? 1
CD4 15 = 1 = 1
CD4 16 & 1 & 1
CD4 17 | 1 | 1
CD4 18 & 1 & 1
CD4 19 ? 1 ? 1
CD4 20 ( 1 ) 1
CD4 21 ) 1 1
CD4 22 ) 1 1
CD4 23 ---- 0 0 ERROR (INVALID) 0
CD4 24 END SYM. STR. 0 0 END SYM. STR.
CD4 25 X 1 BACKSLASH 1
CD4 26 BACKSLASH 1 BACKSLASH 1
CD4 27 $ 1 1
CD4 28 # 1 # 1
CD4 29 LEFT BRACKET 1 LEFT BRACKET 1
CD4 30 RIGHT BRACKET 1 RIGHT BRACKET 1
CD4 31 REPEAT 4 6 CALL XRI6 AND 2
CD4 32 , 1 APPEND " " 1
1 C####
2 NOTES
3 C####
4 USES ROUTINES
5 EXEC XRI
6 EXEC XRE14
7 EXEC XRD18
8 EXEC XHSSG
9 EXEC XHENOV

10 1 BEGIN XILSS
11 2 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
12 2 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
13 2 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
14 2 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
15 2 EXIT TO ERROR 2 IF FIELD = 0
16 3 IF SIZE < 0, THEN
17 3 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
18 3 ENDIF
19 3 IF TOKSIZ < 0, THEN
20 3 SET TOKSIZ TO SIZE + 2
21 3 ENDIF
22 3 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
23 4 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
24 4 SET BUFFER POINTER TO 1ST POSITION FOR DATA
25 4 IF INDICATED TOKEN PROCESSED, THEN
26 5 OUTPUT LINE WITH INDICATOR
27 5 ENDIF
28 3 IF FIELD > 0, THEN
29 4 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
30 3 ELSE
31 4 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
32 5 SOME: CALL XRI WITH VALUE IN NEXT WORD OF SYMB. STRING
33 5 AND PUT RESULTS INTO PRINT BUFFER
34 5 :TWO: CALL XRE14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
35 5 AND PUT RESULTS INTO PRINT BUFFER
36 5 :THREE: CALL XRD18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
37 5 AND PUT RESULTS INTO PRINT BUFFER
38 5 :FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
39 5 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
40 5 :SIX: CALL XRI WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
41 5 INTO PRINT BUFFER FOLLOWED BY AN "R"
42 5 EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
43 5 INDEX BY 1
44 5 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
45 5 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
46 6 INDICATOR LINE NOT YET OUTPUT, THEN
FORTRAN CALLING PROCEDURE

CALL XIMSG (CONT, NUMBER, LOCATE, LENGTH, SOURCE)

**INPUT**
- THREE CONTROL WORDS CONTAINING THE TERMINAL 'LU' AND THE DEBUG CONTROL FLAG. IF CONT(3) BIT 12 IS ON, XUDBG WILL BE CALLED AFTER THE MESSAGE IS OUTPUT

**CONT**
- INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
  1 - AS
  2 - XB
  3 - XE
  4 - XI
  5 - XS
  6 - XT
  7 - XX
  8 - XL
  9 - DF
  10 - SC

**NN**
- MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'

**LOCATE**
- INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED

**SOURCE**
- ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE. ZERO INDICATES NO INSERTION

**LENGTH**
- INTEGER NUMBER OF WORDS OF MESSAGE TO INSERT INTO MESSAGE (NOT USED IF 'LENGTH' IS ZERO)

**OUTPUT**
- UP TO EIGHTY CHARACTER LINE OF TEXT TO UNIT 'LU' OF THE FORM 'ANNM' WHERE MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE

**NOTES**
- USES FOR SYSTEM MESSAGE FILE XIMSG
- USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XRMOV, XUDBG
BEGIN XZMSG
SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
SET NUMBER IN PREFIX
READ MESSAGE DIRECTORY RECORD
IF AREA VALID THEN
SET AREA CODE IN PREFIX
IF MESSAGE NUMBER > 0 THEN
IF VALID MESSAGE NUMBER THEN
COMPUTE MESSAGE RECORD NUMBER
READ RECORD
CALL XRMOV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
ELSE
EXIT TO ERROR:
ENDIF
ENDIF
CALL XRMOV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
IF MESSAGE NUMBER > 0 THEN
CALL XRMOV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
ELSE
SET AREA IN PREFIX
ERROR: CALL XRMOV TO MOVE 'XZMSG ERROR' INTO BUFFER
CALL XRMOV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
ENDIF
OUTPUT BUFFER TO USER'S TERMINAL
THEN
CALL XUDIS
ENDIF
END XZMSG
2071 1 CD0  FORTRAN CALLING PROCEDURE:
2072 1 CD0  CALL XZOPR(ENTRY)
2073 1 CD0
2074 1 CD0
2075 1 CD0
2076 1 CD0  XZOPR IS USED BY THE ASGON ROUTINE XIPS2 TO EVALUATE MATH OPERATIONS
2077 1 CD0
2078 1 CD0
2079 1 CD0
2080 1 CD0
2081 1 CD0  INPUTS
2082 1 CD0
2083 1 CD0  ENTRY - OPERATOR TOKEN CURRENTLY BEING Processed
2084 1 CD0
2085 1 CD0  FROM ASGON - LU,SSTRNG,SYNTAB,DATYPES,RSLTPC,CLSTM,MAPRDS,RESULT
2086 1 CD0
2087 1 CD0
2088 1 CD0  OUTPUTS TO ASGON
2089 1 CD0
2090 1 CD0  RESULT,OPRNDSD,REGST,RSLTPC
2091 1 CD0
2092 1 CD0
2093 1 CD0
2094 1 CD0  INTERNAL VARIABLES
2095 1 CD0
2096 1 CD0
2097 1 CD0  MAPOP - MAPS OPERATOR TOKENS FOR EXECUTION
2098 1 CD0
2099 1 CD0
2100 1 CD0
2101 1 CD0  EXTERNAL REFERENCES
2102 1 CD0
2103 1 CD0
2104 1 CD0
2105 1 CD0  FDS - XPREX,KPITL,KRMOV,XILSS,X2MSG
2106 1 CD0
2107 1 CD0  RTE - IANC,OWF
BEGIN XIOPR

CASE OPERATOR :
  ADD: PERFORM ADDITION
  SUBTR: PERFORM SUBTRACTION
  MUL: PERFORM MULTIPLICATION
  DIV: PERFORM DIVISION
  EXP: PERFORM EXPONENTIATION
  I: CHANGE SIGN OF OPERAND FOR RESULT
ENDCASE

EXIT XIOPR

OVER:
SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
FORTRAN CALLING PROCEDURE:

CALL XPCS(TARGET, OPMON)

XPCS IS USED BY ASSIGN TO POP AN OPERAND FROM THE RESULT STACK, CONVERT IT TO A TARGET TYPE, AND STORE IT FOR USE IN A MATH OR FUNCTION OPERATION

INPUTS

TARGET - DESIRED FDS FIXED DATA TYPE
OPMUN - OPERAND NUMBER FOR ENTRY CURRENTLY BEING SET UP
FROM ASGCON - LU, SSTRNG, RESULT, RSLTP, DATYPES

OUTPUTS TO ASGCON
RSLTP, OPMUNS

EXTERNAL REFERENCES
FDS - XPRIT, XMOV, XZLS, XMSG
RTE - COLS, FLOST, IFFIX, OVF, SGML
FUNCTION CALLING PROCEDURE:
CALL XIPS1

CD0

XIPS1 IS USED TO TRANSLATE THE SYMBOLIC STRING EXPRESSION
TO A POST-FIX NOTATION STRING

CD1

INPUTS FROM ASGCOM
LU, TOKENS, STRING, EXPTK, SYNTAX, FNCTBL

CD2

OUTPUTS TO ASGCOM
EXPTK, RNGSTK, POLISH, SYNTAX

CD3

INTERNAL VARIABLES
GRPSIK - 2X120 ARRAY USED TO TRACK FUNCTIONS, SUBSCRIPTS, AND
PARENTHETICAL GROUPINGS
OPSTK - 2X120 ARRAY; EACH ENTRY CONTAINS OPERATOR TOKEN AND ITS OUTPUT
PRIOR
PRCONT - PRECEDENT TYPE FOR PRECEDING CHARACTER; USED FOR SYNTAX CHECK
TKNPTK - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING
TOKEN - TOKEN CURRENTLY BEING PROCESSED

CD4

EXTERNAL REFERENCES
RTE - IAMD, MIND
FDS - XPSI, XLSK, XIMSG, XSYM
BEGIN SETUP

CASE TOKEN

:SYMBOL: :INVL$: :SYMBOL: :INVL$: 

NAME CHARACTER + 
/


= () 

:ENDCASE: :INVL$: :ENDCASE: :ENDCASE: 

LEFT BRKT BY BRKT 

:ENDCASE: :ENDCASE: 

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF
2400 1 BEGIN RANGE
2401 2 INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCRCMENT = 1
2402 3 IF LAST TOKEN INPUT WAS ; THEN
2403 4 DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
2404 5 ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
2405 6 CALL XLSYM TO INSERT NAME IN SYMOL TABLE, IF NECESSARY, FLAG ENTRY AS
2406 7 AN INDEX, AND RETURN SYMBOL TABLE INDEX
2407 8 ERREXIT IF NEXT TOKENS ARE NOT "-INTEGER,INTEGER" TO :RNGSYM:
2408 9 IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
2409 10 SET INCREMENT TC -1
2410 11 ENDIF
2411 12 PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
2412 13 END DO
2413 14 ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
2414 15 ENDIF
2415 16 EXIT RANGE
2416 :RNGSYM:
2417 2 SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ";NAME=
2418 3 * INTEGER,INTEGER"
2419 4 CALL XLSMC TO OUTPUT MESSAGE TO USER
2420 5 CALL XLSLS TO LIST SYMBOLIC STRING
2421 6 CALL XPSMT TO EXIT PROCESSOR
2422 1 END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

XIPS2 IS USED BY ASSGN TO EVALUATE THE POST-FIX STRING GENERATED
BY XIPS1 AND TO STORE THE RESULTS INTO THE OBJECT DATA ELEMENT

INPUTS FROM ASGCOM

LU, TOKENS, LASTY, SYMTAB, SSTRANG, RESULT, OPARNS

OPINFO, NUMWDS, DATIPS, RSLTPT, EXPR, CLSSEQ, CLSTRM, DECLS,

RNGSTR, POLISH, SYNTAX, FMCTBL

OUTPUTS TO ASGCOM

SYMTAB, RESULT, RERST, RSLTPT, OPINFO

INTERNAL VARIABLES

ENTRY - ENTRY FROM POST-FIX STRING WHICH IS CURRENTLY BEING PROCESSED

EXTERNAL REFERENCES

FDS - XPRER, XPXIT, XROMV, XCHNR, XIFMC, XIFRE, XIFXD, XILL5, XIMSG,

XIOPR, XIPCS, XIRET

RTE - IABS, IANO, MAXB
2465 1 BEGIN XIPS2
2466 2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
2467 2 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
2468 2 INITIALIZE RANGE VALUES FOR ITERATION
2469 3 DO UNTIL ALL RANGES ARE FINISHED
2470 3 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
2471 4 POP ENTRY FROM POLISH STRING
2472 4 IF ENTRY IS AN OPERAND, THEN
2473 5 PERFORM RUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH ONTO RESULT STACK
2474 4 ELSE ENTRY IS AN OPERATOR
2475 5 IF OPERATOR IS NOT =, THEN
2476 6 PERFORM LEVEL TO EVALUATE POLISH STRING
2477 5 ELSE OPERATOR IS =
2478 6 PERFORM REPLACE TO STORE VALUE INTO OBJECT DATA ELEMENT
2479 5 ENDIF
2480 4 END DO
2481 3 END DO
2482 3 PERFORM RNGSET TO DETERMINE CORRECT RANGE VALUES
2483 2 END DO
2484 1 END XIPS2
BEGIN TOC
DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
IF SYMBOL NOT = 0 (SYMBOL IS DE OR INDEX), THEN
IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN
CALL XPREF TO RETRIEVE TOC ENTRY
ELSE SYMBOL IS A RANGE INDEX
SET DATA TYPE TO INTEGER
ENDIF
ENDIF
END DO
CALL XPREF WITH A CLOSE BUFFER REQUEST
END TOC
BEGIN DATA
2500 DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
2501 IF SYMBOL IS A NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENT, THEN
2502 CALL XPREP TO QUEUE REQUEST FOR DATA RETRIEVAL
2503 ENDIF
2504 END DO
2505 CALL XPREP WITH A CLOSE BUFFER REQUEST
2506 END DATA
2508 1 BEGIN RPUSH
2509 2 IF OPERAND IS A CHARACTER STRING POINTER (OPERAND < 0), THEN
2510 3 PUSH ABSOLUTE VALUE OF OPERAND AND -3 DATA TYPE ONTO RESULT STACK
2511 4 ELSE OPERAND IS A SYMBOL TABLE INDEX
2512 5 CALL SIZE TO RETRIEVE DATA AND DATA TYPE FOR OPERAND
2513 6 PUSH RETRIEVED VALUE AND DATA TYPE ONTO RESULT STACK
2514 7 ENDRIP
2515 8 END RPUSH
1 BEGIN EVAL
2   IF OPERATOR IS A FUNCTION, THEN
3     DETERMINE & OPERANDS FROM FUNCTION TABLE
4     ELSE
5       DETERMINE & OPERANDS FROM SYNTAX TABLE
6     ENDIF
7     DO FOR EACH OPERAND
8       IF RESULT STACK CONTAINS A SYMBOL INDEX, THEN
9         ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
10        CALL XPED TO RETRIEVE DATA AND TYPE FOR FIRST ELEMENT OF ARRAY
11        STORE DATA AND TYPE IN RESULT STACK AT THIS ENTRY
12       ELSE
13         ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
14     ENDIF
15     END DO
16     IF THE OPERAND IS A FUNCTION, THEN
17       DO FOR EACH FUNCTION OPERAND
18         SET TARGET TYPE FROM FUNCTION TABLE
19        CALL XPSCS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
20     END DO
21     CALL XPFC TO PERFORM FUNCTION FOR RESULTS
22     ELSE
23       THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "="
24       DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
25     ENDIF
261 4  CASE (TYPE REQUIREMENT+1) :SAME:, :INTET:
278 4  :SAME:
283 5  IF # OPERANDS > 1, THEN
294 5  SET TARGET TYPE TO MAX OF TWO DATA TYPES
305 4  ELSE # OPERANDS = 1
316 5  SET TARGET TYPE TO OPERAND'S DATA TYPE
327 4  ENDF
328 4  :INTET:
329 4  SET TARGET TYPE TO INTEGER
330 3  ENDCASE
331 3  DO FOR EACH OPERAND
332 4  CALL XPSCS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
333 3  END DO
334 3  CALL XPORR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
335 2  ENDF
336 1 EXIT EVAL
337 2
338 2
339 2
340 2
341 2
342 2
343 2
344 2
345 2
346 2
347 2
348 2
349 2
350 2
351 2
352 2
353 2
354 2
355 2
356 1
357 2
358 2
359 2
360 2
361 2
362 2
363 1
2565 1 BEGIN REPLAC
2566 2 IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
2567 3 IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
2568 4 CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
2569 4 STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
2570 4 ENDIF
2571 2 * ENDF
2572 2 * CASE (OBJECT DATA TYPE ) :FREE; :FIXED; :FIXED; :CHAR; :CHAR; 
2573 7 :CHAR; :CHAR; :CHAR;
2574 5 * :FREE:
2575 3 CALL XPFREE TO RETRIEVE DATA AND SET UP FOR STORE
2576 3 :FIXED:
2577 3 CALL FIXOBJ TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
2578 3 :CHAR:
2579 3 CALL CHROBJ TO RETRIEVE DATA AND SET UP FOR STORE
2580 3 END CASE
2581 2 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT, THEN
2582 2 POP OBJECT'S DISPLACEMENT FROM RESULT STACK
2583 2 ELSE OBJECT HAS NOT BEEN SUBSCRIPTED
2584 2 SET OBJECT'S DISPLACEMENT TO 0
2585 2 ENDF
2586 2 BUILD XPREQ REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S
2587 2 DISPLACEMENT
2588 2 CALL XPREQ TO STORE DATA IN OBJECT
2589 1 END REPLAC
BEGIN RNGSET
DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED RANGES ARE PROCESSED
IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN INCREMENT RANGE VALUE
ELSE SET RANGE INDEX VALUE TO BEGIN VALUE
ENDIF
END BO
END RNGSET
FORTRAN CALLING PROCEDURE:
CALL XIRET(ENTRY)

XIRET IS USED BY THE ASSGN ROUTINE XIIPS2 TO RETRIEVE THE DATA VALUE
FOR A GIVEN SYMBOL INDEX FROM THE SYMBOL TABLE AND PUSH IT ONTO THE
RESULT STACK ALONG WITH ITS DATA TYPE

INPUTS
ENTRY - SYMBOL TABLE INDEX CURRENTLY BEING PROCESSED
FROM ASSCOM - SYMTAB,RSLTPT

OUTPUTS TO ASSCOM
RESULT,RSLTPT

EXTERNAL REFERENCES
FDS - XRMV
RTE - IAMO


FORTRAN CALLING PROCEDURE

CALL XSYM (TKNPR, FLAG, SYMIND)

**EXTERNAL REFERENCES**

XPIIT, XILSS, XIMSG, XISYM

**PARAMETERS**

1) KMNPR - INDEX INTO INPUT SYMBOLIC STRING (SSTRING) OF TOKEN TO BE ENTERED OR LOCATED IN SYMBOL TABLE (SYMTAB)
2) FLAG - SPECIAL PROCESSING FLAG
   0, NORMAL PROCESSING
   1, SYMBOL IS A RANGE
   2, SYMBOL IS SUBSCRIPTED
3) SYMIND - INDEX INTO SYMBOL TABLE (SYMTAB) WHERE SYMBOL IS LOCATED.
4) VALUE OF SYMIND INDICATES ENTRY NUMBER, I.E., 1, 2, ...
5) ASGCOM COMMON - LU, SSTRING, STWIDE, SYMTAB, TOKENS.
BEGIN XSYM

CALL XSYM TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY

IF TOKEN IS A NAME, THEN

IF SPECIAL PROCESSING FLAG IS SET (1=RANGE INDEX, 2=SUBSCRIPTED), THEN

IF FLAG INDICATES RANGE INDEX, THEN

EXIT TO :BADRNG: WITH ERROR AS01 IF OBJECT (FIRST ENTRY IN TABLE)

EXIT TO :BADRNG: WITH ERROR AS02 ENTRY IS ALREADY SUBSCRIPTED

EXIT TO :BADRNG: WITH ERROR AS03 ENTRY IS ALREADY A DEFINED RANGE

ENDIF

SET ENTRY FLAG WORD TO FLAG VALUE

ENDIF

EXIT XSYM

:BADRNG:

CALL XMSG TO OUTPUT ERROR DESCRIPTION

CALL XLOSE TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR

CALL XEXIT TO TERMINATE PROCESSOR

END XSYM
FORTRAN CALLING PROCEDURE

CALL XISYT (TKMPtr, SYMID)

Provided key values are suitably located in common, XISYT provides
a general capability for symbol table access for tokens input via
a symbolic string.

INPUT
TKMPtr - index into input symbol string (SSTRNG) of token to be
entered or located in symbol table (SYMtab).

ASCOM COMMON - LASTST, SSTRNG, SYMID, SYMtab, tokens

OUTPUT
SYMID - index into symbol table (SYMtab) where symbol is located.
VALUE OF SYMID INDICATES ENTRY NUMBER, I.E., 1, 2, ...

ASCOM COMMON - LASTST, SYMtab

INTERNAL
DISP - displacement into symbol table entry for field to be used
as key. Names use a value of 1, constants a value of 8.
EDLOOP - index of last allocated word in SYMtab. Value is equal
to LASTST*SYMID.
IBUF - internal buffer for symbol value used to assure last
words of integer and real constants are zero.

EXTERNAL REFERENCES
XRCPR, XRNOW

SPECIAL REMARKS
THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS

I = TOKEN CODE I (3 WDS) I RESERVED FOR TOC ENTRY I (3 WDS) I (IN WDS) I

I TYPE (4 WDS) I NAME I Unused (4 WDS) I VALUE I OTHER I

CD**********
BEGIN XISTT
CLEAR BUFFER TO BE USED IN MOVING TOKEN
IF TOKEN IS A NAME, THEN
  SET COMPARISON DISPLACEMENT IN TABLE TO 1 (NAME FIELD)
ELSE
  SET COMPARISON DISPLACEMENT IN TABLE TO 8 (VALUE FIELD)
ENDIF
MOVE TOKEN INTO BUFFER
START SEARCH UNTIL ALL ALLOCATED SYMBOL TABLE ENTRIES EXAMINED
EXIT IF ENTRY MATCHES BUFFER CONTENTS AND TYPE FIELD MATCHES TOKEN CODE
END LOOP
STORE TOKEN CODE IN TYPE FIELD OF NEXT ENTRY
STORE BUFFER CONTENTS INTO APPROPRIATE FIELD OF ENTRY (NAME OR VALUE)
INCREMENT NUMBER OF ALLOCATED ENTRIES
END SEARCH
SET STRING TO ENTRY NUMBER
END XISTT
DO - CONDITIONAL ITERATION (LOOPING) PROCESSOR

SCHEDULED BY FBS

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FBS USERS WITH THE CAPABILITY OF LOOPING THROUGH A BLOCK OF SEQUENCE TABLE ENTRIES EITHER UNTIL A GIVEN CONDITION IS SATISFIED OR WHILE THE CONDITION IS TRUE.

INPUT FROM MANAGER VIA SCHEDULING PARAMETERS

LU - LOGICAL UNIT OF USER'S TERMINAL
FLAGS - DEBUG FLAGS FROM USER SIGN-ON
SENDSP - INDEX OF CURRENTLY EXECUTING SEQUENCE ENTRY

INPUT FROM INTERFACE TABLE

DQTYPE - 4CH CHARACTER STRING SPECIFYING LOOP CONDITION TYPE
UNTL - LOOP UNTIL. RELATION BETWEEN OPRND1 AND OPRND2 IS TRUE. TEST IS MADE AT THE END OF EACH LOOP EXECUTION.
WHILE - LOOP WHILE RELATION BETWEEN OPRND1 AND OPRND2 IS TRUE. TEST IS MADE PRIOR TO STARTING EACH LOOP EXECUTION.

OPRND1 - FIRST REAL QUANTITY TO COMPARE (VALUE NOT RETRIEVED, CHARACTERISTICS PASSED TO ENDDO)
OPRND2 - SECOND REAL QUANTITY TO COMPARE (SEE OPRND1)

INPUT FROM AVA ON REQUEST TO MANAGER

INBUF - INTERFACE TABLE (LESS LITERAL AREA)
LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE LITERAL AREA
BSOSTK - SEE OUTPUT DEFINITION
BSEOTB - EXEC COPY OF CURRENTLY EXECUTING SEQUENCE TABLE

OUTPUT TO THE MANAGER VIA RETURN PARAMETERS

RETURN - RETURN CODE TO MANAGER
0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL EXECUTION
3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE NUMBER FOR CONTINUED EXECUTION
8 - ABNORMAL TERMINATION. ABORT SEQUENCE EXECUTION

RESET - $FAHENCE RESET NUMBER OF ENDDO IF RETURN = 3

INPUT/OUTPUT FROM TO AVA

BSOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27,\text{n}) \text{ where } n \text{ is the current number of active loops.}
Each entry has the following form:

Words 1-7 - first seven words of do interface table
with the following changes (see fds sd0,
volume iv, figure 1.2-17)

Words 8-14 - interface table entry for oprnd1 (same as
in original table except for possible new
values pointing into literal area)

Words 15-21 - interface table entry for oprnd2 (see
above)

Words 22-25 - literal area for value(s)/subscript(s) of
oprnd1 & 2

Word 26 - relation id code as follows

0 - <
1 - =
2 - > or =>
3 - =
4 - <= or ==
5 - <

Word 27 - sequence number of top of loop

Internal variables:

code - array of eight acceptable relation mnemonics and
corresponding internal codes
doent - index into gdstk for new 27 word entry
mruff - 64 word manager communications buffer. eight word
entries are of the form
renst - ama management request code
class - class and type of data
name - six character data name
size - size of data
disp - displacement into data of transaction origin
clash - class i/o number through which data is
transmitted
top - index into gdstk of top of loop
xpcls - class i/o number for manager communications (set by
xpget)

Referenced routines

exec, iand, mod, xpr, xpget, xpreg(xpget), xpset, xrcpr, xmov,
xset, xu8eg, xvpm, ximsg

Notes

Do and enddo must be used in pairs

Code is allocated in the ama for interprocessor communications

xesc cleans up any residual gdstk after execution controller

termination

The maximum number of nested loops is 4

All nesting (structure) errors are left for execution time

Detection
BEGIN DO
CALL XGET TO INITIALIZE ACCESS TO XPREP AND TO RETRIEVE DTYPE AND RELAT
CALL XYPAM TO RETRIEVE DOSTK INTO BUFFER
IF RETRIEVAL FAILED, THEN
INITIALIZE BUFFER FOR BUILDING FIRST DOSTK ENTRY
ENDIF
IF DOSTK IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
IF XPREP INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXIST, THEN
CALL XPREP TO RETRIEVE LITERALS
ENDIF
IF RELAT IS A VALID RELATION OPERATOR, THEN
SET RELATION CODE IN NEW ENTRY IN BUFFER
SET INTERFACE TABLE HEADER WITH NAME OF BINTAB AND NUMBER OF PARAMETER; OF 2
DO FOR EACH OPND
MOVE OPND ENTRY INTO NEW INTERFACE TABLE BUFFER
IF OPND HAS LITERAL VALUE OR DOUBLE SUBSCRIPTS, THEN
MOVE LITERAL DATA
ADJUST LITERAL POINTERS
ENDIF
ENDIF
CALL XPREP TO RETRIEVE XSERTB (EXECUTING SEQUENCE TABLE)
IF XSERTB DISPLACEMENT (XQDSP) > 0, THEN
SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (0 IF END OF TABLE)
ELSE INSERTED COMMAND
EXIT TO :ERROR2: IF SEQUENCE NUMBER IS ZERO (MANUAL)
LOCATE ORIGINAL SEQUENCE ENTRY
IF ORIGINAL ENTRY WAS ALSO A DO (OVERWRITE CONDITION), THEN
SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (OR ZERO)
ELSE (INSERT)
SET TOP OF LOOP TO CURRENT NUMBER
ENDIF
ENDIF
CASE (:WHILE:, :UNTIL:, :OTHER:) DTYPE
:WHILE:
INVERT RELATION CODE
INITIALIZE NEXT COUNTER TO 1
START SEARCH FROM TOP OF LOOP ENTRY UNTIL ALL ENTRIES HAVE BEEN EXAMINED
IF COMMAND IS ENDDD, THEN
DECREMENT NEXT COUNTER
ELSE
IF COMMAND IS ANOTHER DO, THEN
INCREMENT NEXT COUNTER
ENDIF
ENDIF
EXIT IF NEXT COUNTER IS ZERO
SET RESET NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
END LOOP
EXIT TO :ERROR4: FOR NO MATCHING ENDDO
END SEARCH
:UNTIL:
CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
:OTHER:
TERMINATE WITH ERROR7 FOR UNRECOGNIZED DTYPE
END CASE
CALL XPREP TO OUTPUT NEW EXPANDED DOSTK
ELSE INVALID RELATION
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
'i

k;

W W WN WN WN WN WN WN WN WN WH WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN WN
JJN JN JJ
J J JJ JJ

W
V
2
W
d
W
N
W
S

W
N
K
W

r
W
Z

K
K
K

0
W
i
M
O
W
S
V
H
N
O
W
O
K

O u
r
Z
W NO 10W
"—W
It 10
OZ•-t
rN

r
2
«r
K

WKN
N
OW

MO a
W 2
ZW Zz OW
r
Z^
W
Sr u
r NW xW

N Z

NWW

W OL
I-.. N

0

JUe

r
or
M

K
H
H

O WN W
NN
W W WZ

1
r
rl N

w

.Jaw

MH ZK Ow
WN WJN WOY
JWW OI
d
to

W ►rW

r

SSS
O
♦ IV • l^1^

o>c..'i
r«+Kru
ZW rM,OZx
<t W
N VK ► y W

Z 1

r

W K
CCW
KHW
WW N ►-W m
r6u
K2 O
z4m
Z N0 W>b c
11U,
49
jMOZ
"= z
<u
d0 •2
KO m W
K
2
=n AM
11
W
W 11
K M OD Vl

r

N

^i tW

s

rNI►

r21a01-

a m

Yx ►r

OOtr O
d N J
Y I-W. OJ

0

S
r^
r

K W M

NJtV
K •[

W V W
W W in
={ OJ WO
K
i11
W
Z W
H
M

r Fi 'in

d>>J
ZOJ1^

M MW
k S
4 V

W•

x
G.MW Wx fA
xz
zN
N
K
K
t-t►-2 ON td
>O to
V W K
W
OO 2
to r at W
oe LL,
CL S
Z.0 V W s
t Wz rT x
W
N
00
0
W WruO
N
N
O
O ta W

CD
W
O
W
KOV
JJ
Ms

i&

1
Z
K
d
W
O
IL
IL
It
K
Z
Z
R
K

M

CD

WOLW

oz

r ^r 11
rK un

d
W

OA. x W
VNx r
W
Kd •N
K ZM
OY v^ r

W
V
2
W
d
W
h
O M

1 11

O Z

11 11

N
W
Z
r

aK

W
u

2
W
K
W
W
W pl^b1^,,1 >R

k
K Kwx OtiV
}
<
^nV
Nk k
p
pk O 0:^
}M•
d 14 W O W N f
dv
•
N
cc
KYvK
x x nx NN
xx
*k
4
k
M
*
*
k
41
p
k
M
p
p
k
k
000
k
OYYVVYVYVVVVVVVt.•
O O O O O O O O O O ONNNNNN
O O p GV OV0VVV
O O OVVUV
O 4j o uVo VVUVVUV
0 0 0 O O o oVVVV
OOO V
tJ O O
w

O
rNMf N tO I^dOP Of NMdV1^01^OOPO^NMC to SOP k`P O^NMf V^O I^Iq
PPPPPPPPPPOOOOOOOU00.-.-^^.-.-.-^^^NNNNNNNNN
rr^r^rt^^rrNNNNNNNNNNNNNNN N NN N N N N"NfV"NNN

5-408


230 1 BEGIN ELSE
231 2  CALL RMPAR TO GET SCHEDULING PARAMETERS
232 2  CALL ZIFCL TO ESTABLISH FDS MANAGER'S CLASS NO. (XPCLS)
233 2  CALL ZISCN TO SEARCH FOR MATCHING ENOIF COMMAND
234 2  CALL XPIIT TO RETURN PARAMETERS TO FDS MANAGER
235 1  END ELSE

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
ENDDO -- LOOP TERMINATION PROCESSOR

SCHEDULED BY FDs

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FDs USERS WITH THE
CAPABILITY OF LOOPTING THROUGH A BLOCK OF SEQUENCE TABLE Entries
EITHER UNTIL A GIVEN CONDITION IS SATISFIED OR WHILE THE CONDITION
IS TRUE.

INPUT FROM MANAGER VIA SCHEDULING PARAMETERS
LD - LOGICAL UNIT OF USER'S TERMINAL
FLG - DEBUG FLAGS FROM USER SIGN-ON

INPUT/OUTPUT FROM/TO AWA
BDOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(C27,N) WHERE N IS THE CURRENT NUMBER OF ACTIVE LOOPS.
SEE DO PROCESSOR FOR CONTENTS

THE FDs SYSTEM STATUS TABLE, XSTB, IS ACCESSED FOR THE MANAGER/
PROCESSOR CLASS I/O COMMUNICATIONS NUMBER

OUTPUT TO THE MANAGER VIA RETURN PARAMETERS
RETURN - RETURN CODE TO MANAGER
0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL
EXECUTION
3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE
NUMBER FOR CONTINUED EXECUTION
8 - ABNORMAL TERMINATION. ABORT SEQUENCE EXECUTION
RESET - SEQUENCE RESET NUMBER OF TOP OF LOOP IF RETURN = 3

INTERNAL VARIABLES
DOENT - INDEX INTO BDOSTK FOR LAST 27 WORD ENTRY
MBUFF - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD
ENTRIES ARE OF THE FORM
RESQ - AWA MANAGEMENT REQUEST CODE
CLASS - CLASS AND TYPE OF DATA
NAME - SIX CHARACTER DATA NAME
SIZE - SIZE OF DATA
DISP - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN
CLASS - CLASS I/O NUMBER THROUGH WHICH DATA IS
TRANSMITTED
XPLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

REFERENCE ROUTINES
EXEC, INDX, XRPAF, XPGET, XPERR(XPGET), XPXIT, XRLOC, XRMOV,
XUD0G, XRPAF, XIMSG

ENDDO
298 1 BEGIN ENDDO
299 2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XVSTB
300 2 CALL XUPAN TO RETRIEVE BOSTK
301 2 IF RETRIEVAL WAS SUCCESSFUL, THEN
302 3 WRITE BINTAB TO CLASS I/O FROM LAST BOSTK ENTRY
303 3 WRITE NON-LITERAL PORTION OF BINTAB TO CLASS I/O (LEAVE FOR XPGET)
304 3 CALL XVPAN TO RESTORE BINTAB FROM FIRST CLASS BUFFER INTO AWA
305 3 IF RESTORE SUCCESSFUL, THEN
306 4 CALL XPGET TO RETRIEVE OPRMD
307 4 CALL XEEVL TO EVALUATE RELATION
308 4 IF RELATION IS TRUE, THEN
309 5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
310 5 IF THIS IS LAST BOSTK ENTRY, THEN
311 6 DELETE BOSTK FROM AWA
312 6 ELSE
313 6 CALL XPREQ TO REPLACE BOSTK LESS LAST ENTRY
314 7 ENDIF
315 4 ELSE
316 5 SET RESET NUMBER TO TOP OF LOOP
317 6 ENDIF
318 3 ELSE
319 4 TERMINATE WITH ERR06 FOR AWA OVERFLOW
320 3 ENDIF
321 2 ELSE
322 3 TERMINATE WITH ERR03 FOR ENDDD WITH OUT DO
323 2 ENDIF
324 1 END ENDDO
326 1 CD******
327 1 CD0
328 1 CD0 FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
329 1 CD0
330 1 CD0
331 1 CD0
332 1 CD0 THE ENDF Utility Processor returns to the FDS Manager (via XPIXT) so
333 1 CD0 that normal processor execution sequence may continue
334 1 CD0
335 1 CD0
336 1 CD0
337 1 CD0 INPUTS
338 1 CD0
339 1 CD0 SCHEDULING PARAMETERS
340 1 CD0 LU = LOGICAL UNIT NO. OF FDS USER
341 1 CD0 FLAGS = DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT
342 1 CD0
343 1 CD0
344 1 CD0 OUTPUTS
345 1 CD0
346 1 CD0
347 1 CD0 RMPAMS - RETURN PARAMETERS FOR FDS MANAGER VIA XPIXT
348 1 CD0 (1) = 0 => CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
349 1 CD0
350 1 CD0
351 1 CD0
352 1 CD0 ROUTINES USED
353 1 CD0
354 1 CD0 RMPAR
355 1 CD0 XREXT
356 1 CD0 XPIXT
357 1 CD0 XUDBG
358 1 CD0
359 1 CD0
1 CD********
2 FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

1 CD0
2 CD0
3 CD0
4 CD0

5 CD0
6 CD0
7 CD0
8 CD0

9 CD0
10 CD0
11 CD0
12 CD0

13 CD0
14 CD0
15 CD0
16 CD0

17 CD0
18 CD0
19 CD0
20 CD0

21 CD0
22 CD0
23 CD0
24 CD0

25 CD0
26 CD0
27 CD0
28 CD0

29 CD0
30 CD0
31 CD0
32 CD0

33 CD0
34 CD0
35 CD0
36 CD0

37 CD0
38 CD0
39 CD0
40 CD0

41 CD0
42 CD0
43 CD0
44 CD0

45 CD0
46 CD0
47 CD0
48 CD0

49 CD0
50 CD0
51 CD0
52 CD0

53 CD0
54 CD0
55 CD0
56 CD0

57 CD0
58 CD0
59 CD0
60 CD0

61 CD0
62 CD0
63 CD0
64 CD0

65 CD0
66 CD0
67 CD0
68 CD0

69 CD0
70 CD0
71 CD0
72 CD0

73 CD0
74 CD0
75 CD0
76 CD0

77 CD0
78 CD0
79 CD0
80 CD0

81 CD0
82 CD0
83 CD0
84 CD0

85 CD0
86 CD0
87 CD0
88 CD0

89 CD0
90 CD0
91 CD0
92 CD0

93 CD0
94 CD0
95 CD0
96 CD0

97 CD0
98 CD0
99 CD0
100 CD0

101 CD0
102 CD0
103 CD0
104 CD0

105 CD0
106 CD0
107 CD0
108 CD0

109 CD0
110 CD0
111 CD0
112 CD0

113 CD0
114 CD0
115 CD0
116 CD0

117 CD0
118 CD0
119 CD0
120 CD0

121 CD0
122 CD0
123 CD0
124 CD0

125 CD0
126 CD0
127 CD0
128 CD0

129 CD0
130 CD0
131 CD0
132 CD0

133 CD0
134 CD0
135 CD0
136 CD0

137 CD0
138 CD0
139 CD0
140 CD0

141 CD0
142 CD0
143 CD0
144 CD0

145 CD0
146 CD0
147 CD0
148 CD0

149 CD0
150 CD0
151 CD0
152 CD0

153 CD0
154 CD0
155 CD0
156 CD0

157 CD0
158 CD0
159 CD0
160 CD0

161 CD0
162 CD0
163 CD0
164 CD0

165 CD0
166 CD0
167 CD0
168 CD0

169 CD0
170 CD0
171 CD0
172 CD0

173 CD0
174 CD0
175 CD0
176 CD0

177 CD0
178 CD0
179 CD0
180 CD0

181 CD0
182 CD0
183 CD0
184 CD0

185 CD0
186 CD0
187 CD0
188 CD0

189 CD0
190 CD0
191 CD0
192 CD0

193 CD0
194 CD0
195 CD0
196 CD0

197 CD0
198 CD0
199 CD0
200 CD0

201 CD0
202 CD0
203 CD0
204 CD0

205 CD0
206 CD0
207 CD0
208 CD0

209 CD0
210 CD0
211 CD0
212 CD0

213 CD0
214 CD0
215 CD0
216 CD0

217 CD0
218 CD0
219 CD0
220 CD0

221 CD0
222 CD0
223 CD0
224 CD0

225 CD0
226 CD0
227 CD0
228 CD0

229 CD0
230 CD0
231 CD0
232 CD0

233 CD0
234 CD0
235 CD0
236 CD0

237 CD0
238 CD0
239 CD0
240 CD0

241 CD0
242 CD0
243 CD0
244 CD0

245 CD0
246 CD0
247 CD0
248 CD0

249 CD0
250 CD0
251 CD0
252 CD0

253 CD0
254 CD0
255 CD0
256 CD0

257 CD0
258 CD0
259 CD0
260 CD0

261 CD0
262 CD0
263 CD0
264 CD0

265 CD0
266 CD0
267 CD0
268 CD0

269 CD0
270 CD0
271 CD0
272 CD0

273 CD0
274 CD0
275 CD0
276 CD0

277 CD0
278 CD0
279 CD0
280 CD0

281 CD0
282 CD0
283 CD0
284 CD0

285 CD0
286 CD0
287 CD0
288 CD0

289 CD0
290 CD0
291 CD0
292 CD0

293 CD0
294 CD0
295 CD0
296 CD0

297 CD0
298 CD0
299 CD0
300 CD0

301 CD0
302 CD0
303 CD0
304 CD0

305 CD0
306 CD0
307 CD0
308 CD0

309 CD0
310 CD0
311 CD0
312 CD0

313 CD0
314 CD0
315 CD0
316 CD0

317 CD0
318 CD0
319 CD0
320 CD0

321 CD0
322 CD0
323 CD0
324 CD0

325 CD0
326 CD0
327 CD0
328 CD0

329 CD0
330 CD0
331 CD0
332 CD0

333 CD0
334 CD0
335 CD0
336 CD0

337 CD0
338 CD0
339 CD0
340 CD0

341 CD0
342 CD0
343 CD0
344 CD0

345 CD0
346 CD0
347 CD0
348 CD0

349 CD0
350 CD0
351 CD0
352 CD0

353 CD0
354 CD0
355 CD0
356 CD0

357 CD0
358 CD0
359 CD0
360 CD0

361 CD0
362 CD0
363 CD0
364 CD0

365 CD0
366 CD0
367 CD0
368 CD0

369 CD0
370 CD0
371 CD0
372 CD0

373 CD0
374 CD0
375 CD0
376 CD0

377 CD0
378 CD0
379 CD0
380 CD0

381 CD0
382 CD0
383 CD0
384 CD0

385 CD0
386 CD0
387 CD0
388 CD0

389 CD0
390 CD0
391 CD0
392 CD0

393 CD0
394 CD0
395 CD0
396 CD0

397 CD0
398 CD0
399 CD0
400 CD0

401 CD0
402 CD0
403 CD0
404 CD0

405 CD0
406 CD0
407 CD0
408 CD0

409 CD0
410 CD0
411 CD0
412 CD0

413 CD0
414 CD0
415 CD0
BEGIN IF IF
CALL VMapar TO GET INPUT (Scheduling) Parameters IF IF
CALL Xpget to Retrieve Values for Interface Table Inputs IF IF
VERIFY RELATIONAL OPERATOR INPUT AS VALID and TRANSLATE IT TO A CODE IF
EREXIT IF RELATIONAL OPERATOR INVALID : ERRO3: IF IF
CALL Xievl to EVALUATE THE RELATIONAL EXPRESSION IF IF
IF THE EXPRESSION IS FALSE, THEN IF
CALL Xisch to locate the ELSE or ENL' : CORRESPONDING TO THIS IF IF
AND SET SEQUENCE RESET NUMBER IF IF
ELSE IF
CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE IF IF
ENDIF IF
EXIT IF IF
: ERRO3: CALL Xismg 'INVALID RELATIONAL OPERATOR - MUST BE #, >, =>, >=,  =, < , =, OR => IF IF
END IF IF
FORTAN CALLING PROCEDURE

VALUE = XIEVL (OPRN1), RELAT, OPRN2)

XIEVL COMPARES TWO REAL SINGLE PRECISION VALUES UNDER A SPECIFIED
RELATION AND RETURNS A FUNCTION VALUE OF LOGICAL TRUE OR FALSE.

INPUT
OPRN1 - FIRST REAL VALUE TO COMPARE
OPRN2 - SECOND REAL VALUE TO COMPARE
RELAT - RELATION CODE AS FOLLOWS
 0 - NOT EQUAL
 1 - GREATER THAN
 2 - GREATER THAN OR EQUAL
 3 - EQUAL
 4 - LESS THAN OR EQUAL
 5 - LESS THAN

OUTPUT - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

INTERNAL VARIABLES

TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS
FOLLOWS

OPRN1 - OPRN2
   CODE RELATION + 0 -

   OPRN1 - OPRN2
   RELATION + 0 -

   0  #  T  F  T
   1 >  T  F  T
   2 >= T  F  F
   3 = T  T  F
   4 <= T  F  T

   (BY COMPLEMENTING CODE > 2)

THEORETICAL FOR OPRN1 - OPRN2 POSITIVE (BITS 0-2) 1, 0, 1
   FOR OPRN1 - OPRN2 ZERO (BITS 3-5) 0, 0, 1
   AND FOR OPRN1 - OPRN2 NEGATIVE (BITS 6-8) 1, 0, 0
   (BITS 9-15 ARE NOT USED) IF RELAT IS 0, 1, OR 2 OR THE
   COMPLEMENT IF RELAT IS 3, 4, OR 5

REFERENCED ROUTINES

XEXT

NO CHECKS FOR VALID RELATION CODES OR OVERFLOW/UNDERFLOW ARE MADE
1 BEGIN XIEVL
2   CASE (+: ; :0: , :-: ) DIFFERENCE OF OPRND1 AND OPRND2
3       1+: SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)
3       0: SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
3       1: SET FIELD OFFSET TO SIX (BITS 6-8 OF TTABLE)
4   END CASE
4   IF RELATH > 2 (BOTTOM OF TRUTH TABLE), THEN
4       COMPLEMENT TTABLE
4       DECREMENT RELATH BY 3
4   ENDIF
5   ADD RELATH TO FIELD OFFSET (INDEXES TO CORRECT 4TH FOR RELATH AND DIFFERENCE)
5   SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
5 END XIEVL
**FORTRAN CALLING PROCEDURE**

**CALL ZISCH (SRCFG, PARMS)**

**ZISCH** is called to locate the corresponding ELSE/ENDIF in the current sequence table. The return parameters for the FBS manager are output.

**INPUTS**

**SRCFG** - Flag indicating origination of this call
  - 0 => called by IF to find matching ELSE or ENDIF
  - 1 => called by ELSE to find matching ENDIF

**COMMON**

**DEBUG** - Flag indicating whether online debug to be output
  - 0 => no debug
  - > 0 => debug

**PARMS** - Schedule in parameters from the FBS manager
  - (1) - Logical unit no. of the FBS user
  - (2) - SERDSP, index into sequence table (&SERSTAB) of the current command

**OUTPUTS**

**PARMS** - Parameters to be returned to FBS manager via XPRT
  - (1) - 0 => continue normal processor execution sequence
  - > 0 => execute sequence no. given by PARMS(2) next
  - 0 => abnormally terminate processor execution sequence
  - (2) - Sequence no. to be executed next if PARMS(1) = 3

**Routines Used**

**EXEC**

**Routines Used**

**EXEC**
560 1 BEGIN XISCH
561 2 RETRIEVE $SETA3 FROM THE AVA USING XVPAW
562 3 STARTSEARCH UNTIL ALL COMMANDS IN $SEATAB
563 4 EXIT IF CURRENT COMMAND IS FOUND
564 5 SET NUMBER OF IF NESTS TO 1
565 6 STARTSEARCH FROM NEXT COMMAND IN $SEATAB UNTIL ALL FOLLOWING COMMANDS
566 7 IF COMMAND IS FOR ENDF, THEN
567 8 DECREMENT NUMBER OF IF NESTS BY 1
568 9 ELSE
569 10 IF COMMAND IS FOR IF PROCESSOR, THEN
570 11 INCREMENT NUMBER OF IF NESTS BY 1
571 12 ELSE
572 13 IF CALLED BY IF PROCESSOR, AND
573 14 COMMAND IS FOR ELSE PROCESSOR, THEN
574 15 ERREXIT IF THIS IS THE END OF $SEATAB:ERROR1:
575 16 IF NUMBER OF IF NESTS IS 1, THEN
576 17 DECREMENT NUMBER OF IF NESTS TO 0
577 18 ENDF
578 19 ENDIF
579 20 ENDF
580 21 EXIT IF NUMBER OF IF NESTS IS 0
581 22 SET SEQUENCE RESET NUMBER (RPARMS(2)) TO SEQUENCE NUMBER OF THE
582 23 NEXT COMMAND IN THE TABLE
583 24 ENDLOOP
584 25 ERREXIT:ERROR1:
585 26 ENDF
586 27 ENDSERCH
587 28 ENDLCP
588 29 ERREXIT:ERROR4:
589 30 ENDSERCH
590 31 1 EXIT XISCH
591 32 :ERROR1: CALL X2MSG - 'IF CANNOT "E EXECUTED WITHOUT MATCHING ENDF''
592 33 :ERROR4: CALL X2MSG - 'SYSTEM ERROR - NO $SEATAB'
593 34 1 END XISCH
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
### OPRTTL F.

**FPMR-27R2** RL72-8 10/05/78 08:39:23

END PACK. TEXT=144,LOC=1,SYM=24,REL=5,ABS=1

### OPMTTL F.

**FPMR-191897-7DSIPSL.(1) ELEMENT TABLE**

<table>
<thead>
<tr>
<th>D NAME</th>
<th>VERSION</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ #</th>
<th>SIZE-PRE,TEXT</th>
<th>(CYCLE WORD)</th>
<th>PSRMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>53</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:54:07</td>
<td>2</td>
<td>30</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:54:32</td>
<td>3</td>
<td>2</td>
<td>55</td>
<td>1</td>
<td>1875</td>
</tr>
<tr>
<td>XUPDL</td>
<td></td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>03:37:22</td>
<td>4</td>
<td>84</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAP</td>
<td></td>
<td>ELT SYMB</td>
<td>09 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FAP</td>
<td></td>
<td>ELT SYMB</td>
<td>09 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>280</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>05:47:05</td>
<td>8</td>
<td>109</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>10 NOV 77</td>
<td>06:19:22</td>
<td>9</td>
<td>394</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>23:22:28</td>
<td>10</td>
<td>55</td>
<td>5</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>16 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>373</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>15 FEB 78</td>
<td>19:17:22</td>
<td>12</td>
<td>824</td>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>12:48:36</td>
<td>13</td>
<td>627</td>
<td>5</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>22 FEB 78</td>
<td>23:50:36</td>
<td>14</td>
<td>120</td>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>COMM:N</td>
<td></td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>22:20:04</td>
<td>15</td>
<td>200</td>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>11 MAR 78</td>
<td>03:16:51</td>
<td>16</td>
<td>2</td>
<td>19</td>
<td>5067</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>11 MAR 78</td>
<td>03:17:36</td>
<td>17</td>
<td>1</td>
<td>5</td>
<td>5068</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:29</td>
<td>18</td>
<td>101</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:51</td>
<td>19</td>
<td>334</td>
<td>5</td>
<td>22</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>11:45:48</td>
<td>20</td>
<td>384</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>18 MAY 78</td>
<td>09:12:58</td>
<td>21</td>
<td>1548</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>22</td>
<td>403</td>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:39</td>
<td>23</td>
<td>504</td>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:42</td>
<td>24</td>
<td>151</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>25</td>
<td>118</td>
<td>5</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>26</td>
<td>1201</td>
<td>5</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:30</td>
<td>27</td>
<td>800</td>
<td>5</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:59</td>
<td>28</td>
<td>326</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**EXECUTABLE PROGRAM POOL**