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

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

XE - EXECUTIVE FIXED COMMON (GLOBAL)
XB - EXECUTIVE DYNAMIC BLOCK (SUBSTA LEVEL DEPENDENT)
XS - EXECUTIVE SCRATCH SPACE (VOLATILE ACROSS ALL CALLS)

TO FOS ROUTINES EXCEPT XR,...

********

XE CONTENTS

INTEGER

# CARTPG
# CLASMD
# COMPR
# FLGS
# PROCG(64)
# SEREN
# SERST
# TABMD
# TKMLNG

TOKEMS(32)

DIMENSION

# INTNM(3)

EQUIVALENCE

# (XE(1) , DU)
# (XE(3) , QUA)
# (XE(5) , MASSTA)
# (XE(7) , SEGMN(1))
# (XE(11) , SEREN)
# (XE(13) , INTNM(1))
# (XE(17) , REUPTR)
# (XE(19) , TKMLNG)
# (XE(25) , TDMNG)
# (XE(31) , EXTND)
# (XE(143), NPROC)
# (XE(145), COMPR)
# (XE(145), COMPR)

CARTRG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

CLASMD - EXECUTIVE/MASTER REQUEST BLOCK CLASS I/O NUMBER

COMBUF - TERMINAL COMMUNICATIONS BUFFER

(1) - NUMBER OF TOKENS IN BUFFER

(2) - NUMBER OF USED WORDS IN BUFFER

COMPR - TOKENS REPRESENTING USER'S RESPONSE

EXTND - EXECUTION # WHERE EXECUTION IS TO END ( RETURN TO ? )

FLGSS - EXECUTIVE FLAG WORD (0-OFF, 1-ON)

BITS 0-10 UNUSED

11 PROCESSOR ON-LINE DEBUG
12 MANAGER ON-LINE DEBUG
13 EXECUTIVE ON-LINE DEBUG
14 PRODUCE A DUMP ON ALL TERMINATIONS
15 MANAGER REQUEST TRANSACTION TRACE FLAG

INTNM - NAME OF INTERFACE TABLE INPUT TO INTERFACE TABLE EDITOR
OR ASSOCIATED WITH PROCESSOR EXECUTED IN MANUAL, SEMI OR
AUTO-WITH-TRACE MODE (FIRST WORD = 0 IF DEFAULT INTERFACE
TABLE).

LU - LOGICAL UNIT NUMBER OF TERMINAL BEING SUPPORTED BY THIS
EXECUTIVE.

MST - EXECUTIVE MASTER STATE FLAG (LEVEL LAST PASSED CONTROL BY
XEXEC). RESET TO ZERO BY LEVELS RETURNING TO DIRECTIVE
LEVEL.

BITS 0-9 - NOT USED

BIT 10-13 - DIRECTIVE CONTROL MODE IF BITS 14-15 = 0

- LIST
- TEC
- SAVE
- RECALL
- DELETE
- RENAME
- COPY
- CLEAR
- OFF
- STORE
- RESTORE
- UNLOAD
- LOAD
- BATCH

BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR
IF BITS 14-15 = 1

- INITIALIZATION FROM DIRECTIVE
- INITIALIZATION FOR REENTRY

BIT 12-13 - EXECUTION CONTROL MODE IF BITS 14-15 = 1

- MANUAL
- SEMI-AUTOMATIC
- AUTOMATIC-T
- AUTOMATIC

BIT 14-15 - EXECUTIVE STATE

- DIRECTIVE LEVEL
- EXECUTION CONTROL LEVEL
- SEQUENCE TABLE EDIT LEVEL
- INTERFACE TABLE Edit LEVEL

HOPROC - NUMBER OF PROCESSORS IN LIBRARY

OLDIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

PROCNAME - NAME OF PROCESSOR FOR WHICH INTERFACE TABLE EDITOR WAS

INKED OR BEING EXECUTED IN MANUAL, SEMI OR AUTO-WITH-
TRACE MODE

QUAL - USE: UNIQUE FILE NAME QUALIFIER (SIXTH CHARACTER OF NAME)

REBUFF - BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SDG 6.2.7)

REPTST - POINTER TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUFF
(D INDICATES REBUFF EMPTY) OR RETURN CODE FORM XREQ

SEGNAM - TERMINATING SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED
IN SEMI OR AUTO MODE AS PASSED TO THE MANAGER

SEGNAM - NAME OF SEQUENCE TABLE INPUT TO SEQUENCE TABLE EDITOR OR

EXECUTED IN SEMI OR AUTO MODE

SEOP - POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SEMI

SEOP - ON AUTO-WITH-TRACE MODE

SERST - INITIAL SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN

SEMII OR AUTO MODE

SUBST - EXECUTIVE SUB-STATE FLAG (LEVEL IN COMMUNICATION WITH

USER TERMINAL). SET TO LEVEL TO BE INITIALIZED NEXT OR
ZERO IF LEVEL INITIALIZATION FAILS.
XB CONTENTS (SEQUENCE TABLE EDIT LEVEL)

INTEGER DEBUG, RENUM, PROMPT, PRUNFREE, OLDTAB, TARTTAB, TABSZ, XLIBF, XLIBO

* DEBUG = FLAG SET TO VALUE OF EXECUTIVE DEBUG OPTION (BITS 13 - 15)
  OF Flags (KE(4))

* RENUM = CODE IN XBUF TO FIRST TABLE ENTRY TO BE LISTED OR DELETED
  0 = READ ENTRY AT TABSIZE
  = 1 => INSERT A NEW ENTRY IN FRONT OF TABSIZE

* PROMPT = FLAG INDICATING TO REQUEST WHAT TYPE OF EDIT TO PERFORM:
  0 = REPLACE ENTRY AT TABSIZE
  = 1 => INSERT A NEW ENTRY IN FRONT OF TABSIZE

* PRUNFREE = NUMBER OF ENTRIES (INCLUDING THESE MARKS FOR DELETION)

* OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* TARTTAB = NAME OF SEQUENCE TABLE BEING CREATED

* XLIBO = NAME OF SEQUENCE TABLE BEING CREATED BY ESPRM

* XLIBF = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

* XLIBS = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBO

* XLIBT = CURRENTLY IN XLIBF

* DEBUG = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* PRUNFREE = LENGTH IN WORDS OF PROMPT CREATED BY ESPRM

* PROMPT = current prompting mode:
  = 1 => UPDATE MODE ( # )
  = 2 => CREATE OR RECREATE MODE ( $ 200+: )
  = 3 => ALL MODE ( $ 200+PROC, TABLE:

* OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* TARTTAB = NAME OF SEQUENCE TABLE BEING CREATED

* XLIBO = NAME OF SEQUENCE TABLE BEING CREATED BY ESPRM

* XLIBF = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

* XLIBS = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBO

* XLIBT = CURRENTLY IN XLIBF

* DEBUG = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* PRUNFREE = LENGTH IN WORDS OF PROMPT CREATED BY ESPRM

* PROMPT = current prompting mode:
  = 1 => UPDATE MODE ( # )
  = 2 => CREATE OR RECREATE MODE ( $ 200+: )
  = 3 => ALL MODE ( $ 200+PROC, TABLE:

* OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* TARTTAB = NAME OF SEQUENCE TABLE BEING CREATED

* XLIBO = NAME OF SEQUENCE TABLE BEING CREATED BY ESPRM

* XLIBF = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

* XLIBS = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBO

* XLIBT = CURRENTLY IN XLIBF

* DEBUG = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* PRUNFREE = LENGTH IN WORDS OF PROMPT CREATED BY ESPRM

* PROMPT = current prompting mode:
  = 1 => UPDATE MODE ( # )
  = 2 => CREATE OR RECREATE MODE ( $ 200+: )
  = 3 => ALL MODE ( $ 200+PROC, TABLE:

* OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* TARTTAB = NAME OF SEQUENCE TABLE BEING CREATED

* XLIBO = NAME OF SEQUENCE TABLE BEING CREATED BY ESPRM

* XLIBF = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

* XLIBS = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBO

* XLIBT = CURRENTLY IN XLIBF

* DEBUG = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* PRUNFREE = LENGTH IN WORDS OF PROMPT CREATED BY ESPRM

* PROMPT = current prompting mode:
  = 1 => UPDATE MODE ( # )
  = 2 => CREATE OR RECREATE MODE ( $ 200+: )
  = 3 => ALL MODE ( $ 200+PROC, TABLE:

* OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

* TARTTAB = NAME OF SEQUENCE TABLE BEING CREATED

* XLIBO = NAME OF SEQUENCE TABLE BEING CREATED BY ESPRM

* XLIBF = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

* XLIBS = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBO
251 1 CD XB CONTENTS (INTERFACE TABLE EDIT LEVEL)
252 1 CD INTEGER
254 1 CD * ARGNO
255 1 CD * BITNUM
256 1 CD * DEBUG
257 1 CD * DFLAG
258 1 CD * PARMLEN
259 1 CD * PROMPT(30)
260 1 CD * VERNON
261 1 CD * NKBLNG
262 1 CD DIMENSION
263 1 CD * ISIZE(9)
264 1 CD EQUIVALENCE
266 1 CD * (XB(23), COMFLG)
268 1 CD * (XB(24), ICLASS)
269 1 CD * (XB(25), LSTFLG)
270 1 CD * (XB(26), VERNON)
271 1 CD * (XB(27), DEBUG)
272 1 CD * (XB(28), ICLASS)
273 1 CD * (XB(29), NOSAV)
274 1 CD * (XB(30), IARGA)
275 1 CD * (XB(31), IDIM)
276 1 CD * (XB(32), ISIZE)
277 1 CD * (XB(33), IDFLG)
278 1 CD * (XB(34), JSUB)
279 1 CD * (XB(35), DFLAG)
280 1 CD * (XB(36), DFLAG)
281 1 CD * (XB(37), IARGB)
282 1 CD * (XB(38), ARGTR)
283 1 CD * (XB(39), ISUB)
284 1 CD * (XB(40), IOFLG)
285 1 CD * (XB(41), PARMLEN)
286 1 CD * (XB(42), PROMPT(31))
287 1 CD * (XB(43), PROMPT(32))
288 1 CD * (XB(44), PROMPT(33))
289 1 CD * (XB(45), PROMPT(34))
290 1 CD * (XB(46), LENEF)
291 1 CD * (XB(47), ITYPE)
292 1 CD * (XB(48), BITNO)
293 1 CD * (XB(49), BITMX)
294 1 CD * (XB(50), NOSAV)
295 1 CD * (XB(51), IARGA)
296 1 CD * (XB(52), LENEF)
297 1 CD * (XB(53), ISIZE)
298 1 CD * (XB(54), IARGB)
299 1 CD * (XB(55), IARGC)
300 1 CD * (XB(56), SNEWTAS)
301 1 CD * (XB(57), NNEWTAS)
302 1 CD * (XB(58), XWKBUF)
303 1 CD * (XB(59), NKBLNG)
304 1 CD ARGNO - CURRENT ARGUMENT'S NO. (I.E., 1 TO 64)
305 1 CD ARGTR - INDEX IN WNBF TO ARGUMENT PERSIST FOR CURRENT ARGUMENT
306 1 CD BITNO - BIT NO. IN BIT MASK CURRENTLY BEING PROCESSED
307 1 CD BITMX - BIT NO. IN BIT MASK OF NEXT DIFFERENT BIT
308 1 CD COMFLG - FLAG SET TO 1 IF A CONTINUATION PROMPT FOR CURRENT ARGUMENT
309 1 CD HAS BEEN EXPLICITLY REQUESTED VIA A TRAILING COMA OR IS
310 1 CD IMPLICITLY REQUIRED BECAUSE INCOMPLETE ELEMENTS REMAIN BEYOND
311 1 CD LAST ENTERED ELEMENT OF THIS ARGUMENT
312 1 CD DEBUG - FLAG SET TO VALUE OF DEBUG FLAG (BITS 13-15 OF XE(4))
313 1 CD DIRECT - ASCII ARRAY OF VALID INTERFACE TABLE DIRECTIVES
314 1 CD DFLG - FLAG SET TO VALUE OF DATA BIT FOR THIS ARGUMENT
315 1 CD IARG - INDEX IN WKBUF TO CURRENT ARG
316 1 CD ICLASS - CLASS OF THIS ARGUMENT
317 1 CD IOFLG - FLAG SET TO I/O BITS FOR THIS ARGUMENT
318 1 CD IRET - GENERAL RETURN FLAG (0=OK, -1=ERROR, 5='EXIT')
319 1 CD ISIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
320 1 CD ISIZE - NO. OF WORDS/ELEMENT FOR EACH ALLOWED ITYPE VALUE
321 1 CD ISUB - CURRENT EFFECTIVE SUBSCRIPT FOR THIS ARGUMENT
322 1 CD ITYPE - TYPE FLAG FOR THIS ARGUMENT
1  1 i  JSUB - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
2  1 CO  LONEFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
3  1 CD  LISTLU - LU TO WHICH PRINT SHOULD GO (USED WHEN
4  1 CC  LIST DIRECTIVE CALLS XILSD OR XICHRS)
5  1 CD  LITDSP - DISPL. TO LITERAL DATA FOR THIS ARGUMENT
6  1 CD  LITDWH - INDEX IN WKBUF TO END OF LITERAL DATA
7  1 CD  LILN - LENGTH OF LITERAL DATA AREA OF WKBUF
8  1 CD  LITPFR - INDEX IN WKBUF TO START OF LITERAL DATA
9  1 CD  LITLZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
10  1 CD  LISTL0 - FLG USED TO DETERMINE ORIGIN OF A
11  1 CD  CALL TO XILSD OR XICHRS.
12  1 CD  = 0, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
13  1 CD  AN INTERFACE TABLE
14  1 CD  = 2, CALLED FROM INTERFACE TABLE EDITOR TO LIST ARGUMENT
15  1 CD  DATA, PARAMETER OR INCOMPLETE INDICATORS
16  1 CD  = 4, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
17  1 CD  A DATA ELEMENT'S VALUE(S)
18  1 CD  MODSAV - PREVIOUS VALUE OF PRMTMD WHILE PRMTMD = 4 (CONTINUE)
19  1 CD  NARG - INDEX IN WKBUF TO START OF SHORT PROMPTS
20  1 CD  NDBOTM - INDEX IN WKBUF TO BIT MASK WORD(S) FOR THIS ARGUMENT
21  1 CD  WENDTAB - ASCII NAME OF TABLE BEING GENERATED
22  1 CD  NOTM - NO. OF BIT MASKS ASSOCIATED WITH THIS ARGUMENT
23  1 CD  NUMARG - NO. OF ARGUMENTS IN THIS INTE. TABLE
24  1 CD  NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
25  1 CD  PRMLN - LENGTH (IN WORDS) OF PROMPT BUILT
26  1 CD  PRMTMD - CURRENT PROMPTING MODE
27  1 CD  = 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
28  1 CD  = 3, PROMPT WITH A LIST OF CURRENT VALUES
29  1 CD  = 4, CONTINUATION OF SAME ARGUMENT
30  1 CD  = 5, PROMPT WITH "-" (CREATE MODE)
31  1 CD  PROMPT - ASCII ARRAY FOR PROMPT BUILT
32  1 CD  SFLAG - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
33  1 CD  WRLN - LNGTH OF WKBUF
34  1 CD  WMBNG - LENGTH OF WKBUF
35  1 CD  WKBNG - WORKING BUFFER FOR INTERFACE TABLE BEING EDITED
36  1 CD  ORGANIZED AS:
37  1 CD  HEADER (7 WORDS)
38  1 CD  ARGUMENT CHARACTERISTICS (NUMARG*7 WORDS)
39  1 CD  ARGUMENT PROMPTS (NUMARG*3 WORDS)
40  1 CD  LITERAL DATA AND PARAMETER SUBSCRIPTS (LILLEN WORDS)
3.0 FDS EXECUTIVE MESSAGES

The list of messages generated by the Executive are presented in this section.
# FDS Error Messages

1 * ATTENTION FUNCTION

2  * XA01 MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION

3  * XA02 USER INITIATED INTERRUPT ENTER REQUEST - KILL(), STATUS(), RETURN()

4  * XA03 ERROR, LU IS NOT SIGNED ON TO FDS

5  * XA04 FDS ATTEMPT FUNCTION TERMINATING

6  * XA05 FDS MANAGER SIGNALED TO TERMINATE BOTTOM ASSOC TASK

7  * XA06 FDS EXECUTIVE ACTIVE - NO ACTION TAKEN

8  * XA07 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT

9  * XA08 MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN

10  * XB BATCH JOB CREATION

11  * XC CONFIGURATION PROGRAMS

12  * XC01 LU 'NN' SIGNED ON TO FDS

13  * XC02 LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED

14  * XC03 LU 'LU' IS AN INVALID LU

15  * XC04 FDS CURRENTLY AT MAX USER'S. CANNOT SIGN ON

16  * XC05 'LU' IS ALREADY SIGNED ON TO FDS

17  * XC06 CANNOT FIND 'NAME' ID - SIGN ON TERMINATED

18  * XC07 ENTER VALID USER ID (A-Z)

19  * XE MANAGER

20  * XM01 INVALID REQUEST II FROM 'NAME'

21  * XM02 FDS SIGN OFF FOR LU 'LU'

22  * XM03 INVALID BACK CHAIN FOR CURRENT PROCESSOR

23  * XM04 'H' TRACKS NOT AVAILABLE FOR DWA

24  * XM05 NO DWA SPACE FOR 'NAME'

25  * XM06 NO AWA FOR DWA DIRECTOY

26  * XM07 I/O ERROR FOR DWA, STATUS = 'NNNNNN'

27  * XM08 TOC SEARCH ERROR, PHASE 3 COMPRESS

28  * XP PROCESSOR SERVICES

29  * XP01 TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED

30  * XP02 DATA AREA OVERFLOW

31  * XP03 SUBSCRIPT OUT OF RANGE

32  * XP04 INVALID ENTRY AFTER SUBSCRIPT

33  * XP05 INVALID REPETITION ENTRY

34  * XP06 TOO MANY NESTED REPETITIONS

35  * XP07 INVALID SUBSCRIPT

36  * XP08 RESPONSE IS TOO LONG FOR BUFFER

37  * XP09 INVALID RESPONSE ON OR AFTER COLUMN NN

38  * XP10 PROCESSOR INITIALIZATION ERROR

39  * XP11 AWA ACCESS FAILURE FOR .......

40  * XP12 PARAMETER I/O INCONSISTENCY

41  * XP13 INVALID PARAMETER REQUEST

42  * XP14 INVALID ODE FILE ATTRIBUTE(S)

43  * XP15 RETRIEVAL OF TOO MUCH DATA REQUESTED

44  * XV SYSTEM SERVICES

45  * XV01 'ID-NAME' NAMED PROGRAM MADE A PM REQUEST RESERVED FOR MANAGER

46  * XV02 'ID-NAME' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER REPLY

47  * XV03 'ID-NAME' 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

1 ****
2 1 CD1
3 1 CD1
4 1 CD1
5 1 CD1
6 1 CD1
7 1 CD1
8 1 CD1
9 1 CD1
10 1 CD1
11 1 CD1
12 1 CD1
13 1 CD1
14 1 CD1
15 1 CD1
16 1 CD1
17 1 CD1
18 1 CD1
19 1 CD1
20 1 CD1
21 1 CD1
22 1 CD1
23 1 CD1
24 1 CD1
25 1 CD1
26 1 CD1
27 1 CD1
28 1 CD1
29 1 CD1
30 1 CD1
31 1 CD1

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
BEGIN PDLIST
1 BEGIN PDLIST
2 INITIALIZE SEQUENCE NUMBER, LEVEL AND DEFINITION TABLE COUNTER
3 DO UNTIL END-OF-FILE INPUT
4 READ RECORD
5 INCREMENT SEQUENCE NUMBER
6 IF COLUMN 1 = "
7 THEN
8 SET PAGE EJECT IN IMAGE
9 IF REMAINDER OF IMAGE NOT BLANK
10 THEN
11 SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
12 ENDIF
13 ELSE
14 CLEAR LEVEL INCREMENT
15 IF COLUMN 1 NOT = 
16 THEN
17 IF FIRST CHARACTER = : - INDICATES LABEL:
18 THEN
19 SET LINE SKIP IN IMAGE
20 GENERATE DEFINITION TABLE ENTRY FOR LABEL
21 ELSE
22 CALL FSTDLY TO GET FIRST WORD OF PDL
23 LOOKUP FIRST WORD IN KEY WORD TABLE
24 KEY WORD TABLE CONTAINS
25 1 BEGIN - BEGIN SECTION INDICATOR
26 2 IF - SECTION INDICATOR
27 3 DO - SECTION INDICATOR
28 4 DOFOR - SECTION INDICATOR
29 5 DOUNTI - SECTION INDICATOR
30 6 DOUNTIL - SECTION INDICATOR
31 7 END - SECTION INDICATOR
32 8 START - SECTION INDICATOR
33 5 STARTS - SECTION INDICATOR
34 0 ELSE - SECTION SEPARATOR
35 11 THEN - SECTION SEPARATOR
36 12 EXIT - SECTION SEPARATOR
37 13 EXITIF - SECTION SEPARATOR
38 14 OR - SECTION SEPARATOR
39 15 ORELSE - SECTION SEPARATOR
40 16 ENDOLOD - SECTION SEPARATOR
41 17 END - END OR END LOOP?
42 18 ENDIF - SECTION TERMINATOR
43 19 ENDDO - SECTION TERMINATOR
44 20 ENDCAS - SECTION TERMINATOR
45 21 ENDOSEA - SECTION TERMINATOR
46 IF KEY WORD LOCATED
47 THEN
48 CASE LOCATION :BEGIN; :SECOND; :SECOND; :SECOND; :SECOND; :SECOND; :
49 (:SECOND; :SECOND; :SECOND; :SECOND; :SECOND; :SECOND; :
50 (:SECOND; :SECOND; :SECOND; :SECOND; :SECOND; :SECOND; :
51 (:END; :TERM; :TERM; :TERM; :TERM; :TERM; :TERM; :
52 BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
53 SET LEVEL INCREMENT = 1
54 :SECOND: SET LEVEL INCREMENT = 1
:SEP: SET LEVEL INCREMENT = 1
DECREMENT LEVEL
:END?: CALL FSTWDR TO GET NEXT WORD OF PDL
DECREMENT LEVEL
IF WORD = LOOP
THEN
SET LEVEL INCREMENT = 1
ENDIF
:TERM: DECREMENT LEVEL
ENDCASE
ENDIF
ENDIF
COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
ELSE
SET INDENTATION FACTOR = 1
ENDIF
CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
INPUT RECORD AND ID FIELD
OUTPUT IMAGE
APPLY LEVEL INCREMENT
CLEAR PAGE EJECT
ENDIF
ENDDO
CALL SORT1 TO ORDER DEFINITION TABLE
OUTPUT DEFINITION TABLE
END POLIST
147 1 CD: ALGEBRAIC AND/OR ALPHABETIC ARRAY SORT
148 1 CD: ENTRY POINT INTO SUBROUTINE SORT2
150 1 CD: THIS ROUTINE WAS EXTRACTED FROM THE NDAS SUBMONITOR PROGRAM FOR USE IN
151 1 CD: POLIST. DOCUMENTATION MAY BE FOUND IN 'LEVEL II NDAS PROTOTYPE
152 1 CD: MONITOR PROGRAM DOCUMENT (PART II),' TRW NOTE NO. 74-FMT-937,
156 1 CD: **********
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.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
<th>Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>179</td>
<td>XSENM    SEMI - AUTOMATIC NODE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>180</td>
<td>XSTO     STORE SEQUENCE TABLE IN ASRTAB</td>
<td>DIRECT</td>
</tr>
<tr>
<td>181</td>
<td>XTHP     TEMPORARY EXECUTION OF ONE ENTRY WITH RINTAB</td>
<td>DIRECT</td>
</tr>
<tr>
<td>182</td>
<td>XZ       UTILITY PROCESSORS</td>
<td>DIRECT</td>
</tr>
<tr>
<td>184</td>
<td>ASSGN    ASIGN PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>185</td>
<td>DBDSP    DATA BOX DISPLAY PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>186</td>
<td>DEFIN    DEFINE PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>187</td>
<td>DO       CONDITIONAL LOOP IN SEQUENCE TABLE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>188</td>
<td>ELSE     EXECUTION POINT FOR FALSE IF CONDITION</td>
<td>DIRECT</td>
</tr>
<tr>
<td>189</td>
<td>ENUEF    TERMINATES AN IF STRUCTURE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>190</td>
<td>ENDCDO   TERMINATES A DO LOOP STRUCTURE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>191</td>
<td>ENDCS    END SCAN PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>192</td>
<td>IF       CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES</td>
<td>DIRECT</td>
</tr>
<tr>
<td>193</td>
<td>SCAN     SCAN PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>194</td>
<td>XICHR    CHARACTER OBJECT STORE FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>195</td>
<td>XZPT     FIND ANY TOKEN IN A SYMBOLIC STRING</td>
<td>DIRECT</td>
</tr>
<tr>
<td>196</td>
<td>XZDIN    DATA BOX DISPLAY INPUT PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>197</td>
<td>XZDMK    DATA BOX DISPLAY CONSTRAINT MASKER</td>
<td>DIRECT</td>
</tr>
<tr>
<td>198</td>
<td>XZDOT    DATA BOX DISPLAY OUTPUT ROUTINE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>199</td>
<td>XZOPT1   DATA BOX DISPLAY PASS 1 PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>200</td>
<td>XZOPT2   DATA BOX DISPLAY PASS 2 PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>201</td>
<td>XZEV     PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS</td>
<td>DIRECT</td>
</tr>
<tr>
<td>202</td>
<td>XZFCN    FIND PROCESSOR CLASS NUMBER</td>
<td>DIRECT</td>
</tr>
<tr>
<td>203</td>
<td>XZFCN    FUNCTIONAL OPERATIONS FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>204</td>
<td>XZFE     FREE OBJECT STORE FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>205</td>
<td>XZFD     FIXED OBJECT STORE FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>206</td>
<td>XZISP    REMOVE DPL. BLANKS &amp; BLANK FILL</td>
<td>DIRECT</td>
</tr>
<tr>
<td>207</td>
<td>XZLSS    SYMBOLIC STRING SYNTAX ERROR LISTER</td>
<td>DIRECT</td>
</tr>
<tr>
<td>209</td>
<td>XZMSG    FPS PROCESSOR MESSAGE ROUTINE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>210</td>
<td>XZOPR    MATH OPERATIONS FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>211</td>
<td>XZPES    DATA CONVERSION AND STORAGE FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>212</td>
<td>XZPS1    PASS 1 SUBROUTINE FOR ASSGN PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>213</td>
<td>XZPS2    PASS 2 SUBROUTINE FOR ASSGN PROCESSOR</td>
<td>DIRECT</td>
</tr>
<tr>
<td>214</td>
<td>XZSCI    SEARCHES SEQUENCE TABLE FOR IF STRUCTURES</td>
<td>DIRECT</td>
</tr>
<tr>
<td>215</td>
<td>XZSYM    SYMBOL TABLE INTERFACE FOR ASSGN</td>
<td>DIRECT</td>
</tr>
<tr>
<td>216</td>
<td>XZSYT    SYMBOL TABLE MAINTENANCE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>217</td>
<td></td>
<td>DIRECT</td>
</tr>
</tbody>
</table>
BEGIN XATTN

FUNCTION

FDX ATTENTION TASK-

USER MAY REQUEST THE CURRENT STATUS OF FDS

FOR THE TERMINAL, TO TERMINATE CURRENT PROCESSOR,

OR (IF FDS HAS TERMINATED) TO TEAR DOWN THE

FDX STRUCTURE FOR THIS TERMINAL.

NOTES

THE ID FOR XATTN IS CONNECTED TO THE USER'S

TERMINAL AT FDS SIGN-ON BY USING THE EN.

FOR THE DEVICE, THE FUNCTION IS DISCONNECTED

AT FDS SIGN-OFF.

XATTN

XATTN

XATTN

XATTN

XATTN

XATTN

XATTN

XATTN
SAVE EOT ADDRESS(IN BREG ON ENTRY)
CALL ERLU(BREG) GET LU IN ASCII & BINARY
STARTSearch UNTIL LAST STATUS TABLE ENTRY
EXITIF STBLU EN LU
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STMG)
IF MANAGER IS DORMANT, THEN
WRITE "**XAO- MANAGER HAS TERMINATED;"
REPLY TO CONTINUE TERMINATION;
READ(LU) ** WAIT FOR REPLY **
LOCK ON THE FDS TABLE RESOURCE
CALL SLIRM DISABLE
IF STB(EXECUTIVE ADDRESS .NE. 0, THEN
IF STB(CURRENT) .NE. STBEX, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XMCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STMG(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 SLIMX ENABLE
CALL MESS 'OFF,BOTTOM'
CALL SLIM Disable
SET BOTTOM TO NEXT-AT
ENDIF
ENDIF
CALL SLIST MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBER
ENDIF
DECREMENT NUMBER ACTIVE(STBA)
GET EOT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBER)
CLEAR STBER
CLEAR MANAGER'S ID, STMG, & STBL
ENABLE....(VIA A JMP TO EXEC(DISPATCHER))
RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS
CLEAR LOCK ON FDS TABLE
ELSE ** MANAGER IS STILL ALIVE **
WRITE "**XAO- USER INITIATED INTERRUPT?"
WRITE 'ENTER REQUEST- KILL(0), STATUS(S), OR RETURN(BLANK)' READ (LU) REQUEST
IF REQUEST IS KILL OR 0, THEN
PERFORM KILL
ELSE
IF REQUEST IS STATUS OR S, THEN
PERFORM KSTAS
ENDIF
ENDIF
ENDIF
ELSE
WRITE "**XAO- ERROR LU IS NOT SIGNED-ON TO FDS"
ENDIF
WRITE "**XAO FDS ATTENTION FUNCTION TERMINATING"
END
140 BEGIN XAKILL
141 2  # TERMINATE CURRENT FDS FUNCTION
142 2  # IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
143 2  # ON NEXT RETURN VIA A PAM
144 2  # IF THE EXEC IS ACTIVE-DO NOTHING
145 2  # IF A PROCESSOR IS ACTIVE- USE RTE MESS TO OFF THE PROCESSOR
146 2  # SET MANAGER'S ID ADDRESS(STBMG)
147 2  # IF STATUS OF MANAGER IS NOT WAIT, THEN
148 2  # SET TERMINATE FLAG IN STB-ENTRY
149 2  # WRITE "***XAO6 FDS MANAGER SIGNALED TO TERMINATE SEQUENCE'
150 2  ELSE
151 2  # IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
152 2  # WRITE "***XAO6 FDS EXECUTIVE ACTIVE; NO ACTION TAKEN' 
153 2  ELSE
154 4  # PERFORM XATHM(CURRENT) FIND BOTTOM AT
155 4  # IF BOTTOM AT IS D.RTR OR SPM THEN
156 5  # WRITE "***XAO6 MANAGER IS WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN.' 
157 4  # EXIT XAKIL
158 4  ELSE
159 5  # IF RETURNED BOTTOM IS MANAGER THEN
160 6  # IF MANAGER IS NOT WAITING ON A PROGRAM THEN
161 7  # WRITE "***XAO6 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN.' 
162 6  # EXIT XAKIL
163 6  ELSE
164 7  # SET RETURN PARAMETER TO PROCESSOR ABENDED
165 7  # INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
166 7  # CALL BLIST TO REACTIVATE MANAGER
167 6  # ENDIF
168 5  ENDIF
169 5  WRITE "**XAO7 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT.' 
170 5  IF RETURNED BOTTOM WAS NOT MANAGER THEN
171 6  # SET NAME IN 'OFF' COMMAND
172 6  CALL MESS TO 'OFF' THE PROCESSOR
173 5  # ENDIF
174 4  # ENDIF
175 3  # ENDIF
176 2  # ENDIF
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  4 AND SONS FATHER ID POINTS TO CURRENT
183  5 SET SON AS CURRENT
184  6 ENDDO
185  7 SET BOTTOM AS CURRENT
186  8 END XABTM
BEGIN XCNFG

*01 FDS CONFIGURATION MANAGER
*01 INITIATES AN FDS SYSTEM FOR A REQUESTED TERMINAL OR
*01 TERMINATES AN FDS SYSTEM FOR A REQUESTED TERMINAL
*01 INITIATED VIA
*01 RU,FDS,LU,ID,CNA SIZE,PARMS,OPTIONS
*02 INPUTS
*02 LOGICAL UNIT(LU) FOR THE REQUESTED TERMINAL,
*02 A PARM TO DENOTE INITIATION OR TERMINATION
*02 A ONE CHARACTER USERID
*02 A DEBUG OPTION INDICATOR
*02 THE NUMBER OF DNA TRACKS
*03 OUTPUTS
*03 INITIATION-
*03 A BLANK ID-SEGMENT WILL BE CONSTRUCTED FOR THE FDS MANAGER,
*03 AND FDS EXECUTIVE
*03 THE ERT FOR THE REQUESTED LU WILL BE CONNECTED
*03 TO THE FDS ATTENTION TASK
*03 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
*03 WILL BE UPDATED TO REFLECT THE INITIATION
*03 TERMINATION-
*03 THE FDS ID SEGMENTS WILL BE RETURNED TO BLANK STATUS
*03 THE ERT FOR THE LU WILL BE REINSTATED
*03 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
*03 WILL BE UPDATED FOR THE TERMINATION

RIPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
29 2   * PARM=LU, ID, DWA SIZE, PARM (ON OR OFF), OPTIONS
30 2   * SWITCH PARM PARS AROUND SO THAT
31 2   * NOW PARM=LU,P2 (ON OR OFF),ID,OPTS,DWA SIZE
32 2   * FOR COMPATABILITY TO BUILD 1.
33   CALL RMPAR (PARMS)
34 2   IF LU IS .LT. 0, OR
35 3   .GT. LMAX (1633), 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 2   ELSE
40 3   IF PARM P2 IS OFF THEN
41 4   PERFORM XCOFF SIGN OFF
42 3   ELSE
43 4   PERFORM XCON SIGN ON
44 2   ENDIF
45 2   ENDIF
46 2   :XEXIT
47 2   CALL EXEC PROGRAM TERMINATION
48 1 END XCMFG
1 BEGIN XCOM
2  * SGN ON A USER TO FDS
3  
4  UNTIL VALID USER ID (P3)
5  
6  IF ID NOT A - 2, THEN
7  "XCOB ENTER VALID ID (A - 2)"
8  
9  EXIT: XCEA
10  
11  ENDIF
12  
13  IF USER ID IS BEING USED, THEN
14  "***XCOB LU 'Lu' IS CURRENTLY USING ID 'ID'-SIGN ON REJECTED"
15  
16  EXIT: XCEA
17  
18  ENDIF
19  
20  IF FDS RESOURCE NUMBER NOT DEFINED, THEN
21  CALL RAMQ (GLOBAL ALLOCATE, LOCAL SET)
22  
23  ELSE
24  CALL RAMQ (LOCAL SET)
25  
26  ENDIF
27  
28  IF NUMBER SIGNED ON (STRBAC) .EQ. MAXIMUM USERS (STHN), THEN
29  ISSUE MESSAGE "***XCOB FDS CURRENTLY AT MAX USER'S.
30  
31  ELSE
32  DO FOR STBNK (NUMBER OF FDS ENTRIES)
33  IF ENTRY'S LU'SLTR (LU) .EQ. REQUESTING LU (P1) THEN
34  IF THIS ENTRY IS AVAILABLE, THEN
35  "SET AS CURRENT-ENTRY-ADDRESS"
36  
37  ELSE
38  "BECOME PRIVILEGED & DISABLED"
39  
40  "CALL SLIB"
41  
42  "STARTSEARCH WHILE NUMBER-FOUND LT NUMBER-NEEDED"
43  
44  "SEARCH ID-SEGMENTS USING KEYS (1657)"
45  IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
46  "SET ID ADDRESS OF XEXEC"
47  INCREDENT NUMBER-FOUND
48  ELSE
49  IF XMG NOT FOUND AND THIS ID .EQ. XMG, THEN
50  "SET ID ADDRESS OF XMG"
51  INCREDENT NUMBER-FOUND
52  ELSE
53  IF XATM NOT FOUND AND THIS ID .EQ. XATM, THEN
54  "SET ID ADDRESS OF XATM"
55  INCREDENT NUMBER-FOUND
56  ELSE
57  IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
58  "SET ID ADDRESS OF FIRST-BLANK"
59  INCREDENT NUMBER-FOUND
60  ELSE
61  IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
62  "SET ID ADDRESS OF SECOND-BLANK"
63  INCREDENT COUNT
64  
65  THEN
66  ENDIF
67  
68  ENDIF
69  
70  ENDIF
71  
72  "EXIT: XCEA"
73  
74  EXIT: XCEA
75  
76  EXIT: XCEA
77  
78  EXIT: XCEA
CALL BLISRX  ENABLE
ISSUE MESSAGES "***XCIO? CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP
BUILD ENTRY IN VSMTB
* SET LU IN'Q STBLU
SET LU IN ASCII INTO STBLA
SET USER'S ID INTO STBID
SET ADDRESS OF FIRST-BLANK INTO STBMT
SET ADDRESS OF SECOND-BLANK INTO STBEX
INCREMENT ACTIVE COUNT(STBAC)
BUILD XPGNN & XEXEC
MOVE PRIORITY THRU DISC ADDRESS FROM XMGH TO FIRST-BLANK
TURN ON TH BIT
SET NAME TO XPGNN
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
TURN ON TH BIT
SET NAME TO XEXEC
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE EAT OVERLAY
SAVE EAT VALUE IN STBEN
SET ID ADDRESS OF XATHM INTO EAT
ENDF
SET INITIAPARMS INTO ID OF XMGH
SCHEDULE XMGH VIA BLIST
CALL BLIST
CALL BLISRX  ENABLE
IF DFS HAS A FATHER, THEN
CALL MESSAGE 'OFF,Father'
ENDF
ISSUE MESSAGE "***XCIO1 LU "MM" SIGNED ON TO DFS"
ENDSEARCH

EXEC
CALL RMNR (LOCAL CLEAR)
ENDF
CALL EXEC TERMINATE
END XCON
Reproducibility of the original page is poor.
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 .ENTR, XDLS, XERTN

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 XDLS

ENDCASE

CALL XERTN TO RETURN FROM SEGMENT

END XDCLD
40 1 #00  FORTRAN CALLING PROCEDURE
41 1 #00  CALL XECLS (XDLFL)
43 1 #00  ********
44 1 #01  XDLFL 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 (BITS 10-13 CONTAIN A 1-8 INDEX INTO A LIST OF
59 1 #02  DIRECTIVES)
60 1 #02  ********
61 1 #04  INTERNAL VARIABLES
62 1 #04  LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
64 1 #04  65 1 #05  NOTES
66 1 #05  USES AXENT, XDEL, XDCOP, XDELE, XDOFF, XDREC, XDREM, XDSAV,
68 1 #05  XTOX, XETRN
69 1 #05  XDLFL IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
71 1 #05  CONTAINING THE REFERENCED DIRECTIVES
72 1 #05  74 1 *
75 1 *
76 1 *
77 1 BEGIN XDLFL
78 2  EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
80 3  :TOC: CALL XTOC
81 3  :SAVE: CALL XDSAVE
82 3  :RECA: CALL XDREC
83 3  :DELE: CALL XDELE
84 3  :REMA: CALL XDREM
85 3  :COPY: CALL XDOP
86 3  :CLEA: CALL XDCLF
87 3  :OFF: CALL XDOFF
88 2  ENDCASE
89 2  CALL XETRN TO RETURN FROM SEGMENT
90 1 END XDLFL
1 *DO     FORTRAN CALLING PROCEDURE
2 *DO     CALL XELDS (XDCLU)
3 *DO
4 ********
5 *DO
6 *DO     XDCLU DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
7 *DO     REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
8 *DO     STORE
9 *DO     RESTORE
10 *DO     UNLOAD
11 *DO     LOAD
12 *DO     BATCH
13 *DO
14 ********
15 *DO     INPUT
16 *DO     XE COMON - MASSTA (BITS 10-13 CONTAIN A 9-13 INDEX INTO A LIST OF
17 *DO     DIRECTIVES)
18 *DO
19 *DO
20 ********
21 *DO     INTERNAL VARIABLES
22 *DO     LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
23 *DO
24 *DO
25 ********
26 *DO     NOTES
27 *DO     USES .ENTR, XDSTO, XDRES, XDUNL, XDLOA, XDBAT, XERTH
28 *DO     XDCLU IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
29 *DO     CONTAINING THE REFERENCED DIRECTIVES
30 *DO
31 ********
32 *DO
33 *DO
34 *DO
35 *DO
36 *DO
37 BEGIN XDCLU
38 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7
39 CASE (:STOR, :REST, :UMLO, :LOAD, :BATCH) INDEX
40     :STOR: CALL XDSTO
41     :REST: CALL XDRES
42     :UMLO: CALL XDUNL
43     :LOAD: CALL XDLOA
44     :BATCH: CALL XDBAT
45     END CASE
46     CALL XERTH TO RETURN FROM SEGMENT
47 END XDCLU
FORTRAN CALLING PROCEDURE

CALL XDCLE

XDCLE PURGES ALL DATA FROM THE AMA. THE ONLY ELEMENTS REMAINING ARE PERMANENT SYSTEM TABLES AND DATA BASE FILES (CHAINS 1 & 6)

INPUT

X COMMON - XTCRG, FLGSS, LU
MANAGER - AMA HEADER AND TOC (SEE XMANA)

OUTPUT

X COMMON - REBUF, REOPTR, COMBUF-SCRATCH
XB COMMON - SCRATCH BEYOND XB(200)
XS COMMON - SCRATCH
MANAGER - COMMANDS TO PURGE AMA AND RESTORE CHAINS 0 AND 8

LOCAL VARIABLES

AVA - AMA HEADER AND TOC RECEIVED FROM MANAGER
HC1 - TOC CHAIN 1 HEAD
HC2 - TOC CHAIN 2 HEAD
HC3 - TOC CHAIN 3 HEAD
HC5 - TOC CHAIN 5 HEAD
HC6 - TOC CHAIN 6 HEAD
HC7 - TOC CHAIN 7 HEAD
HC8 - TOC CHAIN 8 HEAD
NEXT - INDEX TO NEXT ALLOCATABLE EIGHT WORD ENTRY IN QUEUE
NMAX - MAXIMUM SIZE OF QUEUE
ORG - BASE REFERENCE ADDRESS FOR AMA ADDRESSES
QUEUE - BUFFER FOR CONSTRUCTING AMA RESTORE REQUESTS FOR NON-DELETED ITEMS
TOCMAX - MAXIMUM SIZE OF TOC ACCOMODATABLE BY XDCLE

NOTES

USES EXEC, IAND, PRN, PURGE, XREQ, XRIG, XRMOV, XRMSG, XOR6, XRGN

IN THE EVENT THAT A COMPACTED AMA 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, FBS WILL BE TERMINATED.
IN ORDER TO PROTECT USER DISK FILES.
**Module Description**

**FUNCTION**

`CALL XDCOP`

**Explanation**

The XDCOP module is a FORTRAN calling procedure that processes a user request to copy a sequence table, interface table, data element, or queue. It is only applicable for those tables or elements logged in the user's area that can be copied.

**Example Declaration**

```fortran
COMMON XE - CARTAG, COMBUF, QUAL, RECPTR, TOKENS
```

**Additional Details**

- **Input Parameters**
  - `IDCB`: EXEC BUFFER
  - `ERR`: FILE MANAGER ERROR RETURN
  - `HNAME`: NEW FMGR FILE NAME
  - `FNAME`: OLD FMGR FILE NAME
  - `POINTER`: POINT TO OLD NAME IN COMBUF
  - `ENTRY`: TOC ENTRY FOR DATA BASE RENAME
  - `EOF`: EOF RETURN FROM FMGR

- **Output Parameters**
  - `BUFF`: USER BUFFER FOR EXEC I/O
  - `FMTYPE`: FMGR FILE TYPE
  - `PARM1`: FIRST BYTE OF RECPTR
  - `PARM2`: SECOND BYTE OF RECPTR
  - `PREFIX`: PREFIX FOR FILE NAME
  - `SCRTY`: SECURITY CODE
  - `SIZE`: FILE SIZE IN BLOCKS

**Usage Example**

```fortran
CALL XDCOP
```
BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY

EXIT TO :TOCERR: IF ERROR IS INDICATED

ELSE

BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY

BUILD MANAGER REQUEST FOR DATA RETRIEVAL

CALL XREQ

EXIT TO :TOCERR: IF ERROR IS INDICATED

CALL EXEC TO GET TOC ENTRY

BUILD MANAGER REQUEST TO ALLOCATE NEW TABLE

BUILD MANAGER REQUEST TO STORE TABLE

IF CLASS IS INTERFACE TABLE, THEN

CALL EXEC TO WRITE NEW TABLE NAME TO SAM

BUILD MANAGER REQUEST TO STORE NEW NAME IN TABLE

ENDIF

CALL XREQ

EXIT TO :TOCERR: IF ERROR IS INDICATED

ENDIF

1 EXIT XDOP

:SYNTAX: CALL XRMSG "SYNTAX ERROR ..." AND EXIT

:CLASER: CALL XRMSG "INVALID CLASS DESIGNATOR ..." AND EXIT

:NAMEERR: CALL XRMSG "NEW NAME IS INVALID ..." AND EXIT

:MAXERR: CALL XRMSG "AUTHORIZED LIMIT ..." AND EXIT

:INVALS: CALL XRMSG "NOB CANNOT BE ..." AND EXIT

:FILEERR: CALL XRMSG "FILE ACCESS ERROR ... ON ......." AND EXIT TO :END:

:TOCERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:

:TYPERR: CALL XRMSG "INCONSISTENT FILE TYPE ..."

:END:

IF POB HAS BEEN LOGGED IN XPDG, THEN

CALL XDGRD TO DELETE POB FROM XPDG

ENDIF

IF A NEW FILE HAS BEEN BUILT, THEN

PURGE NEW FILE

CLOSE OLD FILE

ENDIF

1 END XDOP
445 1 BEGIN XD08A
446 2 STORE RETURN ADDRESS
447 3 CALL .ENTRY TO SET UP CALLING ARGUMENTS
448 4 CALL XNLCK FOR EXCLUSIVE USE OF XD08
449 5 CALL OPEN FOR EXCLUSIVE USE OF XD08
450 1 EXIT TO :FILERR: IF ERROR RETURNED
451 2 COMPUTE RECORD NUMBER FOR USER'S DIRECTORY (QUAL-77B)/2+1
452 2 CALL READ FOR RECORD COMPUTED
453 1 EXIT TO :FILERR: IF ERROR RETURNED
454 2 DETERMINE PART OF RECORD TO BE USED
455 2 IF REQUEST IS FOR ADD, THEN
456 3 EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
457 3 START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
458 3 EXIT TO :NAMERR: IF ENTRY NAME MATCHES PARAMETER NAME
459 4 ENDLOOP
460 3 ENDSERACH
461 3 STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
462 3 INCREMENT # OF CURRENT ENTRIES
463 2 ELSE
464 3 START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
465 3 EXIT IF ENTRY NAME MATCHES PARAMETER NAME
466 4 ENDLOOP
467 3 EXIT TO :NAMERR:
468 4 ENDSERACH
469 3 IF REQUEST IS FOR VERIFY, THEN
470 4 STORE WORD 3 OF ENTRY IN ISIZE
471 4 ELSE
472 4 REPLACE ENTRY WITH LAST ENTRY
473 4 STORE ENTRIES IN LAST ENTRY
474 4 DECREMENT # OF CURRENT ENTRIES
475 3 ENDF
476 2 ENDF
477 2 CALL WRITF TO WRITE RECORD TO XD08
478 1 EXIT TO :RETURN;
479 2 :NAMERR: SET IERR = 1 AND EXIT TO :RETURN;
480 2 :MAXERR: SET IERR = 2 AND EXIT TO :RETURN;
481 2 :FILERR: SET IERR = FPGA ERROR CODE
482 2 :RETURN: CALL CLOSE FOR XD08
483 2 CALL XNLCK TO RETURN RESOURCE #
484 1 END XD08A
486 1 C**********
487 1 CARD
488 1 C00
489 1 C00 FORTRAN CALLING PROCEDURE
490 1 C00 CALL XDELE
491 1 C00
492 1 C**********
493 1 C01 XDELE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT
494 1 C01 SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE DATA.
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 COMMON XE - COMBUF, COMPTR, LU, QUAL, TOKENS
503 1 C02
504 1 C02
505 1 C02
506 1 C**********
507 1 C03 OUTPUT
508 1 C03 COMMON XE - REGBUF
509 1 C03
510 1 C03
511 1 C03
512 1 C**********
513 1 C05 NOTES
514 1 C05
515 1 C05 ROUTINES USED
516 1 C05
517 1 C05
518 1 C05 EXEC
519 1 C05 JAND
520 1 C05 PURGE
521 1 C05 XDDBB
522 1 C05 XREN
523 1 C05 XREXT
524 1 C05 XRIG
525 1 C05 XMOW
526 1 C05 XMSG
527 1 C05 XRPCX
528 1 C05 XRVM
529 1 C05 XAFPK
530 1 C05 XDBUG
531 1 C05
532 1 C**********
1 BEGIN XDDE
2 DO WHILE END-OF-STATEMENT NOT REACHED
3 PROCESSING EACH ELEMENT SPECIFIED
4 ERROR IF COMMA IS NOT NEXT SYNTAX ELEMENT :ERROR:
5 IF CLASS DESIGNATOR IS SPECIFIED, THEN
6 SET REQUESTED CLASS APPROPRIATELY (B, S, I, D, OR F)
7 ELSE
8 SET REQUESTED CLASS TO BE (D)
9 ENDIF
10 IF DATA BASE TO BE DELETED, THEN
11 BUILD AND ISSUE ASA MANAGER REQUEST FOR TOC ENTRY
12 ENDIF
13 IF ELEMENT IS NOT A MASTER DATA BASE, THEN
14 IF ELEMENT IS A PERSONAL DATA BASE, THEN
15 CALL XBDDO TO DELETE THIS PDN FROM XPDD
16 IF ERROR IS RETURNED, THEN
17 CALL XRMSG - "FILE ACCESS ERROR 0... XPDD"
18 SET ERROR FLAG
19 ENDIF
20 BUILD AND ISSUE ASA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
21 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
22 CALL XRMSG - "xxxxx NOT FOUND"
23 SET ERROR FLAG
24 ENDIF
25 IF CLASS IS DATA BASE (C), OR
26 CLASS IS DRE (F), THEN
27 IF ERROR FLAG IS NOT SET, THEN
28 CALL XBFFM TO CONSTRUCT FILE NAME
29 CALL XBFFP TO PURGE THE FILE
30 IF RETURN CODE FROM PURGE, THEN
31 CALL XRMSG - "FILE ERROR NAME XXXXXX"
32 ENDIF
33 ENDIF
34 ELSE
35 CALL XRMSG - "... IS A MBD. NOT DELETED."
36 ENDIF
37 ENDDO
38 1 EXIT TO :RETURN:
39 2 :ERROR: CALL XRMSG - "SYNTAX ERROR"
40 2 :RETURN:
41 1 END XDDE
CALL XLIS (TOCLST)

XLIST PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECODING 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) TOTIPS, XB(158) TOTWD

INTERNAL XB COMMON USED:

XB(151) ABLG - (INTEGER, 1 WORD) ABORT FLAG
XB(152) ABLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TOTIPS - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
XB(158) TOTWD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
XB(198) RETC - (INTEGER, 1 WORD) XTCOM RETURN CODE
XB(199) BACTL - (INTEGER, 1 WORD) CLS OF DATA BEING SEARCHED
XB(200) HNMPTR - (INTEGER, 1 WORD) POINT TO NAME IN COMBUF
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCLST

COMMON USED:

EQUIVALENCE,

+ (RE(5), MASSTA), (RE(8), EOS)
+ (RE(85), NAME)
+ (RE(113), COMMA)
+ (RE(144), COMTRA)
1 BEGIN XDLIS
2  DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
3    TURN ERFLG OFF

   :RTM1:
3    DO UNTIL EOS IS SENSED IN COMBUS
4      ERREXIT IF TOKEN IS NOT "NAME" TO :ERR1:
5        SAVE INDEX TO NAME FIELD
6      INCREMENT TO NEXT TOKEN
7      IF TOKEN IS A HYPHEN THEN
8          ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:
9      INCREMENT TO NEXT TOKEN
10     DECODE CLASS NAME (I, S, D, F)
11     ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:
12     SET CLASS TO CLASS SPECIFIED
13     ELSE
14       SET CLASS TO DATA ELEMENT
15      ENDIF
16     IF XDLIS CALLED FROM STORE THEN
17       ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:
18     ENDIF
19     ERREXIT IF NAME/CLASS ENTRY NOT FOUND IN TOC TO :ERR2:
20     CALL ARSET TO TURN STORE/RESTORE BIT ON
21     INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
22 ENDDO

   :RTN2:
3    IF ERROR FLAG IS ON THEN
4      CALL XCOM TO REPROMPT USER TO CONTINUE
5      ERREXIT IF RESPONSE IS X TO :ERR3:
6    ENDIF
7    EXIT XDLIS

   :ERR1:
2    SET ERROR FLAG ON
3    CALL XRMSS TO DISPLAY SYNTAX ERROR
4    GO TO :RTM2:

   :ERR2:
2    IF ERROR FLAG IS OFF THEN
3      TURN ERROR FLAG ON
4    CALL XRMSS TO DISPLAY NOT STORED/RESTORED MESSAGE
5    ENDIF
6    CALL EXEC TO DISPLAY ELEMENT NAME
7    GO TO :RTM1:

   :ERR3:
2    SET ABFLG TO ABORT STORE/RESTORE OPERATION
3    END XDLIS
FORTRAN CALLING PROCEDURE

CALL XLSST

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

INPUT

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

OUTPUT

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

USES ROUTINES - EXEC, XEINT, XILSD, XILST,

XREQ, XREX, XRMOV, XRXMSG,

XRPCK, XRPK, XSLST, OPEN,

CLOSE, READY

NOTES
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>783</td>
<td>1 CD*******</td>
</tr>
<tr>
<td>784</td>
<td>1 CD0 FORTRAN CALLING PROCEDURE</td>
</tr>
<tr>
<td>785</td>
<td>1 CD0 CALL XDOFF</td>
</tr>
<tr>
<td>786</td>
<td>1 CD0</td>
</tr>
<tr>
<td>787</td>
<td>1 CD0</td>
</tr>
<tr>
<td>788</td>
<td>1 CD0</td>
</tr>
<tr>
<td>789</td>
<td>1 CD0</td>
</tr>
<tr>
<td>790</td>
<td>1 CD0 XDOFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,</td>
</tr>
<tr>
<td>791</td>
<td>1 CD0 DELETES ALL DRE AND TDBF FILES LOGGED IN THE ANA,</td>
</tr>
<tr>
<td>792</td>
<td>1 CD0 PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR</td>
</tr>
<tr>
<td>793</td>
<td>1 CD0 RETURNS NORMAL PARS TO THE FDS MANAGER AND TERMINATES</td>
</tr>
<tr>
<td>794</td>
<td>1 CD0 NORMALLY VIA RTE.</td>
</tr>
<tr>
<td>795</td>
<td>1 CD0</td>
</tr>
<tr>
<td>796</td>
<td>1 CD0</td>
</tr>
<tr>
<td>797</td>
<td>1 CD0</td>
</tr>
<tr>
<td>798</td>
<td>1 CD0 INPUT</td>
</tr>
<tr>
<td>799</td>
<td>1 CD0</td>
</tr>
<tr>
<td>800</td>
<td>1 CD0 COMMON XE = LU, FLAGS, QUAL, RERBUF</td>
</tr>
<tr>
<td>801</td>
<td>1 CD0 COMMON XB = ORG = ORIGIN ADDRESS OF ANA. USED TO CALCULATE</td>
</tr>
<tr>
<td>802</td>
<td>1 CD0 INDICES INTO 'ANA' FROM ADDRESS POINTERS</td>
</tr>
<tr>
<td>803</td>
<td>1 CD0</td>
</tr>
<tr>
<td>804</td>
<td>1 CD0 OF TOC ENTRIES</td>
</tr>
<tr>
<td>805</td>
<td>1 CD0 ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND</td>
</tr>
<tr>
<td>806</td>
<td>1 CD0 TOC RETRIEVED VIA XREQ</td>
</tr>
<tr>
<td>807</td>
<td>1 CD0</td>
</tr>
<tr>
<td>808</td>
<td>1 CD0</td>
</tr>
<tr>
<td>809</td>
<td>1 CD0</td>
</tr>
<tr>
<td>810</td>
<td>1 CD0 INTERNAL VARIABLES</td>
</tr>
<tr>
<td>811</td>
<td>1 CD0</td>
</tr>
<tr>
<td>812</td>
<td>1 CD0</td>
</tr>
<tr>
<td>813</td>
<td>1 CD0 COMMON X5 = POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC</td>
</tr>
<tr>
<td>814</td>
<td>1 CD0 ENTRY. MOST SIGNIFICANT (BIT 15) BIT</td>
</tr>
<tr>
<td>815</td>
<td>1 CD0 SET TO INDICATE END-OF-CHAIN</td>
</tr>
<tr>
<td>816</td>
<td>1 CD0 INDEX = VALUE COMPUTED FROM POINTER TO BE</td>
</tr>
<tr>
<td>817</td>
<td>1 CD0 FORTRAN INDEX INTO 'ANA' FOR NEXT TOC</td>
</tr>
<tr>
<td>818</td>
<td>1 CD0 ENTRY</td>
</tr>
<tr>
<td>819</td>
<td>1 CD0</td>
</tr>
<tr>
<td>820</td>
<td>1 CD0 NOTES</td>
</tr>
<tr>
<td>821</td>
<td>1 CD0</td>
</tr>
<tr>
<td>822</td>
<td>1 CD0 ROUTINE USED - EXEC, PURGE, XDST, XPIT, XREQ, XREXT, XRI6,</td>
</tr>
<tr>
<td>823</td>
<td>1 CD0 XRN0, XRF5G, XRSPC, XRSET, XRUPK, XICOM,</td>
</tr>
<tr>
<td>824</td>
<td>1 CD0</td>
</tr>
<tr>
<td>825</td>
<td>1 CD0 XUD6G, XVABN</td>
</tr>
</tbody>
</table>
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL XRT TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR ORDE FILES IS NOT NEGATIVE THEN
6 DO UNTIL CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEYOND END OF TOC Buffer THEN
8 OUTPUT "XDOF TOC TOO LARGE, PURGE INCOMPLETE"
9 EXIT DO
10 ENDF
11 CALL XRQFN TO CREATE FILE NAME 'XXXXX'
12 CALL PURGE TO SCRATCH FILE
13 SET ORDE 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 "XDOF TOC TOO LARGE, PURGE INCOMPLETE"
20 EXIT DO
21 ENDF
22 IF TYPE OF DATA BASE IS UTDD, THEN
23 CALL XRQFN TO CREATE FILE NAME 'XXXXX'
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 XRDATA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XUDBG
32 ENDF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVBGN - NO RETURN FROM THIS CALL
35 ENDF
36 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
37 CALL XPRT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE EXEC
38 ENDF
39 RETURN
40 END XDOFF
FORTRAN CALLING SEQUENCE:

CALL XDBE (DATBUF, DBDBCB)

XDBE HANDLES THE RESTORING OF DATA ELEMENTS, SEQUENCE TABLES AND INTERFACE TABLES

INPUTS FROM CALLING SEQUENCE:

DATBUF - (INTEGER, 140 WORDS) BUFFER USED TO READ IN RECORDS
OF DATA FROM DATA BASE FILE
DBDBCB - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DCB

INPUTS FROM XB COMMON:

XB(154) ALLFLG, XB(170) ENDBL, XB(182) TOTSIZ,
XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

DATBUF, DBDBCB

OUTPUTS IN XB COMMON:

XB(151) ADBFLG, XB(152) ERFLG, XB(165) TOCHDS,
XB(169) TOPBLK, XB(170) ENDBL, XB(171) TOCHDS,
XB(201) NOTOC

INTERNAL XB COMMON USED:

XB(151) ADBFLG - ABORT FLAG
XB(152) ERFLG - ERROR MESSAGE FLAG
XB(155) ALLFLG - RESTORE ALL UDB FLAG
XB(155) DEBUG - DEBUG FLAG
XB(157) XFILE- DD FILE NAME
XB(162) DATBLK- BLOCK # WHERE DATA ITEM BEGINS
XB(163) DATINDEX- WORD INDEX INTO DATBUF WHERE DATA BEGINS
XB(164) ERR- ERROR FLAG FOR FMNR CALLS
XB(165) TOCDBS- # WORDS OF TOC IN TOCBUF (NOTOC *B) + B
XB(166) NOBLKS- # BLOCKS TO READ
XB(167) LEM - # WORDS OF DATA TO MOVE
XB(168) DATEND- BLOCK # WHERE DATA ITEM ENDS
XB(169) TOPBLK- BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBL- BLOCK # OF LAST BLOCK READ
XB(171) TOCINDEX- INDEX INTO TOC OF CURRENT DATA ITEM
XB(172) IL - # WORDS OF DATA TO READ
XB(176) TOCENT- DATA BASE TOC ENTRY
XB(201) NOTOC - # TOC ENTRIES IN TOCBUF
XB(201) TOCBUF- TOC BUFFER
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-ORDE 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 XRNMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
13 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
14 CALL READ TO READ ENOUGH TO FILL DATBUF
15 ERREXIT IF READ ERROR TO :ERR1:
16 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
17 ENDIF
18 ENDIF
19 BUILD SAVE REQUEST TO ALLOCATE AND STORE DATA
20 CALL EXEC TO WRITE DATA TO SAN
21 ERREXIT IF ERROR FROM EXEC TO :ERR2:
22 IF SAVE REQUEST BUFFER IS FULL THEN
23 CALL XDRDE TO ISSUE REQUEST
24 EXIT XDRDE IF ERROR IN XDRDE
25 ENDIF
26 ENDIF
27 END
28 1 EXIT XDRDE
29 :ERR1:
30 CALL XR16 TO CONVERT ERROR CODE TO ASCII
31 CALL XRMSG TO DISPLAY ERROR MESSAGE (200)
32 GO TO :ERR3:
33 :ERR2:
34 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
35 :ERR3:
36 SET ABFLG TO SAY ABD SIG RESTORE
37 1 END XDRDE
**Fortran Calling Sequence:**

```fortran
CALL XDRUF (DATBUF, DBDCB)
```

**Inputs in Calling Sequence:**

- `DATBUF` - (INTEGER, 1408 WORDS) BUFFER USED TO READ IN RECORDS OF ODBE FILES FROM DATA BASE FILE
- `DBDCB` - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DBC

**Outputs in Calling Sequence:**

- `DBDCB`

**Inputs in XDB Common:**

- `XB(152) ERFILE, XB(154) ALLFLG, XB(156) TOVWDS,
- `XB(160) TPOOL, XB(170) ENDBLK, XB(171) TOCNRS,
- `XB(182) TOTSIZ`

**Outputs in XDB Common:**

- `XB(151) ADFLG`

**Internal XDB Common Used:**

- `XB(150) ADFLG` - ABDT FLAG
- `XB(152) ERFILE` - ERROR MESSAGE FLAG
- `XB(154) ALLFLG` - RESTORE ALL UDBO FLAG
- `XB(155) DEBUG` - DEBUG FLAG
- `XB(159) FILENAME` - DB FILE NAME
- `XB(162) DATBLK` - BLOCK # WHERE DBE BEGINS
- `XB(163) DATINDEX` - WORD INDEX INTO DATBUF WHERE DBE BEGINS
- `XB(164) ERR` - ERROR FLAG FOR FNGR CALLS
- `XB(165) TOCWDS` - WORDS OF TOC IN TOCBUF (NOTOC #8) + 8
- `XB(166) NOBLKS` - # BLOCKS OF UDBO/DRDE TO READ/WRIT
- `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) TOCNRS` - INDEX TO CURRENT TOC ENTRY
- `XB(172) DREDFIL` - DRDE FILE NAME
- `XB(175) IL` - # WORDS OF DATA TO READ/WRIT
- `XB(201) TOCBUF` - TOC BUFFER

**Common Used:**

- `XB(151)`
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 XRDF TO CREATE THE DRE FILE
1065 4 CALL CREFT TO CREATE THE DRE FILE
1066 4 IF THERE WAS A CREATE ERROR THEN
1067 5 IF ERROR FLAG IS OFF THEN
1068 6 SET ERROR FLAG ON
1069 6 CALL XRMG TO DISPLAY MAIN MESSAGE
1070 6 ENDIF
1071 7 CALL XR16 TO CONVERT ERROR CODE TO ASCII
1072 7 CALL XEXEC TO DISPLAY DRE NAME AND ERROR CODE
1073 4 ELSE
1074 5 CALL CLOSE TO CLOSE FILE
1075 5 ERREXIT IF CLOSE ERROR TO :ERR1:
1076 5 CALL OPEN TO OPEN DRE FILE AS TYPE 1
1077 5 ERREXIT IF OPEN ERROR TO :ERR1:
1078 5 DO UNTIL ALL BLOCKS OF DRE HAVE BEEN PROCESSED
1079 6 IF DATA FOR DRE IS IN DATAUF THEN
1080 7 CALL WRITF DATA TO DRE
1081 7 ERREXIT IF WRITF ERROR TO :ERR1:
1082 6 ELSE
1083 7 CALL READF TO READ NEXT BUFFER OF DATA
1084 7 ERREXIT IF READF ERROR TO :ERR2:
1085 7 SET INDICES INDICATING DATA IN BUFFER
1086 6 ENDIF
1087 5 ENDDO
1088 5 CALL CLOSE TO CLOSE DRE FILE
1089 5 ERREXIT IF CLOSE FAILED TO :ERR1:
1090 5 BUILD AWA REQUEST TO ALLOCATE DRE IN AWA
1091 5 IF AWA REQUEST BUFFER IS FULL THEN
1092 6 CALL XDRER TO MAKE REQUEST
1093 5 EXIT XRDF IF XRER ERROR
1094 5 ENDIF
1095 4 ENDDO
1096 3 ENDF
1097 2 ENDDO
1098 1 EXIT XRDF
1099 2 :ERR1:
1100 2 CALL PURGE TO PURGE DRE FILE
1101 2 :ERR2:
1102 2 CALL XRMG TO DISPLAY ERROR MESSAGE WITH FILE NAME
1103 2 SET ABFLG TO SAY ABORT RESTORE
1104 1 END XRDF
1106  1 C**********
1107  1 CD0    FORTRAN CALLING PROCEDURE
1108  1 CD0
1109  1 CD0
1110  1 CD0    CALL XDBD
1111  1 CD0
1112  1 C**********
1113  1 CD1    XDBD PROCESSES THE RECALL DIRECTIVE. A UDBD IS CREATED AND
1114  1 CD1    THE CONTENTS OF THE SPECIFIED PDB ARE COPIED TO IT.
1115  1 CD1
1116  1 CD1
1117  1 C**********
1118  1 CD2    INPUT
1119  1 CD2
1120  1 CD2    COMMON XE - CARTAG, COMBUF, COMPTR, FLAGS, LV, TOKENS
1121  1 CD2
1122  1 CD2    FILES - )XIXX (PDB FILE SPECIFIED)
1123  1 CD2
1124  1 CD2
1125  1 C**********
1126  1 CD3    OUTPUT
1127  1 CD3
1128  1 CD3    COMMON XE - REGBUF, REOPTR
1129  1 CD3
1130  1 CD3    FILES - )XIXX (UDBD FILE SPECIFIED)
1131  1 CD3
1132  1 CD3
1133  1 C**********
1134  1 CD4    INTERNAL VARIABLES
1135  1 CD4
1136  1 CD4
1137  1 CD4    DCBPDO - DCB FOR THE PDB FILE; ALLOCATED IN X8 COMMON;
1138  1 CD4    CONTAINS 1152 WORD BUFFER USED TO READ THE PDB
1139  1 CD4
1140  1 CD4    AND TO WRITE THE UDBD
1141  1 CD4    DCBUDB - DCB FOR THE UDBD FILE; ALLOCATED IN X8 COMMON
1142  1 C**********
BEGIN XREC
  1  SET STATUS FLAG INDICATING PROG & UTDO FILES NOT OPEN
  2  ERREXIT IF "." IS NOT NEXT TOKEN :ERROR:
  3  INCREMENT TO NEXT TOKEN
  4  ERREXIT IF NEXT TOKEN IS NOT "NAME" :ERROR:
  5  ERREXIT IF THIS NAME IS > 1 CHARACTERS :ERROR:
  6  ERREXIT IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
  7  RETAIN NAME AS PROG
  8  RETAIN CURRENT USER IN (QUALIFIER)
  9  INCREMENT TO NEXT TOKEN
 10  IF NEXT TOKEN IS "\r", THEN
   11  INCREMENT TO NEXT TOKEN
   12  ERREXIT IF NEXT TOKEN IS NOT "NAME" :ERROR:
   13  ERREXIT IF "NAME" IS GREATER THAN 1 CHARACTER :ERROR:
   14  ERREXIT IF QUALIFIER < "A" OR > "Z" :ERROR:
   15  SAVE IN AT (ZERO FILE) RIGHT-JUSTIFIED FORMAT AS QUALIFIER
   16  INCREMENT TO NEXT TOKEN
   17  ENDF.
  18  ERREXIT IF NEXT TOKEN IS NOT A "NAME" :ERROR:
  19  INCREMENT TO NEXT TOKEN
  20  ERREXIT IF NEXT TOKEN IS NOT A "NAME" :ERROR:
  21  ERREXIT IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
  22  RETAIN THIS NAME AS UTDO
  23  INCREMENT TO NEXT TOKEN
  24  ERREXIT IF NEXT TOKEN IS NOT EOS :ERROR:
  25  BUILD ARM REQUEST TO VERIFY EXISTENCE OF UTDO
  26  CALL XREQ TO PROCESS ARM REQUEST
  27  ERREXIT IF UTDO DOES ALREADY EXIST :ERROR:
  28  CALL XGETN TO VERIFY PROG AND RETRIEVE SIZE
  29  ERREXIT IF PROG DOES NOT EXIST :ERROR:
  30  ERREXIT IF FMGR ERROR RETURNED :ERROR:
  31  CALL XRNKN TO CULR PROG FILE NAME
  32  CALL OPEN TO OPEN PROG FILE
  33  ERREXIT IF OPEN FAILED :ERROR:
  34  SET STATUS FLAG INDICATING PBFD FILE OPEN
  35  BUILD ARM REQUEST TO ALLOCATE UTDO
  36  CALL XREQ TO PROCESS ARM REQUEST
  37  ERREXIT IF ALLOCATE REQUEST FAILS :ERROR:
  38  SET STATUS FLAG TO INDICATE UTDO FILE ALLOCATED IN AMA
  39  RESTORE CURRENT USER'S ID (QUALIFIER)
  40  CALL XRNKN TO BUILD UTDO FILE NAME
  41  CALL CREAT TO CREATE TYPE 1 UTDO FILE
  42  ERREXIT IF CREATE FAILED :ERROR:
  43  SET STATUS FLAG INDICATING UTDO FILE NOW OPEN
  44  DO FOR EACH BUFFER OF DATA IN PROG FILE
  45  CALL READF TO READ 1 BUFFER FROM PROG FILE
  46  ERREXIT IF READ ERROR :ERROR:
  47  CALL WRITF TO WRITE 1 BUFFER TO UTDO FILE
  48  ERREXIT IF WRITE ERROR :ERROR:
  49  ENDDO
  50  CALL CLOSE FOR PROG
  51  CALL CLOSE FOR UTDO
  52  EXIT XREC
  53  :ERROR: ISSUE MESSAGE - "SYNTAX ERROR. EXTRANEOUS DATA"
  54  :ERROR: ISSUE MESSAGE - "ILLEGAL UTDO NAME (NOT FOUND OR TOO LONG)"
2 | :ERROR7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
3 | :ERROR8: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
4 | :ERROR9: ISSUE MESSAGE - "INVALID PDB FILE NAME ..."
5 | :ERROR10: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
6 | :ERROR11: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
7 | :ERROR12: ISSUE MESSAGE - "AVA OVERFLOW - XXXX NOT LOGGED"
8 | :ERROR13: ISSUE MESSAGE - "XXXX ALREADY EXISTS"
9 | :ERROR14: ISSUE MESSAGE - "FILE ACCESS ERROR #-- XPDB"
10 | :RETURN:
11 | IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
12 | PURGE UTDB
13 | ENDIF
14 | IF STATUS FLAG INDICATES UTDB IS LOGGED IN AWA, THEN
15 | CALL XER TO DELETE UTDB FROM AWA
16 | ENDIF
17 | IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
18 | CALL CLOSE FOR PDB FILE
19 | ENDIF
20 | "ND XDREC
1220  1 C*******
1221  1 CD0  FORTRAN CALLING PROCEDURE
1222  1 CD0  CALL XOREN
1223  1 CD0  ********
1224  1 C01  XOREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE,
1225  1 CD0  INTERFACE TABLE, DATA ELEMENT, DREX, UTDB, OR PDG. ONLY
1226  1 CD0  THOSE TABLES OR ELEMENTS LOGGED IN THE USERS ANA ARE RENAMED.
1227  1 CD0  ********
1228  1 CD0  INPUT
1229  1 CD0  COMMON XE - CARTAG, COMBUF, COMPTR, QUAL, RERPTR, TOKENS
1230  1 CD0  ********
1231  1 CD0  OUTPUT
1232  1 CD0  COMMON XE - RERBUF
1233  1 CD0  ********
1234  1 CD0  INTERNAL VARIABLES
1235  1 CD0  COMMON XS - (2) DBTYPE: TYPE CODE FOR DATA BASE FILES
1236  1 CD0  (3) J: INDEX
1237  1 CD0  (4) JERR: FILE MANAGER ERROR RETURN
1238  1 CD0  (148) JIME: ERROR MESSAGE NUMBERS
1239  1 CD0  (150) NAME: NEW FBGR FILE NAME
1240  1 CD0  (153) NAMEP: POINTER TO NEW NAME IN COMBUF
1241  1 CD0  (154) NAME: OLD FBGR FILE NAME
1242  1 CD0  (157) NAMEP: POINTER TO OLD NAME IN COMBUF
1243  1 CD0  (158) TOCEN: TOC ENTRY FOR DATA BASE RENAME
1244  1 CD0  (166) TATELS: DATA CLASS CODE (STORED IN LEFT BYTE)
1245  1 CD0  COMMON XB - (201) IDCB: EXEC BUFFER
1246  1 CD0  ********
1247  1 CD5  EXTERNAL REFERENCES
1248  1 CD5  ROUTINES USED -
1249  1 CD5  EXEC
1250  1 CD5  JIAN
1251  1 CD5  KFVT
1252  1 CD5  MANP
1253  1 CD5  XDOMA
1254  1 CD5  XDUBD
1255  1 CD5  XREG
1256  1 CD5  XRMV
1257  1 CD5  XRMSG
1258  1 CD5  XRFSN
1259  1 CD5  XRFSR
1260  1 C****
BEGIN XDREN
EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
SAVE POINTER TO CURRENT NAME
INCREMENT TO NEXT TOKEN
IF TOKEN IS A SYMPHON, THEN
EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
DECODE CLASS NAME
EXIT TO :CLASER: IF CLASS SPECIFIED IS NOT VALID (S,T,F,B) ELSE
SET CLASS TO BE
ENDIF
EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
SAVE POINTER TO NEW NAME
IF CLASS IS DATA BASE OR ORDER, THEN
EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS ENDIF
IF CLASS IS DATA BASE, THEN
EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION ENDIF
IF CLASS IS INTEGRITY TABLE, THEN
CALL EXEC TO WRITE/READ NEW NAME
BUILD AVA MANAGER REQUEST TO CHANGE NAME IN TOC
BUILD AVA MANAGER REQUEST TO STORE NEW NAME IN TABLE
CALL XREN TO ISSUE REQUESTS
EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR ELSE IF CLASS IS DATA BASE, THEN
BUILD AND ISSUE AVA MANAGER REQUEST FOR TOC ENTRY EXIT TO :NAMERR: IF RETURN INDICATES ERROR EXIT TO :INVALID: IF DATA BASE IS AN NDB ENDIF
BUILD AND ISSUE AVA MANAGER REQUEST FOR NAME CHANGE EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR IF CLASS IS DATA BASE OR ORDER, THEN CALL XORIN TO FORMAT FILE NAME
CALL FILE MANAGER TO CHANGE DISC FILE NAME
EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR IF FILE IS A PDB, THEN CALL XORDD TO DELETE OLD PDB FROM XPPB EXIT TO :NAMANG: IF ERROR RETURNED CALL XORDD TO ADD NEW PDB NAME TO XPPB EXIT TO :TELLER: IF ERROR RETURNED ENDIF
ENDIF
ENDIF
EXIT XDREN
:SYNTAX: CALL XRMSP TO DISPLAY SYNTAX ERROR AND EXIT :TOOLNG: CALL XRMSP ("NEW NAME IS TOO LONG") AND EXIT :CLASER: CALL XRMSP TO DISPLAY CLASS DESIGNATION ERROR AND EXIT :INVALID: CALL XRMSP ("AN NDB CANNOT BE RENAMED") AND EXIT
BEGIN XREQ
CALL XREQ TO PROCESS AVA REQUEST(S)
IF AN ERROR RETURNED BY AVA MANAGER, THEN
IF ERROR FLAG (ERFLG) IS ZERO, THEN
TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED
CALL XRMSSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
ENDIF
CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
IF CLASS OF ELEMENT IS DDE, THEN
CALL PURGE TO DELETE THE FILE
ELSE, ELEMENT RESIDES IN AVA
CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
ENDIF
IF AVA REQUESTS EXIST IN REQBUF BEYOND FAILING REQUEST, THEN
MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
ENDIF
ELSE
SET REQPTR TO 1 INDICATING NO REQUESTS PRESENT
ENDIF
EXIT XREQ
END XREQ
FORTRAN CALLING PROCEDURE

CALL XDRES (DATBUF)

XDRES PROCESSES THE RESTORE DIRECTIVE. THE SPECIFIED UTDB OR MDB
FILE IS OPENED AND ITS TOC IS READ. XDLSIS IS CALLED TO MARK
TOC ENTRIES FOR RESTORE. XDRADE IS CALLED TO RESTORE DATA
ELEMENTS AND TABLES. XBRDF IS CALL TO RESTORE DRE'S.

INPUT
COMMON XD - LU, FLGS, REOPTR, REBUF, TOKENS,
CARTAGS, COMPTR, CONBUF

CALLING SEQUENCE

DATBUF - 1408 WORD BUFFER USED TO READ UTDB FILE

OUTPUT
COMMON XD - REBUF

COMMON XB - ABFLG = ABORT FLAG. NON-ZERO VALUE FROM XDRE,
XDREF, XDLSIS OR XDRENG INDICATES ABORT
OF RESTORE
ALLFLG = SET NON-ZERO IF LIST OF ELEMENTS SPECIFIED
DEBUG = DEBUG BIT OF 'FLGS' IN XD COMMON
ENDBLK = BLOCK # OF LAST BLOCK READ
ERFLG = SET NON-ZERO IF MESSAGE #54 ISSUED SO
THAT IT IS ISSUED ONLY ONCE
FILNAM = UTDB/MDB FILE NAME
TOCBUF = UTDB TOC ENTRIES, 8 WORDS EACH, MAX 1200
WORDS
TOCENT = AWA TOC ENTRY FOR THE UTDB
TOCDNX = INDEX TO NEXT UTDB TOC ENTRY
TOTSIZ = NO. OF BLOCKS REMAINING IN UTDB FILE

NOTES

CLOSE
EXEC
OPEN
READF
XDLSIS
XDRADE
XDRENG
XDRES
XRDE

Routines Called

CLOSE
EXEC
OPEN
READF
XDLSIS
XDRADE
XDRENG
XDRES
XRDE
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1476 1 BEGIN XRES
1477 2 SET ABFLG TO ZERO
1478 3 ERREXIT IF NEXT TOKEN IS NOT A COMMA :ERR09:
1479 4 ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERR09:
1480 5 BTAHE THIS NAME AS DATA BASE TO BE RESTORED
1481 6 INCREMENT TO NEXT TOKEN
1482 7 IF TOKEN IS NOT A COMMA, THEN
1483 8 ERREXIT IF TOKEN IS NOT EOS :ERR04:
1484 9 ENDEF
1485 10 BLDW AVA REQUEST FOR TOL ENTRY RETRIEVE
1486 11 CALL XRES TO PROCESS AVA REQUEST
1487 12 ERREXIT IF AVA 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 XRMTH TO CONSTRUCT QUALIFIED FILE NAME
1491 16 ENDDIF
1492 17 CALL OPEN TO OPEN SPECIFIED FILE
1493 18 ERREXIT IF OPEN FAILED :ERR08:
1494 19 CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1495 20 ERREXIT IF READ FAILED :ERR08:
1496 21 INITIALIZE EDBLK TO NUMBER OF TOL BLOCKS
1497 22 IF TOL IS MORE THAN 1 BLOCK LONG, THEN
1498 23 CALL READ TO READ REMAINING TOL ENTRIES INTO TOCBUF
1499 24 ERREXIT IF READ FAILED :ERR08:
1500 2 ENDIF
1501 2 UPDATE TOSIZ TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY EDBLK)
1502 3 CLEAR ERROR MESSAGE FLAG (ERFLG)
1503 4 IF TOKEN IS EOS (I.E., NO LIST OF ELEMENTS), THEN
1504 5 SET ALLFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 6 ELSE
1506 7 SET ALLFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 8 CALL XDLIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 9 EXIT XRES IF ABFLG SET BY XDLIS
1509 10 ENDDIF
1510 11 CALL XROE TO RESTORE AVA RESIDENT ELEMENTS
1511 2 1 EXIT XRES IF ABFLG SET BY XRDE
1512 2 CALL XRDF TO RESTORE XRDE'S
1513 1 EXIT XRES IF ABFLG SET BY XRDF
1514 2 CALL CLOSE TO CLOSE DATA BASE FILE
1515 2 ERREXIT IF CLOSE FAILED :ERR08:
1516 2 DO WHILE AVA REQUESTS REMAIN IN RERBUF
1517 3 CALL XRDE TO PROCESS AVA REQUESTS
1518 2 EXIT XRES IF ABFLG SET BY XRDE
1519 2 ENDDO
1520 1 EXIT XRES
1521 2 :ERR04: CALL XRMTH - "SYNTAX ERROR, EXTRAMEOS DATA"
1522 2 :ERR08: CALL XRMTH - "FILE MANAGER ERROR ............."
1523 2 :ERR09: CALL XRMTH - "SYNTAX ERROR, MISSING OR ILLEGAL FIELD"
1524 2 :ERR10: CALL XRMTH - "........ NOT FOUND"
1525 2 :ERR33: CALL XRMTH - "CAN NOT RESTORE A PDB"
1526 2 DO UNTIL ALL AVA REQUESTS IN RERBUF 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
1 C*********
2 C0090 FORTRAN CALLING PROCEDURE
3 C0091 CALL XDSAV
4 C0092 C*********
5 C0093 C0096 INPUT
6 C0097 C0098 COMMON XE - CARTAG, COMBUF, COMPTA, FLAGS, L1, TOKENS
7 C0099 C0092 FILES - **XXXQ (UTDB FILE SPECIFIED)
8 C0093 C0093 OUTPUT
9 C0094 C0095 COMMON XE - REGBUF, RERPTA
10 C0096 C0093 FILES - **XXXQ (PDB FILE SPECIFIED)
11 C0097 C0095 INTERNAL VARIABLES
12 C0098 C0098 DCBPDB - DCB FOR THE PDB FILE; ALLOCATED IN JS COMMON
13 C0099 C0098 DEBUPO - DCB FOR THE UTDB FILE; ALLOCATED IN XB COMMON;
14 C0096 C0098 CONTAINS 152 WORD BUFFER USED TO READ THE
15 C0097 C0098 UTDB AND TO WRITE THE PDB.
1572 1  BEGIN YDSAV
1573 2  SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
1574 2  ERROR: IF "." IS NOT NEXT TOKEN: ERROR:
1575 2  INCREMENT TO NEXT TOKEN
1576 2  ERREXIT IF TOKEN IS NOT 'NAME': ERROR:
1577 2  ERREXIT IF THIS NAME IS MORE THAN 8 CHARACTERS: ERROR6:
1578 2  ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION: ERROR6:
1579 2  RETAIN THIS NAME AS UTDB
1580 2  INCREMENT TO NEXT TOKEN
1581 2  ERREXIT IF "." IS NOT NEXT TOKEN: ERROR:
1582 2  INCREMENT TO NEXT TOKEN
1583 2  ERREXIT IF TOKEN IS NOT 'NAME': ERROR:
1584 2  ERREXIT IF THIS NAME IS MORE THAN 8 CHARACTERS: ERROR6:
1585 2  ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION: ERROR6:
1586 2  RETURN THIS NAME AS PDB
1587 2  INCREMENT TO NEXT TOKEN
1588 2  ERREXIT IF TOKEN IS NOT EOS: ERROR4:
1589 2  BUILD AWK REQUEST TO RETRIEVE UTDB'S TOC ENTRY
1590 2  CALL XHEN TO PROCESS AWK REQUESTS
1591 2  ERREXIT IF TOC RETRIEVE FAILED: ERROR6:
1592 2  ERREXIT IF TOC ENTRY DOES NOT INDICATE UTDB: ERROR6:
1593 2  CALL XODSA TO ADD PDB TO XPO
1594 2  ERREXIT IF PDB ERROR: ERROR:
1595 2  ERREXIT IF PDB LIMIT EXCEEDED: ERROR2:
1596 2  ERREXIT IF PDB OPEN FAILED: ERROR1:
1597 2  ERREXIT IF OPEN FAILED: ERROR1:
1598 2  SET STATUS FLAG INDICATING PDB LOGGED
1599 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1600 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1601 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1602 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1603 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1604 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1605 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1606 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1607 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1608 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1609 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1610 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1611 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1612 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1613 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1614 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1615 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1616 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1617 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1618 2  ERREXIT IF PUT COMMAND FAILED: ERROR1:
1619 1  EXIT XDSAV
1620 2  :ERROR9: ISSUE MESSAGE "SYNTAX ERROR -- ILLEGAL OR MISSING FIELD"
1621 2  :ERROR6: ISSUE MESSAGE "INVALID UTDB FILE NAME ...."
1622 2  :ERROR16: ISSUE MESSAGE "INVALID PDB FILE NAME ...."
1623 2  :ERROR4: ISSUE MESSAGE "SYNTAX ERROR. EXTRANEOUS DATA"
1624 2  :ERROR17: ISSUE MESSAGE "PDB ..... ALREADY EXISTS"
1625 2 :ERR18: ISSUE MESSAGE "PRO FILE ACCESS ERROR ... "
1626 2 :ERR00: ISSUE MESSAGE "FILE MANAGER ERROR ... "
1627 2 :ERR20: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PRO'S ALREADY REACHED"
1628 2 :ERR21: ISSUE MESSAGE "ANA OVERFLOW. NOT LOGGED"
1629 2 RETURN:
1630 2 IF STATUS FLAG INDICATES PRO FILE IS OPEN, THEN
1631 3 PURGE PRO FILE
1632 3 ENDIF
1633 2 IF FLAG INDICATES PRO IS IN ANA, THEN
1634 3 CALL XDR TO DELETE PRO FROM ANA
1635 2 ENDIF
1636 2 IF FLAG INDICATES UTOB IS OPEN, THEN
1637 3 CLOSE UTOB
1638 2 ENDIF
1639 2 IF FLAG INDICATES PRO IS IN XPDB, THEN
1640 3 CALL XDR TO DELETE PRO FROM XPDB
1641 2 ENDIF
1642 1 END XDSAV
1643 1 XDSAV
164: 1 CRO  CALL XSTA (LU)
168: 1 CTO CALL XSTA (LU)
164: 1 CTO  CALL XSTA (LU)
165: 1 CTO  OUTPUT ANA AND DMA USAGE DATA AND STATISTICS ON ANA COMPACT
165: 1 CTO  PHASES
165: 1 CTO  INPUT
165: 1 CTO  LU   = LOGICAL UNIT OF OUTPUT DEVICE
165: 1 CTO  XSB COMMON
165: 1 CTO  KZ - TOTAL SIZE OF ANA
165: 1 CTO  FRE - AMOUNT OF FREE SPACE IN ANA
165: 1 CTO  DSR - TOTAL SIZE OF DMA (IN 64 WORD SECTORS)
165: 1 CTO  DFR - AMOUNT OF FREE SECTORS IN DMA
165: 1 CTO  NP - ARRAY OF THREE COMPACTION PHASE COUNTS
165: 1 CTO  OUTPUT
165: 1 CTO  USAGE AND STATISTICS TO LU
165: 1 CTO  NOTES
165: 1 CTO  USES EXEC, XREG
165: 1 CTO  PONES
165: 1 CTO  BEGIN XSTA
167: 2 FORMAT AND PRINT ANA USAGE DATA
167: 2 FORMAT AND PRINT DMA USAGE DATA
168: 2 FORMAT AND PRINT COMPACT STATISTICS
168: 1 END XSTA
00000000000000000000000000000000000000000000000000000000000

---

1683 1 C0************
1684 1 C00 CALL XDSTO (DATBUF)
1685 1 C00
1686 1 C00
1687 1 C00
1688 1 C00
1689 1 C00
1690 1 C00 XDSTO IS THE STORE DIRECTIVE HANDLER. IT VERIFIES INPUTS ON DIRECTIVE, BUILDS UTDB TOC, CREATES UTDB AND STORES UTDB TOC
1691 1 C00 ENTRY IN AWA.
1692 1 C00
1693 1 C00
1694 1 C00
1695 1 C00
1696 1 C00
1697 1 C00
1698 1 C00
1699 1 C00
1700 1 C00
1701 1 C04
1702 1 C04
1703 1 C04
1704 1 C04
1705 1 C04
1706 1 C04
1707 1 C04
1708 1 C04
1709 1 C04
1710 1 C04
1711 1 C04
1712 1 C04
1713 1 C04
1714 1 C04
1715 1 C04
1716 1 C05
1717 1 C05
1718 1 C05
1719 1 C05
1720 1 C05
1721 1 C05
1722 1 C05
1723 1 C05
1724 1 C05
1725 1 C05
1726 1 C05
1727 1 C05
1728 1 C05
1729 1 C05
1730 1 C05
1731 1 C05
1732 1 C05
1733 1 C05
1734 1 C05
1735 1 C05

---

INTERNAL XD COMMON USED:

XG(151) ABFLE - (INTEGER, 1 WORD) ABORT FLAG
XG(152) ERFLE - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XG(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER TO BE DISPLAYED
XG(157) TOTOZ (INTEGER, 1 WORD) TOTAL SIZE OF UTDB FILE
XG(158) TOTWRD - (INTEGER, 1 WORD) TOTAL WORDS IN A UTDB FILE
XG(159) FILE - (INTEGER, 3 WORDS) UTDB FILE NAME (*XXX)
XG(162) DATREC - (INTEGER, 1 WORD) RECORD # WHERE DATA GOES NEXT
XG(164) UDBERR - (INTEGER, 1 WORD) UTDB FILE ERROR FLAG
XG(166) UDBNAM - (INTEGER, 1 WORD) UTDB NAME (XXX)
XG(171) NTOC - (INTEGER, 1 WORD) NUMBER OF TOC ENTRIES
XG(201) TOBUF - (INTEGER, 1200 WORDS) UTDB TOC BUFFER
1737  1 BEGIN XSTO
1738  2 :ERR1:
1739  2 ABFLG TO ZERO (ABORT FLAG)
1740  2 ERREXIT IF UTDB NAME IS NOT VALID TO :ERR2:
1741  2 BUILD REQUEST FOR AWA TOC
1742  2 CALL XREQ TO MAKE MANAGER REQUEST
1743  2 CALL XEC TO GET AWA TOC
1744  2 ERREXIT IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1745  2 DO FOR ALL DATA BASE CLASS ENTRIES
1746  2 ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1747  2 ENDO
1748  2 END
1749  2 SET ERFLG OFF (ERROR MESSAGE FLAG)
1750  2 ERREXIT IF ABFLG IS NOT ZERO TO :ERR5:
1751  2 ELSE
1752  3 CALL XLIST TO PROCESS LIST TO BE STORED
1753  3 ERREXIT IF ABFLG IS NOT ZERO TO :ERR5:
1754  3 ENDF
1755  4 ERREXIT IF THERE ARE NO UTDB TOC ENTRIES (TOTOC=0) TO :ERR2:
1756  4 COMPUTE DAIREC AS FIRST RECORD AVAILABLE FOR DATA
1757  4 CALL XP' 'TO CREATE FILE NAME
1758  4 CALL TO CREATE UTDB FOR TOTSZ
1759  4 ERREXIT ERROR IN CREATE TO :ERR3:
1760  4 CALL XWRITE TO WRITE UTDB FILE
1761  4 ERREXIT IF ABFLG IS 4 (ORDE LARGER THAN SPECIFIED) TO :ERR1:
1762  4 ERREXIT IF ABFLG IS 3 (ORDE FILE ERROR) TO :ERR4:
1763  4 ERREXIT IF ABFLG IS 2 (UTOB FILE ERROR) TO :ERR3:
1764  4 CALL WRITE TO WRITE TOC RECORDS AT RECORD 1
1765  4 ERREXIT IF ERROR IN WRITE TO :ERR3:
1766  4 CALL CLOSE TO CLOSE UTDB FILE
1767  4 ERREXIT IF ERROR IN CLOSE TO :ERR3:
1768  4 BUILD REQUEST TO ALLOCATE UTDB IN AWA
1769  4 CALL XREQ TO MAKE REQUEST
1770  4 ERREXIT IF AWA OVERFLOW TO :ERR1:
1771  4 EXIT XSTO
1772  2 :ERR1:
1773  2 CALL XMSG TO DISPLAY MSGNO
1774  2 GO TO :ERR4:
1775  2 :ERR2:
1776  2 EXIT XSTO
1777  2 XSTO
1778  2 XSTO
1779  2 XSTO
1780  2 XSTO
1781  2 XSTO
1782  2 XSTO
1783  2 XSTO
1784  2 XSTO
1785  2 XSTO
1786  2 XSTO
1787  2 XSTO
1788  2 XSTO
1789  1 XSTO
1794 2 CALL XRMNG TO DISPLAY MSGNO
1795 2 GO TO :ERR5;
1796 2 :ERR3:
1797 2 CALL XRMNG 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 XRMNG TO DISPLAY STORE ABORTED/ UTDB NOT CREATED MESSAGE
1803 1 END XDSTO
FORTRAN CALLING PROCEDURE

CALL XDOTC

*********

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

*********

INPUT

**COMBUF, COMPTR, FLAGS, LU**

**ANAB - AWS HEADER AND TOC (SEE MODULE XMWA)**

**FILES - DATABASE FILES AS APPROPRIATE**

OUTPUT

**COMBUF, RERBUF, RERPTR**

**PRINTED OUTPUT - AWS OR UDB TOC DISPLAY**

*********

LOCAL VARIABLES

**ASZ - TOTAL ALLOCATABLE SIZE OF AWS (SEE MODULE XMWA)**

**ANA - FIRST PORTION OF AWS (HEADER AND TOC)**

**CLASS - ARRAY OF ALLOCATABLE CLASS DESIGNATORS FOR TOC DIRECTIVE**

**CLS - CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE**

**CODE - CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'**

**EOC - END OF TOC CHAIN DESIGNATOR (~32767)**

**FRE - CURRENT AMOUNT OF FREE SPACE IN AWS (SEE MODULE XMWA)**

**NO - ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)**

**IDIM - TOC ENTRY DIM FIELD (WORD 8)**

**LINE - 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 AWS (SEE MODULE XMWA)**

**POS - VALUE OF 2 OR 22 INDICATING FIRST OR SECOND POSITION IN TOC DISPLAY LINE**

**TCMDIM - 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, IAND, IXOR, KCVT, OPEN, READF, XDSTA, XREG, XREG, XRMOV, XRSG, XRSG, XRSG**

**IN THE EVENT THE COMPACTED AWS 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 DATABASE FILE MAY NOT HAVE A TOC TOO**

**LARGE FOR THE XDOTC INTERNAL BUFFER.**
1 BECUM YDTC
2 SL) FOR OUTPUT OF ALL USER CLASSES FROM AWA TO TERMINAL
3 IF NEXT TOKEN IS NOT EOS
4 THEN
5 IF TOKEN IS A HYPHEN
6 THEN
7 INCREMENT TO NEXT TOKEN
8 IF TOKEN IS NOT A CHARACTER T
9 THEN
10 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
11 SET OUTPUT UNIT FOR LINE PRINTER
12 ENDIF
13 INCREMENT TO NEXT TOKEN
14 ENDIF
15 IF TOKEN IS NOT EOS
16 THEN
17 EXIT TO :ERROR: IF TOKEN IS NOT A COMA
18 INCREMENT TO NEXT TOKEN
19 IF TOKEN IS NOT A COMA
20 THEN
21 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (D, F, I, S OR B)
22 SET OPTION FOR INDICATED CLASS
23 INCREMENT TO NEXT TOKEN
24 ENDIF
25 IF TOKEN IS NOT EOS
26 THEN
27 EXIT TO :ERROR: IF TOKEN IS NOT A COMA
28 INCREMENT TO NEXT TOKEN
29 EXIT TO :ERROR: IF TOKEN IS NOT A FOUR CHARACTER NAME
30 INCREMENT TO NEXT TOKEN
31 EXIT TO :ERROR: IF TOKEN IS NOT EOS
32 CALL XRQ TO RETRIEVE DATA BASE FILE TOC ENTRY
33 EXIT TO :ERROR: IF NOT LOGGED IN TOC
34 GET TYPE AND SET NAME IN HEADER
35 CONSTRUCT FILE NAME
36 READ FIRST DATA BASE FILE TOC RECORD
37 INITIALIZE TOC HEAD TO APPPEAR SIMILAR TO AWA TOC
38 READ SUBSEQUENT TOC RECORDS
39 EXIT TO :ERROR: IF FILE ACCESS FAILS
40 DO FOR EACH NON-EMPTY TOC CHAIN
41 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
42 IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NUL CHAIN EXISTED
43 THEN
44 MARK PREVIOUS ENTRY AS AN END OF CHAIN
45 ENDIF
46 EMDDO
47 DO FOR EACH ENTRY IN TOC
48 IF NOT MARKED AS AN END OF CHAIN
49 THEN
50 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
51 ENDIF
52 EMDDO
53 EXCLUDE CHAIN B (DATA BASE FILES) FROM DISPLAY
54 ENDIF
55 ENDIF
56 IF REFERENCING AWA TOC
57 THEN
58 CALL XRQ TO RETRIEVE AWA TOC
59 ENDIF
IF DEBUG AND/OR TRACE FLAGS ARE SET
   THEN
   EXTEND OPTION TO INCLUDE SYSTEM CLASSES (0, 1, 5 AND 7)
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 "OINTS 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
   ELSE
      PRINT "DATA LOST" MESSAGE
      EXIT PROCESSING FOR THIS CHAIN
   ENDIF
   PRINT ENTRY
ENDIF
ENDIF
ENDIF
ENDIF
IF PROCESSING AWX TOC
THEN
   CALL XSTA TO DISPLAY AWX USAGE STATISTICS
ENDIF
1 EXiT XDTOC
ERROR: EXIT WITH INVALID OUTPUT DEVICE ID
ERROR: EXIT WITH SYNTAX ERROR
ERROR: EXIT WITH INVALID CLASS DESIGNATOR
ERROR: EXIT WITH INVALID UTDB FILE NAME
ERROR: EXIT WITH UTDB FILE ACCESS ERROR
1 END XDTOC
1 BEGIN XDMRT
2 SET WRDNO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
3 SET MORE = 0 (NUMBER OF AWS REQUESTS IN REBUFF)
4 DO FOR ALL UTDB TOC ENTRIES UNTIL CLASS IS DRDE
5 BUILD REQUEST FOR DATA FROM AWS
6 INCREMENT MORE BY 1
7 IF REQUEST BUFFER IS FULL (MORE=8) THEN
8 PERFORM READTO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
9 ENDF
10 IF THERE ARE REMAINING REQUESTS (MORE>0) THEN
11 SET NEXT REQUEST TO SAY END OF REQUEST LIST
12 PERFORM READTO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
13 IF THERE IS A PARTIAL DATA RECORD LEFT (WRDNO>1) THEN
14 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
15 ENDF
16 DO FOR EACH DRDE UTDB TOC ENTRY
17 STORE DATREC IN UTDB TOC ENTRY
18 CALL BGNFM TO CREATE FILE NAME
19 IF DRDE FILE IS TYPE 3 THEN
20 CALL OPEN TO OPEN FILE AS CORRECT TYPE
21 ERREXIT IF OPEN ERROR TO :ERR1:
22 DJ UNTIL EOF IS READ
23 CALL READ TO READ 1 RECORD
24 ERREXIT IF READING ERROR TO :ERR1:
25 STORE RECORD LENGTH AT FRONT AND REAR OF DATA
26 INCREMENT WRDNO BY LENGTH + 2
27 IF THERE IS ENOUGH DATA TO WRITE (WRDNO>128) THEN
28 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
29 ENDF
30 IF THERE IS REMAINING DATA (WRDNO>1) THEN
31 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
32 ENDF
33 SET DATREC TO NEXT AVAILABLE RECORD FOR DATA
34 ELSE
35 CALL OPEN TO OPEN FILE AS TYPE 1
36 ERREXIT IF OPEN ERROR TO :ERR1:
37 COMPUTE TOTAL SIZE OF FILE IN WORDS
38 DO UNTIL ALL DATA IS COPIED TO UTDB (SIZE=0)
39 IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
40 SET LENGTH = SIZE
41 ENDF
42 CALL READ TO READ LENGTH DATA
43 ERREXIT IF READING ERROR TO :ERR1:
44 CALL WRITF TO WRITE LENGTH DATA
45 ERREXIT IF WRITE ERROR TO :ERR1:
46 INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
47 DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
48 ENDF
49 ENDF
50 CALL CLOSE TO CLOSE DRDE FILE
51 ERREXIT IF CLOSE ERROR TO :ERR1:
52 ENDF
53 EXIT XDMRT
BEGIN READ
2090  5 CALL XREN TO MAKE REQUEST
2091  5 ERRNE IF THERE IS AN OVRFLOW TO :ERR4:
2092  5 DO FOR NUMBER OF REQUESTS (MOREB) IN BUFFER
2094  4 CALL EXEC TO GET DATA FROM SAN
2094  4 GET DATAB R/WNO INTO THIS UTDB TOC ENTRY
2094  4 INCREMENT R/WNO BY SIZE OF THIS ELEMENT
2097  4 IF THERE IS ENOUGH DATA TO WRITE (R/WNO>128) THEN
2096  5 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
2092  4 ENDIF
2100  3 ENDDO
2101  3 REINITIALIZE REQUEST BUFFER (MOREB=0)
2102  2 END READ
2104  2 *
2106  2 *
2107  3 BEGIN WRITE
2108  3 COMPUTE LENGTH TO WRITE IN WORDS AND RECORDS
2109  3 IF THERE IS NOT AT LEAST 1 RECORD TO WRITE THEN
2110  3 SET LENGTH TO MINIMUM OF 1 RECORD
2110  3 ENDIF
2111  3 IF THIS IS A DRE TYPE 3 FILE THEN
2112  4 ERRNE IF THERE ARE MORE BLOCKS THAN SPECIFIED TO :ERR2:
2113  4 ENDIF
2114  3 CALL WRITF TO WRITE LENGTH DATA TO UTDB FILE
2115  3 ERRNE IF WRITF ERROR TO :ERR3:
2116  3 DECREMENT R/WNO BY LENGTH
2117  3 INCREMENT DATAB R/WNO BY LENGTH/128
2118  3 IF THERE IS REMAINING DATA (R/WNO>1) THEN
2119  4 CALL XRMOV TO MOVE REMAINING DATA UP IN BUFFER
2120  3 ENDIF
2121  2 END WRITE
2122  2 :ERR1:
2123  2 CALL XRMSG WITH DRE NAME AND RC
2124  2 SET ABFLG TO SAY A DRE ERROR
2125  2 GO TO :ERR3:
2126  2 :ERR2:
2127  2 CALL XRMSG TO SAY DRE FILE LARGER THEN SPECIFIED IN TOC
2128  2 SET ABFLG TO SAY DRE FILE ERROR
2129  2 :ERR3:
2130  2 CALL CLOSE TO CLOSE DRE FILE
2131  2 SET ABFLG TO SAY UTDB FILE ERROR
2132  1 EXIT XDMRT
2133  2 :ERRA:
2134  2 SET ABFLG TO SAY AWF OVRFLOW
2135  2 DO FOR REMAINING REQUESTS
2136  2 CALL EXEC TO FREE CLASS NUMBER
2137  2 ENDDO
2138  1 END XDMRT
FORTRAN CALLING PROCEDURE

CALL XELDS (THREE WORD ARRAY CONTAINING 'XECAL')

FDX 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 XEIND, XEINI, XEIMS, XEINX, XERTN

BEGIN XECAL

CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SEREDT:, :INTEDT:) SUBSTA (-1 TO 3)

:GLOBAL: CALL XEINI TO INITIALIZE GLOBAL COMMON

:DIRECT: CALL XEIND TO INITIALIZE DYNAMIC COMMON OR DIRECTIVES

:EXECUT: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL

:SEREDT: CALL XEIMS TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING

:INTEDT: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING

END CASE

END XECAL
39  1 CDO      FORTRAN CALLING PROCEDURE
40  1 CDO
41  1 CDO      CALL XEIND
42  1 CDO
43  1 C------
44  1 C------
45  1 C01     INITIALIZE XB COMMON FOR DIRECTIVE LEVEL
46  1 C01
47  1 C------
48  1 C02     INPUT
49  1 C02     COMMON XE - CARTRG, FLAGS
50  1 C02
51  1 C02
52  1 C------
53  1 C03     OUTPUT
54  1 C03     COMMON XB - DIRECT, NUMDIR
55  1 C03
56  1 C------
57  1 C05     NOTES
58  1 C05     USES CLOSE, EXEC, OPEN, PRTR, READ, XREXT, XRIG, XRMSG, XVABN
59  1 C05
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
1 BEGIN XEIME
2 INITIALIZE COMMON TO ZEROS
3 SET FILE (ART-LOC) NUMBER
4 SET STATE TO DIRECTIVE LEVEL
5 SET TKMLNG
6 INITIALIZE TOKENS
7 READ LIBRARY DIRECTARY FIRST RECORD
8 EXIT TO :ERROR1: IF READ FAILED
9 STORE NUMBER OF PROCESSORS IN NOPROC
10 READ PROCESSOR NAME RECORD
11 EXIT TO :ERROR1: IF READ FAILED
12 CALL XREN TO ALLOCATE AND STORE PROCESSOR DIRECTARY IN AMA
13 EXIT TO :ERROR1: IF REQUEST FAILED
14 DO FOR EACH ID SEGMENT IN SYSTEM
15 DO FOR EACH PROCESSOR IN LIBRARY DIRECTARY
16 IF NAMES ARE THE SAME
17 THEN
18 NOTE EXISTANCE OF ID
19 ENDIF
20 ENDDO
21 IF ANY PROCESSORS NOT MARKED
22 THEN
23 LIST PROCESSORS MISSING ID SEGMENTS
24 EXIT XREN TO INITIALIZATION FAILURE
25 ENDF
26 CALL OPEN TO OPEN PDB/MDB DIRECTORY ( XPDB)
27 EXIT TO :ERROR1: IF OPEN FAILED
28 CALL READ TO READ 1ST RECORD OF XPDB (LIST OF MDB'S)
29 EXIT TO :ERROR1: IF READ FAILED
30 PERFORM DLOG TO LOG MDB'S IN AMA
31 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER
32 CALL READ TO READ THAT RECORD OF XPDB
33 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)
34 CALL CLOSE TO CLOSE XPDB
35 PERFORM DLOG TO LOG MDB'S IN AMA
36 IF THERE ARE REQUESTS IN THE AMA REQUEST BUFFER, THEN
37 CALL XREN TO PROCESS THE AMA REQUESTS
38 ENDF
39 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
40 EXIT XEIME
41 BEGIN DLOG
42 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
43 BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
44 IF 8 AMA REQUESTS HAVE BEEN BUILT, THEN
45 CALL XREN TO PROCESS AMA REQUESTS
46 ENDIF
47 EXIT TO :ERROR1: IF A REQUEST FAILED
48 ENDDO
49 END DLOG
50 :ERROR1: LIBRARY INITIALIZATION ERROR TERMINATION
51 1 END XEIME
FORTRAN CALLING PROCEDURE

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********

CALL XEINI

*********
POSITION TO 1ST TOKEN AFTER 'IMTE'

EXIT IF TOKEN IS NOT ',' : ERR1:

INCREMENT TO NEXT TOKEN

EXIT IF TOKEN IS NOT A NAME : ERR2:

SET PRCNAM TO THIS NAME

INCREMENT TO NEXT TOKEN

SET INTNAM TO O

SET NEWTAB TO 'EXIT'

IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN

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 INTNAM TO THE NAME

INCREMENT TO NEXT TOKEN

ENDIF

IF TOKEN IS NOT EOS, THEN

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

ENDIF

EXIT IF TOKEN IS NOT EOS : ERR2:

ENDIF

ELSE

INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER

NEWTAB IS SET TO 'EXIT'

ISSUE MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED

ENDIF

MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE

EXIT IF REQUEST IS UNSUCCESSFUL : ERR11:

START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR

EXIT IF PROCESSOR PRCNAM IS FOUND

ELSE

INCREMENT TO NEXT ENTRY

ENDLOOP

EXIT : ERR8:

ENDSEARCH

SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND

IF INTNAM = O, THEN

CONSTRUCT THE NAME OF THE DEFAULT INTERFACE TABLE AS "PRCNAM"

OPEN THE DEFAULT INTERFACE TABLE FILE

EXIT IF OPEN FAILED : ERR12:

READ THE DEFAULT INTERFACE TABLE

EXIT IF THE READ FAILED : ERR16:

READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM

OF WKBUF

EXIT IF READ FAILED : ERR14:

ELSE

MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE

EXIT IF ERROR OR COULD NOT FIND : ERR:

ENDIF

EXIT IF VERS .ME. VERSION NO. OF TABLE TO BE EDITED : ERR10:

CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"

OPEN THE PROMPT TABLE

EXIT IF OPEN FAILED : ERR15:
279 2 READ 2ND RECORD OF THIS PROMPT TABLE
280 2 ERREXIT IF READ FAILED :ERR17:
281 2 MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUFF TO AREA FOLLOWING
282 2 THE PROMPT TABLE
283 2 THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
284 2 LITERAL DISPLAY THAT MATCH EACH LITERAL DATA ENTRY
285 2 THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
286 2 MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
287 2 AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
288 1 EXIT XEINI
289 2 :ERR2: "SYNTAX ERROR ... "
290 2 :ERR4: "INTNAM" NOT FOUND
291 2 :ERR6: INVALID NAME FIELD
292 2 :ERR8: "PRCNAM" NOT IN DIRECTORY
293 2 :ERR10: VERSION OF INTNAM DOES MATCH CURRENT VERSION
294 2 :ERR11: XEINI OP 1 INITIALIZATION ERROR
295 2 :ERR12: XEINI OP 2 INITIALIZATION ERROR
296 2 :ERR13: XEINI OP 3 INITIALIZATION ERROR
297 2 :ERR14: XEINI OP 4 INITIALIZATION ERROR
298 2 :ERR15: XEINI OP 5 INITIALIZATION ERROR
299 2 :ERR16: XEINI OP 6 INITIALIZATION ERROR
300 2 :ERR17: XEINI OP 7 INITIALIZATION ERROR
301 2 SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
302 1 END XEINI
BEGIN XEMS
BUILD AWAY REQUEST TO RETRIEVE "XLIBD"
ERREXIT IF TOKEN IS NOT COMMA :ERROR2:
INCREMENT TO NEXT TOKEN
IF TOKEN IS NAME, THEN
    BUILD AWAY REQUEST TO RETRIEVE OLDTAB
    RETAIN THIS NAME AS OLDTAB
    INCREMENT TO NEXT TOKEN
ELSE
    SET OLDTAB TO ZERO
ENDIF
ERREXIT IF TOKEN IS NOT COMMA :ERROR2:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT NAME :ERROR2:
REPLACEMENT NAME AS NEWTAB
IF NEWTAB NOT EQUAL TO OLDTAB, THEN
    BUILD AWAY REQUEST TO VERIFY EXISTENCE OF NEWTAB
ENDIF
CALL XREX TO PROCESS AWAY REQUEST(S)
IF NEWTAB NOT EQUAL TO OLDTAB, THEN
    ERREXIT IF NO ERROR FROM XREX :ERROR12:
ENDIF
IF OLDTAB NOT ZERO, THEN
    ERREXIT IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED :ERROR4:
SET PROMPT MODE AS UPDATE
SET NO. ENTRIES AS OLDTAB SIZE / 7
READ OLDTAB INTO WORKING BUFFER
ERREXIT IF FIRST REQUEST (RETRIEVE "XLIBD) FAILED :ERROR1:
READ "XLIBD" INTO COMMON
ELSE
    SET PROMPT MODE AS CREATE
    SET NO. ENTRIES TO ZERO
ENDIF
OPEN, READ AND CLOSE FILE XSPRM
SET COUNT AND REREAD DIRECTIVES INTO XB COMMON
SET SUBSTITUTE FLAG TO SEQ. EDIT. (=2)
EXIT XEMS
:ERROR1: CALL XREXG - 'INITIALIZATION ERROR ....'
:ERROR2: CALL XREXG - 'SYNTAX ERROR'
:ERROR4: DEFAULT MESSAGE TO '....NOT FOUND'
IF ERROR WAS NO AWAY SPACE THEN
    SET MSG TO '....NO AWAY SPACE'
ENDIF
CALL XREXG TO DISPLAY MESSAGE
:ERROR12: CALL XREXG - '.... ALREADY EXISTS'
END XEMS
FORTRAN CALLING PROCEDEURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XB - LITPTR, NUMARG, WKBLNG, WKBUF

NOTE: WKBUF IS INPUT WITH THE INTERFACE TABLE'S CHARACTERISTICS IN THE TOP AND THE "PACKED" LITERAL AREA IN THE BOTTOM.

COMMON XB - LITPTR, LITDNW, MARL, WHDF

NOTE: WKBUF IS OUTPUT WITH THE LITERAL AREAS IN THEIR "UNPACKED" FORM.

USES Routines

XIEXT
XRMOF
XRMKB
XRMSC
XRESET
1. **BEGIN**

   - **ENTRY**: ALL LITERAL ENTRIES HAVE BEEN PROCESSED
   - **START SEARCH**: UNLESS ALL ARGUMENTS SEARCHED
   - **EXIT**: IF BIT FIELD FOR ARGUMENT = 0000

2. **BUILD**: LITERAL ENTRY (3, 1-16, 1-12)

3. **EXECUTE**: BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA

4. **END**

5. **EXIT**: 

   - **RETURN**

**END SEARCH**

**END**

---

**REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR**
`FORTRAN CALLING PROCEDURE`

1 CD0  CALL XEIXX
1 CD1  CALL XEIXX
1 CD2  CALL XEIXX
1 CD3  CALL XEIXX
1 CD4  CALL XEIXX
1 CD5  CALL XEIXX
1 CD6  CALL XEIXX

**INITIALIZE XE AND XB COMMON FOR EXECUTION CONTROLLER**

1 CD1  COMMON XE - COMBUF, COMPTR, FLAGS, LU, MASSTA, NOPROC, TOKENS
1 CD2  AVA - SEQUENCE TABLE, LIBRARY DIRECTORY NAME TABLE
1 CD3  COMMON XE - COMBUF, RMBUF, RMBPTR, SERROR, SERNAME, SERPTR
1 CD4  SERROR, SUBSTA
1 CD5  COMMON XE - LIBD, NOPRC2, SEOLNG, WKBLNG, WKBUF

**OUTPUT**

1 CD6  COMMON XE - COMBUF, RMBUF, RMBPTR, SERROR, SERNAME, SERPTR
1 CD7  SERROR, SUBSTA

**NOTES**

1 CD8  USES EXEC, PRM, XREG, XREXT, XRI6, XRMV, XRMSG, XUDIG, XVABN
1 CD9  USES EXEC, PRM, XREG, XREXT, XRI6, XRMV, XRMSG, XUDIG, XVABN
1  BEGIN XEINX
2  IF INITIALIZATION FROM DIRECTIVE
3     THEN
4  IF DIRECTIVE IS HANU
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 HYPHEN
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  ENDF I
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 SENSTA TO FIRST SEQUENCE NUMBER
26  SET SEGREE 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  ENDF I
38  IF TOKEN NOT END.
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 :ERRORS: 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  ENDF I
49  ENDF I
50  SET SLENTR TO SENSSTH
51  ENDF I
52  ENDF I
53  INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
54  EXIT TO :ERROR: IF INITIALIZATION FAILS
55  EXIT XEINX
56  IF INITIALIZATION FAILURE TERMINATION
57  EXIT TO :ERROR2: SET SUBSTA 1) DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
58  XEINX
551  2  :ERR03: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552  2  :ERR04: SET SUBSTA TO DIRECTIVE LEVEL
554  2  IF ERROR WAS NO AWA SPACE THEN
556  2  EXIT WITH NO AWA SPACE ERROR
555  2  ELSE
557  2  EXIT WITH SEQUENCE TABLE NOT FOUND ERROR
558  2  ENDIF
559  2  :ERR05: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
560  1  EXIT XEINX
FORTRAN CALLING PROCEDURE:
CALL XELS (SEGNM) TO CALL SEGMENT FROM MAIN
CALL XERTH TO RETURN TO MAIN PROGRAM

YELDS ALLOWS A MAIN PROGRAM TO "CALL" A SEGMENT AND
THE ENTRY POINT XERTH 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 XERTH
2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
3) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XERTH"
4) WHEN LOADER IS RUN, THE FOURTH PARAMETER MUST BE 1
   INDICATING TO LOAD MAIN PLUS SEGMENTS

BEGIN 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

RETURN TO MAIN PROGRAM
FORTNAM CALLING PROCEDURE

CALL XECM

* * * * * *

XECM IS CALLED BY XEXEC AFTER COMPLETION OF ALL AUTOMATIC AND
SEMIAUTOMATIC EXECUTIONS TO ASSURE THAT ALL DO AND SCAN PROCESSOR
CONTROL DATA IS PURGED FROM THE ANA AND ALL INCOMPLETE DATA BOX
FILES ARE PURGED FROM THE SYSTEM.

* * * * * *

INPUT

X COMMON - CARTRG
ANAC - DOSTK, SCNTB (SEE DO AND SCAN PROCESSORS)

* * * * * *

OUTPUT

X COMMON - REBUF, REPTR
X COMMON - SCRATCH

* * * * * *

EXTERNAL ROUTINES
CLOSE, EXEC, PURGE, XREQ, XRIG, XMOV, XRUNG

* * * * * *

* * * * * *

BEGIN XECM

BUILD REQUESTS TO DELETE DOSTK AND RETRIEVE SCNTB THEN DELETE IT
CALL XREQ TO ATTEMPT REQUESTS
ISSUE MESSAGE XE18 FOR SUCCESSFUL DELETES
IF XREQ COMPLETED REQUESTS, I.E., SCNTB EXISTED, THEN
DO FOR EACH SCAN CONTROL ENTRY IN SCNTB
CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN
CALL PURGE TO PURGE THE FILE
IF PURGE RETURNED AN ERROR, THEN
CALL XMSG TO DISPLAY WARNING MESSAGE
ENDIF
BUILD REQUEST TO DELETE DATA BOX FROM T0C
ENDDO
CALL XREQ TO DELETE DATA BOXES FROM ANA TOC

END XECM
685   1 BEGIN EXEC
686     2 RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASNO, QUAL & FLAGS
687     2 CALL XEINI TO INITIALIZE GLOBAL COMMON
688     2 DO FOREVER -- TERMINATES INSIDE HANDLER FOR OFF
689     3 CALL STCM FOR INPUT OF DIRECTIVE
690     3 IF ERROR OR NOT A VALID DIRECTIVE NAME
691     3 THEN
692 4 ISSUE MESSAGE ED6
693 3 ELSE
694 4 IF NAME IS INTE
695 4 THEN
696 5 SET STATES TO INTE LEVEL
697 5 CALL XEINI TO INITIALIZE DYNAMIC COMMON
698 4 EXIT TO :RESET; IF ERROR
699 5 CALL XINTE TO EDIT TABLE
700 4 ELSE
701 5 IF NAME IS SEER
702 5 THEN
703 6 SET STATES TO SEER LEVEL
704 6 CALL XEINS TO INITIALIZE DYNAMIC COMMON
705 6 EXIT TO :RESET; IF ERROR
706 6 CALL XSERE TO EDIT TABLE
707 5 ELSE
708 6 IF NAME IS FOR SOME EXECUTION CONTROL OPTION
709 6 THEN
710 7 SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
711 7 DO UNTIL MASTOA IS AT DIRECTIVE LEVEL
712 8 CALL XEINX TO INITIALIZE DYNAMIC COMMON
713 8 EXIT TO :RESET; IF ERROR
714 8 CALL XXCMT TO PERFORM EXECUTIONS
715 8 IF SUBSTA IS SET TO SEER LEVEL
716 8 THEN
717 9 CALL XEINS TO REINITIALIZE DYNAMIC COMMON
718 9 EXIT TO :RESET; IF ERROR
719 9 CALL XSERE TO SUPPORT EXECUTION CONTROL
720 8 ENDIF
721 8 IF SUBSTA IS SET TO INTE LEVEL
722 8 THEN
723 9 CALL XEINI TO REINITIALIZE DYNAMIC COMMON
724 9 EXIT TO :RESET; IF ERROR
725 9 CALL XINTE TO SUPPORT EXECUTION CONTROL
726 8 ENDIF
727 7 ENDIF
728 7 IF EXECUTION MODE WAS SEMI OR AUTO
729 7 THEN
730 8 CALL XECON TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES
731 8 ENDIF
732 6 ELSE
733 7 CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?
734 6 ENDIF
735 5 ENDIF
736 4 ENDIF
737 4 :RESET;
738 4 IF SUBSTA IS NOT DIRECTIVE LEVEL
739 4 THEN
740 5 CALL XEIND TO REINITIALIZE DYNAMIC COMMON
741 4 ENDIF
742 3 ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CDO FORTRAN CALLING PROCEDURE
2 CDO CALL XINTE
3 CDO C********
4 CDO OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR
5 CDO C********
6 CDO INPUT
7 CDO COMMON XE - 
8 CDO LU = USER'S LOGICAL UNIT NO.
9 CDO COMMON XB - 
10 CDO DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE 
11 CDO TABLE EDITOR ROUTINES
12 CDO C********
13 CDO NOTES
14 CDO USES ROUTINES
15 CDO XINIX
16 CDO XUDMG
17 CDO XERTN
18 CDO C********
19 CDO * XINTE IS THE INTERFACE ROUTINE FOR THE INTERFACE TABLE EDITOR
20 CDO WHEN CALLED IN THE FDS EXECUTIVE'S OVERLAY STRUCTURE.
21 CDO *
22 CDO * USER XINTE
23 CDO BEGIN XINTE
24 CDO CALL XINIX TO EXECUTE INTERFACE TABLE EDITOR
25 CDO CALL XERTN TO RETURN TO XEXEC IN MAIN SEGMENT
26 CDO END XINTE
FORTRAN CALLING PROCEDURE
CALL XINIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

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

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

OUTPUT

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

COMMON XB -
PRMTTN = CURRENT PROMPT MODE. SET TO 5 (MODIFY MODE)
IF 'V' INPUT
WKBUF = WORKING BUFFER CONTAINING INTERFACE TABLE
WITH LITERALS IN COMPACTED FORM
AMA = XXXXX = NEW INTERFACE TABLE AND LITERAL AREA
(NOT DONE IF CALLED BY LIBRARY MAINTENANCE)

NOTE

USES Routines
EXEC
XRED
XILIT
XLPRM
XIMPT
THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'IMM' DIRECTIVE OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE. NOTE: ALL INITIALIZATION, INCLUDING MBUFE (OLD INTERFACE TABLE), HAS BEEN PERFORMED BY XENIX.

BEGIN XENIX

IF #P (NO. OF PARAMETERS) > 0, THEN
SET ArgNo (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0

PRMTNO = 1 => CREATE M MODE
PRMTNO = 3 => CREATE A MODE
PRMTNO = 4 => CREATE CONTINUE MODE
PRMTNO = 5 => MODIFY MODE

DO UNTIL 'EXIT' OR "X" IS ENTERED
CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTNO, SIZE, TYPE, AND STATUS OF NEXT ARGUMENT
CALL XITCOM TO PROMPT USER AND RETURN PARSED INPUT
IF 'X' WAS NOT ENTERED, THEN
IF '\' WAS ENTERED, THEN
SET PRMTNO TO 5
ELSE IF NOTHING WAS ENTERED (I.E. TOKEN IS COS), THEN
INCREMENT TO NEXT ARGUMENT
ELSE CALL XIPMT TO PROCESS THE USER'S INPUT
ENDIF
ENDIF
ENDO

IF A 'X' WAS ENTERED, THEN
SET RETURN CODE INDICATING X
(I.E. MASSTA = 0)
ELSE
COMpress THE LITERAL LIST AREA
ENDIF
ENDIF

IF STORE INTERFACE TABLE AS NEWNAME THEN
STORE INTERFACE TABLE AS NAME
IF STORE INTO AWA FAILED, THEN
SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
ELSE
SET GOOD RETURN CODE
ENDIF
ENDIF

END XENIX
1 * CONSTRUCT PROMPT TO BE ISSUED
2 BEGIN XIPRN
3 DO UNTIL A PROMPT IS CONSTRUCTED
4 IF PRNTND = 5, THEN
5 CONSTRUCT A ';' PROMPT
6 ELSE IF PRNTND = 4 (CONTINUE MODE), OR
7 PRNTND = 6 (CONTINUE HERE MODE), THEN
8 IF ARGNO IS A SCALAR, THEN
9 CONSTRUCT PROMPT AS '\ARG=' OR '\ARG:'
10 ELSE IF PRNTND NOT = 6, THEN
11 COMPUTE ISUB AS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LAST)
12 ELSE SET PRNTND TO 4
13 COMPUTE CURRENT ELEMENT NO. (SUBSCRIPT) FROM CURRENT INDEX
14 ENDIF
15 IF THERE ARE NO EMPTY SLOTS BEYOND LAST, THEN
16 SET PRNTND TO NOOSAV (EXIT THE CONTINUE MODE)
17 ELSE IF ARGNO IS DOUBLY SUBSCRIPTED PARAMETER, THEN
18 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
19 CONSTRUCT PROMPT AS '\ARG=(I,J)'
20 ELSE CONSTRUCT PROMPT AS '\ARG=(ISUB)'
21 ENDIF
22 ELSE IF ARGNO = BP, THEN
23 SET PRNTND TO 5
24 ELSE INCEREMENT ARGNO TO NEXT PARAMETER
25 SET IFLAG TO 1, 2, OR 3 INDICATING 1:0, OR 10
26 SET LAST ENTERED INDICATOR (LAST) TO 0
27 IF PRNTND = 3, THEN
28 IF SOME DATA VALUE(S) OR PARAM NAME EXISTS FOR ARGNO, THEN
29 CAL XLISD TO LIST DATA FOR THIS ARGUMENT
30 ENDIF
31 CONSTRUCT PROMPT AS '\ARG=' OR 0 OR =B
32 ELSE IF ARGNO IS MARKED INCOMPLETE, THEN
33 IF A PARTIAL LITERAL LIST EXISTS, OR
34 THIS ARGUMENT IS A SCALAR, THEN
35 COMPUTE ISUB AS FIRST EMPTY ELEMENT
36 IF DOUBLY SUBSCRIPTED PARAMETER, THEN
37 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
38 CONSTRUCT PROMPT '\ARG=(I,J)'
39 ELSE CONSTRUCT PROMPT '\ARG=(ISUB)'
40 ENDIF
41 ELSE CONSTRUCT PROMPT AS '\ARG=' OR 0 OR =B
42 ENDIF
43 ENDIF
44 ENDIF
45 ENDIF
46 ENDIF
47 ENDIF
48 ENDIF
49 ENDIF
50 ENDIF
51 ENDIF
52 ENDIF
53 ENDIF
FOR CALLING PROCEDURE

CALL XPHT

PROMPT DIRECTIVE PROCESSOR

INPUT

COMMON IXE - COMBUF, COMPTA, TOKENS

OUTPUT

COMMON XB - ARGNO, FRMTMD

* XPHT PROCEDURE IS THE PROMPT DIRECTIVE

BEGIN XPHT

EXIT IF TOKEN IS NOT COMMA :ERROR2:

POSITION TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT NAME :ERROR2:

ERREXIT IF TOKEN(S) FOLLOW THE NAME :ERROR2:

IF NAME IS 'A', THEN

SET FRMTMD TO 9

ELSE IF NAME IS 'A', THEN

SET FRMTMD TO 3

ELSE

ERREXIT :ERROR2:

ENDIF

END IF ARGNO TO 0

EXIT TO :RETURN:

:ERROR2: CALL XHNSG FOR 'INVALID SYNTAX'

:RETURN:

END XPHT
* XLIST PROCESSES THE LIST DIRECTIVE

1 BEGIN XLIST

IF TOKEN IS ' - ', THEN
  POSITION TO NEXT TOKEN
ENDIF

ERREXIT IF TOKEN IS NOT NAME : ERR02:

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

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

ELSE
  SET MODEFG TO 2
ENDIF

IF TOKEN IS EOS, THEN
  WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
  AND STATUS
  DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
  IF MODEFG = 1 OR MODEFG = 3, THEN
    CALL XICHNR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
  ENDF
  IF MODEFG = 2 OR MODEFG = 3, THEN
    CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
  ENDF
ENDIF

ELSE
  DO UNTIL EOS IS REACHED
  ERREXIT IF TOKEN IS NOT COMMA : ERR02:
  ERREXIT IF NEXT TOKEN IS NOT NAME : ERR02:
  SET ARGNO TO 1
  START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
  EXIT IF NAME = ARGNO'S NAME IN PROPMT TABLE
  IF MODEFG = 1 OR MODEFG = 3, THEN
    CALL XICHNR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
  ENDF
  IF MODEFG = 2 OR MODEFG = 3, THEN
    CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
  ENDF
ENDIF
ENDLOOP

PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
ENDSEARCH
INCREMENT TO NEXT TOKEN
ENDDO

EXIT TO :RETURN:

:ERR02: CALL XRMSC TO WRITE 'INVALID SYNTAX'

:RETURN:

END XLIST
1 CDO FORTRAN CALLING PROCEDURE
2 CDO CALL XISUB
3 C************
4 CDO 1 CDO EFFECTIVE SUBSCRIPT CALCULATION ROUTINE
5 CDO 2 CDO INPUT
6 CDO 2 CDO COMMON XE - COMBUF, COMPR, TOKENS
7 CDO 2 CDO COMMON XB - IDIM, ISIZE, LENEFF
8 CDO 2 CDO COMMON X0 - IRET, ISUB
9 CDO 2 CDO
10 C************
11 CDO 5 CDO OUTPUT
12 CDO 5 CDO USES ROUTINES
13 CDO 5 CDO XRMSG
14 CDO 5 CDO
15 C************
16 CDO 5 CDO XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
17 CDO 5 CDO THE INPUT SUBSCRIPT
18 CDO 5 CDO BEGIN XISUB
19 CDO 5 CDO INCREMENT TO NEXT TOKEN
20 CDO 5 CDO ERROR IF TOKEN IS NOT INTEGER VALUE :ERR14:
21 CDO 5 CDO IF IDIM FOR THIS ARGUMENT > 0, THEN
22 CDO 5 CDO ERROR IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
23 CDO 5 CDO INCREMENT TO NEXT TOKEN
24 CDO 5 CDO ERROR IF TOKEN IS NOT COMMA :ERR15:
25 CDO 5 CDO INCREMENT TO NEXT TOKEN
26 CDO 5 CDO ERROR IF TOKEN IS NOT INTEGER VALUE :ERR14:
27 CDO 5 CDO CALCULATE ISUB AS (J-1)*IDIM+1
28 CDO 5 CDO ELSE
29 CDO 5 CDO SET ISUB TO INTEGER VALUE
30 CDO 5 CDO ENDIF
31 CDO 5 CDO ERROR IF ISUB > 2**13 :ERR16:
32 CDO 5 CDO INCREMENT TO NEXT TOKEN
33 CDO 5 CDO ERROR IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
34 CDO 5 CDO EXIT TO :RETURN:
35 CDO 5 CDO :ERR14: CALL XRMSG - 'INVALID SUBSCRIPT SYNTAX'
36 CDO 5 CDO :ERR15: CALL XRMSG - 'DOUBLY SUBSCRIPTED - MUST SPECIFY BOTH'
37 CDO 5 CDO :ERR16: CALL XRMSG - 'INVALID SUBSCRIPT VALUE'
38 CDO 1 END XISUB
FORTRAN CALLING PROCEDURE

CALL XDAT

***********
LITERAL DATA PROCESSOR

***********
INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XE - CFLAG, COMFLG, DFLAG, IARG, IARG4,
ISIZE, ISIZE4, XTYPE, LEN, LITDP,
LITOWN, LITSIZE, MODAV, MXBTH, NOBITM,
PRMTHD, SFLAG, WKBNG, WKBUF

***********
OUTPUT

COMMON XE - IRET, ISUB, LITDP, LITOWN, MXBTH,
PRMTHD, WKBUF

***********
INTERNAL VARIABLES

***********
COMMON XS -

IDISP = INDEX INTO WBUF OF LOCATION FOR THIS
LITERAL DATUM
NUMCP = NUMBER OF CONSECUTIVE ELEMENTS TO BE
MARKED COMPLETED AS A RESULT OF THIS
STACK = PUSH-DOWN LIST (MAX. OF 4 ENTRIES)
DESCRIBING NESTED REPEAT GROUPS.
REPEAT GROUP IS PARENTHE TICALLY
GROUPED

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

***********
NOTES

USES ROUTINES

XXX

***********

REPRODUCIBILITY INDEX OF THE
IF THIS IS A DATA ELEMENT, THEN
(I.E. INTEGER, REAL, DOUBLE OR CHAR.)
IF ARG'S TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
ELSE IF ARG'S TYPE IS NOT SAME AS ITYPE :ERR10:
ELSE IF ARG TYPE IS FREE, THEN
FOR DATA INPUT
ELSE THIS MUST BE CHARACTER DATA BEING INPUT
IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
ENDIF
INCREMENT TOKEN POINTER TO THE DATA INPUT
VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
ARGUMENT FOR DATA INPUT (ISUB <= ISIZE - LEMNOV + 1)
ERREXIT IF INSUFFICIENT SPACE :ERR11:
ENDIF
MOVE DATA FROM INPUT COMMUNICATIONS BUFFERS TO LITERAL AREA
SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
ARGUMENT, TO LEMNOV
IF LEMNOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
MOVE LEMNOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
ENDIF
INCREMENT TO NEXT TOKEN
INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LEMNOV
SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
ELSE IF TOKEN INDICATES NON-DATA ELEMENT
IF THIS IS A "(" , THEN
CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITYPE) AND
SECONDARY DIMENSION (IDIM)
EXIT XIAD IF ERR0R (IURET < 0)
ELSE IF THIS IS AN "=" , THEN
IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
ELSE
INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
LENGTH OF ONE ELEMENT (LENEFF)
ENDIF
TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
INCREMENT TO NEXT TOKEN
ELSE
  IF TOKEN IS A REPEAT COUNT, THEN
    IF NEXT TOKEN IS '(', THEN
      SET PARENFLAG TO 1
    ENDIF
    INCREMENT TO NEXT TOKEN
  ELSE
    SET PARENFLAG TO 0
 ENDIF
SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
ELSE
  EREXIT (INVALID FIELD) :ERROR:
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
            EREXIT IF TOKEN IS NOT A COMMA :ERROR:
          ELSE
            INCREMENT TO NEXT TOKEN
          ENDIF
        ENDIF
      ENDIF
  ENDIF
ENDIF
ENDDO
IF PROMPT MODE NOT CONTINUE (=A), AND
  (THERE ARE EMPTY ELEMENTS BEYOND LASTE, OR
  PREVIOUS TOKEN WAS A COMMA), THEN
RETAIN PROMPT MODE (SET MODESW TO PRMTMD)
SET PROMPT MODE TO CONTINUE (=A)
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
1 EXIT TO :RETURN:
2 :ERROR2: CALL XRM5G "INVALID SYNTAX"
2 :ERROR7: CALL XRM5G "ONLY DATA VALID TO RIGHT OF ="
2 :ERROR10: CALL XRM5G "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
2 :RETURN:
1 END XIDAT
*XIMP PROCESSES THE USER'S INPUT TEXT

BEGIN XIMP

IF PRMTD = 5, THEN

SET IFLAG OFF (=0)

EXIT: IF TOKEN IS NOT A NAME :ERROR2:

SAVE NAME AND POSITION TO NEXT TOKEN

IF TOKEN IS "=" , THEN

POSITION TO NEXT TOKEN

IF TOKEN IS 'b', THEN

SET IFLAG TO 10

POSITION TO NEXT TOKEN

ELSE

SET IFLAG TO 1

ENDIF

ELSE

IF TOKEN IS 'q', THEN

SET IFLAG TO 0

ENDIF

ENDIF

IF IFLAG NOT SET, THEN

CASE NAME (:EXIT: :PROMPT: :LIST:)

EXIT: IF ANOTHER TOKEN FOLLOWS :ERROR2:

:EXIT: SET IRET TO THAT PROMPTING LOOP TERMINATES

:PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE

:LIST: CALL XLIST TO PROCESS LIST DIRECTIVE

ENDCASE

ENDIF

START SEARCH UNTIL # OF ENTRIES

EXIT IF NAME FOUND IN PROMPT TABLE

SET ARGNO TO ENTRY #

SET ISUB TO 1

ORELSE

INCREMENT TO NEXT PROMPT TABLE ENTRY

ENDCMP

ERREXIT :ERROR1:

ERREXIT IF IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:

ENDIF

IF NEXT TOKEN IS A NAME, THEN

CALL XIPAR TO PROCESS A PARAMETER FIELD

ELSE

ERREXIT IF IFLAG IS NOT I ("=") :ERROR8:

CALL XIDAT TO PROCESS DATA LIST

ENDIF

EXIT XIMP

EXIT TO :RETURN:

:ERROR2: CALL XIMSG "INVALID SYNTAX"

:ERROR8: CALL XIMSG "MUST USE PARAMETER NAME TO RIGHT OF B OR =B"

:RETURN:

END XIMP
FORTRAN CALLING PROCEDURE

CALL XIPAR

*********

PROCESS AN INPUT PARAMETER NAME AND ANY ASSOCIATED SUBSCRIPT

FIELD(S)

*********

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - DFLAG, IARG, IAREA, ICURRENT, LITDSP, SFLAG, WKBLNG, WKBUF

*********

OUTPUT

COMMON XB - IRETC, LITDWN, WKBUF

*********

NOTES

USES ROUTINES

XRMC

XRMS

XRES
1 * XIPAR PROCFSSES A USER SPECIFIED PARAMETER FIELD

1 BEGIN XIPAR

2 IF A NAME IS SPECIFIED, THEN

3 INCOREMENT TO NEXT TOKEN

4 IF TOKEN IS '(' , THEN

5 PROCESS I AND J SUBSCRIPTS

6 ERREY IF INVALID SUBSCRIPTING :ERR14:

7 IF DOUBLY SUBSCRIPTED, THEN

8 SET S-FLAG IN ARGNO'S SPECS. FIELD

9 SET LITOSP IN ARGNO'S SPECS. TO NEXT LITERAL AREA SPACE (LITDOWN)

10 PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT

11 ELSE

12 SET LITOSP IN ARGNO'S SPECS TO ISUB

13 ENDIF

14 ELSE

15 SET LITOSP IN ARGNO'S SPECS TO 0

16 ENDIF

17 ERREY IF ORDE HAS MORE THAN A CHARACTERS :ERR18:

18 ERREY IF EXTRAN'OUS FIELD INPUT :ERR02:

19 TURN OF D-FLAG (SAME LITERAL DATA) IN ARGNO'S SPECS.

20 SET PARAMETER NAME INTO ARGUMENT'S CHARACTERISTICS

21 SET COMPLETE (AND S) FLAG IN ARGUMENT'S CHARACTERISTICS

22 IF ALL ARGUMENTS ARE COMPLETE, THEN

23 SET IFFACE TABLE COMPLETE FLAG

24 ENDIF

25 ELSE

26 MUST BE A & INPUT

27 "REXIT IF NOT AN AMPERAND (&) INPUT :ERR02:

28 CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS

29 SET ARGUMENT AND INTERFACE TABLE INCOMPLETE

30 ENDIF

31 EXIT TO :RETURN:

32 :ERR02: CALL XRMSC - "INVALID SYNTAX"

33 :ERR14: CALL XRMSC - "INVALID SUBSCRIPT SYNTAX"

34 :ERR18: CALL XRMSC - "INVALID ORDE NAME"

35 :RETURN:

36 END XIPAR
<table>
<thead>
<tr>
<th>CALL</th>
<th>PROCEDURE</th>
<th>DATA OR PARAMETER VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>XILSD</td>
<td>LIST DATA OR PARAMETER VALUES FOR ONE ARGUMENT</td>
</tr>
</tbody>
</table>

**INTERNAL VARIABLES**

<table>
<thead>
<tr>
<th>COMMON SS - BUFFER</th>
<th>LINE TO BE OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER - INDEX INTO BUFFER FOR NEXT ASCII DATA</td>
<td></td>
</tr>
<tr>
<td>RETURN - RETURN IDENTIFIER FOR INTERNAL ROUTINES</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**

<table>
<thead>
<tr>
<th>XE</th>
<th>ETA</th>
<th>RAG</th>
<th>EKE</th>
<th>XILE</th>
<th>REK</th>
<th>ZEUK</th>
</tr>
</thead>
</table>
833 1 * XILSD WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
833 2 * IS RETURNED AS A PROMPT.
833 3 BEGIN XILSD
833 4 SET ARGUMENT NAME INTO BUFFER
833 5 USE I-O-FLAG TO DETERMINE WHICH OF 'B', 'N', OR 'xN'
833 6 WILL GO INTO THE PRINT BUFFER
833 7 IF D-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
833 8 IF A PARAMETER NAME IS SPECIFIED, THEN
833 9 PUT PARAMETER NAME INTO BUFFER
833 10 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
833 11 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
833 12 PUT SUBSCRIPT INTO BUFFER
833 13 ELSE
833 14 IF LIDSP OF ARGUMENT IS > 0, THEN
833 15 COMPUTE AND CONVERT THIS SUBSCRIPT
833 16 PUT SUBSCRIPT INTO BUFFER
833 17 ENDF
833 18 ENSDIF
833 19 WRITE OUT THE PRINT BUFFER BUILT
833 20 ENDSIF
833 21 ELSE
833 22 LOCATE LITERAL LIST AND MASK
833 23 IF SYMBOLIC STRING, THEN
833 24 CALL FILES TO PRINT SYMBOLIC STRING
833 25 ELSE
833 26 DO UNTIL ALL ELEMENTS PROCESSED
833 27 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
833 28 UNTIL ALL ELEMENT'S PROCESSED
833 29 COMPUTE AND CONVERT THE SUBSCRIPT
833 30 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
833 31 PUT "x" INTO BUFFER
833 32 ELSE
833 33 CONVERT THE DATA USING XN05, XN14, OR XN16
833 34 PUT DATA AND "x" INTO BUFFER
833 35 ENDF
833 36 ENDSIF
833 37 ENDSIF
833 38 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
833 39 REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
833 40 WRITE OUT THE PRINT BUFFER BUILT
833 41 ENDSIF
833 42 ENDSIF
833 43 1 END XILSD
FORTRAN CALLING PROCEDURE
CALL XILSS

XILSS IS CALLED BY XILSP TO LIST SYMBOLIC STRING DATA

COMMON IXE = LISTING, IXAPP = PRINTING, IXCD = NAME
COMMON IXB = BUFFER ALREADY INITIALIZED WITH OUTPUT COMMON IXS = BUFFER INTO WGROUP OF SYMERIC STRING DATA
COMMON IXT = BUFFER, IXPT = DATAP

INPUT

COMMON IXE = LISTING, IXAPP = PRINTING, IXCD = NAME
COMMON IXB = BUFFER ALREADY INITIALIZED WITH OUTPUT COMMON IXS = BUFFER INTO WGROUP OF SYMERIC STRING DATA
COMMON IXT = BUFFER, IXPT = DATAP

OUTPUT
INTERNAL VARIABLES

1. 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 (TOKSZI) = NO. OF WORDS IN SYMBOLIC STRING

2. ENTRY TOKEN WORD 1 \((\text{SIZE})\) WORD 2 \((\text{FIELD})\) WORD 3 \((\text{TOKSZI})\)
3. 001 CD4 9 \(\text{INTEGER} \\#3\) 0 \(\text{->CALL X16}\) 0
4. 002 CD4 4 \(\text{REAL} \\#7\) 0 \(\text{->CALL X14}\) 3
5. 003 CD4 0 \(\text{DOUBLE} \\#9\) 0 \(\text{->CALL X16}+1\) 4
6. 005 CD4 0 \(\text{NAME} \\#3\) 0 \(\text{->USE 3 WORDS}\) 4
7. 006 CD4 0 \(\text{FOLLOWING TOKEN} \quad \text{BEGINNING \& FAST}\)
8. 007 CD4 2 \(\text{TOKEN} \quad \text{TOKEN}\)
9. 008 CD4 0 \(\text{->ERROR (INVALID)}\) 0
10. 009 CD4 1 + 1 + 1
11. 010 CD4 1 - 1 - 1
12. 011 CD4 2 * 1 + 1
13. 012 CD4 1 / 1 / 1
14. 013 CD4 1 > 1 > 1
15. 014 CD4 0 < 0 < 1
16. 015 CD4 0 \# 0 \# 1
17. 016 CD4 0 \# 0 \# 1
18. 017 CD4 0 \# 0 \# 1
19. 018 CD4 0 \# 0 \# 1
20. 019 CD4 0 \# 0 \# 1
21. 020 CD4 0 \# 0 \# 1
22. 021 CD4 0 \# 0 \# 1
23. 022 CD4 0 \# 0 \# 1
24. 023 CD4 0 \# 0 \# 1
25. 024 CD4 0 \# 0 \# 1
26. 025 CD4 0 \# 0 \# 1
27. 026 CD4 0 \# 0 \# 1
28. 027 CD4 0 \# 0 \# 1
29. 028 CD4 0 \# 0 \# 1
30. 029 CD4 0 \# 0 \# 1
31. 030 CD4 0 \# 0 \# 1
32. 031 CD4 0 \# 0 \# 1
C程序

1 C05
2 NOTES
3 C05
4 USES ROUTINES
5 C05
6 EXEC
7 C05
8 X016
9 C05
10 X014
11 C05
12 X018
13 C05
14 X000
15 C05
16 X000
17 C05

1 *******
2 BEGIN XILSS
3 MOVE A * CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
4 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
5 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
6 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSZ)
7 EXIT TO ERROR 2 IF FIELD = 0
8 IF SIZE < 0, THEN
9 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
10 ENDIF
11 IF TOKSIZ < 0, THEN
12 SET TOKSIZ TO SIZE + 2
13 ENDIF
14 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
15 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
16 SET BUFFER POINTER TO 1ST POSITION FOR DATA
17 CLEAR PRINT BUFFER TO BLANKS
18 ENDIF
19 IF FIELD > 0, THEN
20 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
21 ELSE
22 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
23 ONE: CALL XR16 WITH VALUE IN NEXT WORD OF SYMB. STRING
24 AND PUT RESULTS INTO PRINT BUFFER
25 TWO: CALL XR014 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
26 AND PUT RESULTS INTO PRINT BUFFER
27 THREE: CALL XR016 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
28 AND PUT RESULTS INTO PRINT BUFFER
29 MOV THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
30 FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
31 SIX: CALL XR16 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
32 INTO PRINT BUFFER FOLLOWED BY AN "R"
33 EXIT: PUT A * CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
34 INDEX BY 1
35 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
36 EXIT XILSS
37 END CASE
38 INCREMENT PRINT BUFFER INDEX BY SIZE
1015 1 CDO FORTRAN CALLING PROCEDURE
1016 1 CDO
1017 1 CDO
1018 1 CDO CALL XICFR
1019 1 CDO
1020 1 C********
1021 1 CDO
1022 1 CDO PRINT THE CHARACTERISTICS OF AN ARGUMENT
1023 1 CDO
1024 1 C********
1025 1 CDO
1026 1 CDO INPUT
1027 1 CDO
1028 1 CDO COMMON XE - LU
1029 1 CDO
1030 1 CDO COMMON XV - ARNGO, IOFLAG, ISIZE, ITYPE,
1031 1 CDO LENCEF, NARG
1032 1 CDO
1033 1 C********
1034 1 CDO
1035 1 CDO NOTES
1036 1 CDO
1037 1 CDO USES ROUTINES
1038 1 CDO
1039 1 CDO EXEC
1040 1 CDO EXEC
1041 1 CDO EXEC
1042 1 CDO EXEC
1043 1 CDO EXEC
1044 1 CDO
1045 1 C********
1046 1 C********
1047 1 BEGIN XICFR
1048 2 WRITE ARGUMENT CHARACTERISTICS
1049 2 BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPTS, I/O TYPE AND
1050 2 DATA TYPE
1051 2 WRITE OUT THE PRINT BUFFER
1052 1 END XICFR
1053  1 CD0       FORTRAN CALLING PROCEDURE
1054  1 CD0
1055  1 CD0
1056  1 CD0       CALL XEXIT
1057  1 CD0
1058  1 CD0
1059  1 CD0       ************
1060  1 CD0       EXTRACT VARIOUS FIELDS OF AN ARGUMENT 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       ************
1069  1 CD0       COMMON XB - ARGNO, ISIZE, WBUF
1070  1 CD0       COMMON XB - DFLAG, DFLAG, IARG, IARGA,
1071  1 CD0       COMMON XB - DCLASS, IDIM, DFLAG, ISIZE,
1072  1 CD0       COMMON XB - ISUB, ITYPE, LENEFF, LIDSP,
1073  1 CD0       COMMON XB - LITIZ, NDIBTM, NOBITM, SFLAG
1074  1 CD0
1075  1 CD0       ************
1076  1 CD0       ************
1077  1 CD0       ************
1078  1 CD0       ************
1079  1 CD0       ************
1080  1 CD0       ************
1081  1 CD0       NOTES
1082  1 CD0       ************
1083  1 CD0       USES ROUTINES
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       BEGIN XEXIT
1092  1 CD0       USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
1093  2 CD0       IN THE WORKING BUFFER
1094  2 CD0       EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
1095  1 CD0       END XEXIT
107 1 CD0 FORTRAN CALLING PROCEDURE
108 1 CD0
109 1 CD0 CALL XILIT
110 1 CD0
111 1 CD0
112 1 CD0
113 1 CD0
114 1 CD0
115 1 CD0
116 1 CD0
117 1 CD0
118 1 CD0
119 1 CD0
120 1 CD0
121 1 CD0
122 1 CD0
123 1 CD0
124 1 CD0
125 1 CD0
126 1 CD0
127 1 CD0
128 1 CD0
129 1 CD0
130 1 CD0
131 1 CD0
132 1 CD0
133 1 CD0
134 1 CD0
135 1 CD0
136 1 CD0
137 1 CD0
138 1 CD0
139 1 CD0
140 1 CD0
141 1 CD0
142 1 CD0
143 1 CD0

FORTRAN CALLING PROCEDURE
CALL XILIT

PACK LITERAL ENTRIES INTO FORMAT FOR STORAGE OF INTERFACE
TABLE INTO AWA

INPUTS
COMMON XB - ARGO, DFLAG, IARG4, ISIZE,
LENFF, LITDSP, LITDOWN, LITPTR,
LITSIZE, LITN, NEWBM, NOBITM
NUMARG, SFLAG, WBLNG, WBUF

OUTPUTS

COMMON XB - LITDOWN, LITLEN, LITPTR, WBUF

INTERNAL VARIABLES
COMMON X5 - LITUP = INDEX INTO WBUF OF AREA FOR NEXT LITERAL
ENTRY TO BE MOVED INTO
LITUP = INDEX INTO WBUF OF LITERAL ENTRY TO BE
COMPRESSED AND MOVED

NOTES

USES ROUTINES
XIEXT
XHIBIT
XMOV
XNXT
XRSET

/* AUTOINCREMENT OR THE
ORIGINAL PAGE IS POOR */
1145 1 * PACK LITERAL AREA INTO FORMAT FOR STORAGE OF INTERFACE TABLE
1146 1 BEGIN XLIST
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 (LENENF) OF LITERAL
1156 11 UP IN THE WORKING BUFFER
1157 12 ENDEF
1158 13 ENDDO
1159 14 ENDIF
1160 15 CALCULATE NEW DISPLACEMENT AND SET IN LITDSP
1161 16 ENDIF
1162 17 ENDDO
1163 18 ENDDO
1164 1 END XLIST
**RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:**

1. CD
2. CD
3. CD
4. CD
5. CD
6. CD
7. CD
8. CD
9. CD
10. CD
11. CD
12. CD
13. CD
14. CD
15. CD
16. CD
17. CD
18. CD
19. CD
20. CD
21. CD
22. CD
23. CD
24. CD
25. CD
26. CD
27. CD
28. CD
29. CD
30. CD
31. CD
32. CD
33. CD
34. CD
35. CD
36. CD
37. CD
38. CD
39. CD
40. CD
41. CD
42. CD
43. CD
44. CD
45. CD
46. CD
47. CD
48. CD
49. CD
50. CD
51. CD
52. CD
53. CD

**INTERNAL VARIABLES:**

1. PROM1 - (INTEGER, 31 WORDS) PROMPT FOR OPTION
2. TEMPO - (INTEGER, 1 WORD) TEMPORARY USED FOR RESPONSE
3. TEMPZ - (INTEGER, 29 WORDS) THE 29 TOKENS TO BE

**EXECUTION AND SUBROUTINES USED:**

1. XELS, XLEDF, XDEL, XLINT, XLMOD, XLMSG,
2. XLMRM, XLEPRO, XTCOM, XMMOV, XMSG

**XE COMMON USED:**

1. EQUIVALENCE (XE(1), LU )
2. +XE(2), ICASS, XE(3), ISECU ,
3. +XE(4), FLAGS, XE(5), MASMRA,
4. +XE(6), SUBSTR, XE(7)
5. +XE(7), VALFIL, XE(8), TOKENS,
6. +XE(142), ICR, XE(145), COMBUF
BEGIN XLMAN
CALL RMPAR TO GET INPUT PARAMETERS
INITIALIZE COMMON TO ZERO
SET CLASS NUMBER TO ZERO
CALL EXEC TO GET A CLASS NUMBER
EXIT XLMAN IF SECURITY CODE IS NOT VALID
CALL XRMOV TO INITIALIZE TOKENS IN COMMON
DO FOREVER
:PROMPT:
:INITIALIZE MASTER AND SUBSTATE FLAGS
:CALL XTCM TO PROMPT FOR OPTION
:IF XTCM RETURN CODE IS NOT ZERO OR
:FIRST TOKEN IS NOT AN INTEGER OR
:INTEGER > 7 THEN
:CALL XRMSE TO WRITE INVALID RESPONSE
:GO TO :PROMPT:
:ENDIF
CASE INTEGER (:XLPRM::XLPRM::XLPRM::XLPRM::XLDEL::XLMOD::
:XLMSG::XLDBF::XLDBF::XLDBF::EXIT::
:XLPRM:
:SET NUMBER TO INTEGER
:CALL XELDS TO LOAD XLPRM TO CREATE SYSTEM PROMPT FILE
:XLPRO:
:SET VALFLG TO SAY ORIGINAL XLPRO REQUEST
:CALL XELDS TO LOAD XLPRO TO ADD A PROCESSOR
:DO UNTIL VALFLG SAYS EXIT (X)
:CALL XELDS TO LOAD XLIST TO ENTER DEFAULT VALUES
:CALL XELDS TO LOAD XLPRO TO COMPLETE PROCESSING
:ENDDO
:XLDEL:
:CALL XELDS TO LOAD XLDEL TO DELETE A PROCESSOR
:XLMOD:
:SET VALFLG TO SAY ORIGINAL XLMOD REQUEST
:CALL XELDS TO LOAD XLMOD TO MODIFY A PROCESSOR
:DO UNTIL VALFLG SAYS EXIT (X)
:CALL XELDS TO LOAD XLIST TO ENTER DEFAULT VALUES
:CALL XELDS TO LOAD XLMOD TO COMPLETE PROCESSING
:ENDDO
:XLMSG:
:CALL XELDS TO LOAD XLMSG TO ADD A MESSAGE
:XLDBF:
:CALL XELDS TO LOAD XLDBF TO HANDLE DATA BASE FILES
:ENDCASE
ENDDO
:EXIT:
:CALL EXEC TO RELEASE CLASS NUMBER
END XLMAN
### Fortran Calling Procedure:

CALL XELS ('XLPBM')

### Creates One of the System Prompt Files Depending on "Number"

#### Input from COMMON:

**NUMBER** - (INTEGER, 1 WORD) Users response indicating which system prompt file to create:
- 1 - >IPRM
- 2 - >IPRM
- 3 - >IPRM

#### Internal Variables:

**COM3** - (INTEGER, 1 WORD) First token in COMBUF
**COM4** - (INTEGER, 1 WORD) First data in COMBUF
**MSG** - (INTEGER, 1 WORD) Contains appropriate message number
**NAME** - (INTEGER, 9 WORDS) 5 element array, each element is a system prompt file name
**MODIR** - (INTEGER, 1 WORD) Number of directives
**NOTOK** - (INTEGER, 1 WORD) Number of tokens
**PROMS** - (INTEGERS) PROM2, PROM3, PROM4 are all user prompt arrays

### RTE Functions and Subroutines Used:

**RCVT**, **CLOSE**, **CREATE**, **PURGE**, **WAITF**

### FRS Functions and Routines Used:

**XMOV**, **XMSG**, **XTCOM**

### Common Used:

**EQUIVALENCE**
- *(X(3), ISECU), (X(7), NUMB )*
- *(X(145), XER), (X(145), COMEB )*
- *(X(145), NOTOK), (X(147), COMS )*
- *(X(146), COMA ), (X(1), IBUF )*
- *(X(128), MODIR)**
160 1 BEGIN XLPMB
161 2 :PRM1:
162 3 CALL XCTOM TO PROMPT USER FOR 0 DIRECTIVES
163 3 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
164 3 CALL CREAT TO CREATE PROMPT FILE
165 3 IF FILE ALREADY EXISTS THEN
166 3 CALL XCTOM TO PROMPT USER FOR SCRATCH/CANCEL
167 3 EXIT XLPMB IF RESPONSE IS CANCEL
168 3 CALL PURGE TO PURGE PROMPT FILE
169 3 ERREXIT IF PURGE ERROR TO :FILERR:
170 3 GO TO :PRM1:
171 3 ELSE (CREATE NEW FILE)
172 3 ERREXIT IF CREAT ERROR TO :FILERR:
173 3 CALL XCTOM TO PROMPT USER FOR LIST OF DIRECTIVES
174 3 ERREXIT IF LIST IS INCONSISTENT WITH 0 OF DIRECTIVES TO :ERR1:
175 3 DO FOR EACH DIRECTIVE
176 4 4 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
177 4 4 CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
178 4 4 ENDBO
179 4 5 CALL WRTFT TO WRITE LIST OF DIRECTIVES TO FILE
180 4 5 ERREXIT IF WRTFT ERROR TO :FILERR:
181 4 5 DO FOR EACH DIRECTIVE
182 4 6 CALL XCTOM TO PROMPT USER FOR DEFINITION
183 4 6 DO UNTIL END IS REACHED IN RESPONSE
184 5 7 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185 5 7 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186 5 7 MOVE RESPONSE INTO BUFFER
187 5 7 SET CONTROL CHARACTERS IN BUFFER
188 5 7 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
189 5 7 ENDDO
190 4 8 SET REMAINDER OF BUFFER TO NULL
191 4 8 CALL WRTFT TO WRITE DEFINITION
192 4 8 ERREXIT IF WRTFT ERROR TO :FILERR:
193 4 8 ENDBO
194 3 9 CALL CLOSE TO CLOSE FILE
195 3 9 ERREXIT IF CLOSE ERROR TO :FILERR:
196 3 9 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
197 2 ENDF
198 1 EXIT XLPMB
199 2 :ERR1: (ERROR IN RESPONSE)
200 2 CALL XMSG TO DISPLAY ERROR IN RESPONSE
201 2 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
202 2 :FILERR: (FILE ACCESS ERROR)
203 2 CALL XMSG TO DISPLAY ERROR
204 2 CALL CLOSE TO CLOSE FILE
205 2 CALL PURGE TO PURGE FILE
206 1 END XLPMB
1 BEGIN XLPRO
2 IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
3 INITIALIZE MASTER AND SUB STATES
4 PERFORM LIBD TO UPDATE LIBRARY DIRECTORY
5 IF PROCESSOR HAS AN INTERFACE TABLE THEN
6 SET RETN = 2
7 :PRMPT2:
8 CALL XTCOM TO PROMPT FOR & PARAMETERS
9 ERREXIT IF XTCOM RETURN CODE IS NOT ZERO OR
10 ERREXIT IF & PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
11 CREATE HEADEP ENTRY WITH &PARAMETERS AND PROCESSOR NAME
12 CALL XLPS TO WRITE INSTRUCTIONS FOR ENTERING SPECS
13 DO FOR &PARAMETERS
14 CALL XLPS TO CREATE ONE PARAMETER ENTRY
15 :ENDPS
16 SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
17 CALL XLPFL TO CREATE PROMPT FILE
18 SET RETN = 3
19 :PRMPT3:
20 CALL XTCOM FOR DEFAULT VALUES DECISION
21 ERREXIT IF RETURN CODE IS NOT ZERO OR
22 ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
23 IF RESPONSE WAS YE THEN
24 SET FLAG TO CALL INTERFACE TABLE EDITOR
25 EXIT XLPRO
26 ENDIF
27 ELSE
28 SET CODES ARRAY TO ADD ONLY ABSTRACT
29 CALL XLPFL TO CREATE PROMPT FILE
30 CALL XLPS TO WRITE INSTRUCTIONS FOR ENTERING SPECS
31 PERFORM XLPRO - NO RETURN EXPECTED
32 ENDIF
33 ENDIF
34 CALL TLFL TO CREATE THE DEFAULT INTERFACE TABLE/FILE
35 SET XFLAG TO SAT ORIGINAL REQUEST TO ADD A PROCESSOR
36 PERFORM XLPRO - NO RETURN EXPECTED
37 :PRMERR:
38 CALL XMSG TO DISPLAY ERROR MESSAGE
39 GO TO (:PRMPT1; :PRMPT2; :PRMPT3), RETN
40 END XLPRO
CD************
CD
CD FORTRAN CALLING SEQUENCE:
CD CALL XLDB
CD
CD************
CD XLDB CREATES A NEW DATA BASE FILE (MDB/PDB) FROM AN OLD
CD DATA BASE FILE (MDB/PDB) AND DELETES THE OLD FILE
CD
CD************
CD
CD************
CD INPUTS IN COMMON:
CD XE(3) QUAL, XE(7) NUMBR, XE(8) SECU
CD XE(142) ICR, XB(3) OLD_FIX, XB(6) NEW_FIX
CD XB(9) TOTSI
CD
CD************
CD
CD************
CD INTERNAL VARIABLES:
CD XB(17) FREC - FIRST RECORD NUMBER IN DATABUF
CD XB(18) LREC - LAST RECORD NUMBER IN DATABUF
CD XB(10) IDC - DCB FOR OLD_FIX
CD XB(56) IDC2 - DCB FOR NEW_FIX
CD XB(20) TOLBUF - BUFFER FOR COMPLETE DATA BASE TOC
CD
CD************
CD
CD************
CD RTE Routines Used:
CD CLOSE, CREAT, KCVT, OPE,
CD PURGE, READF, WRITF
CD
CD************
CD PSD ROUTINES USED:
CD XODB, XREX, XREX
CD
CD************
CD COMMON USED:
CD EQUVALENCE (XE(3), QUAL, XE(7), NUMBR, XE(8) SECU)
CD + (XE(17), NUMBR), (XE(13), SECU)
CD + (XE(14), ICR), (XB(3) OLD_FIX)
CD + (XB(6), NEW_FIX), (XB(9), TOTSI)
CD + (XB(10), DBCH), (XB(12), DBCH)
CD + (XB(13), OTOC), (XB(14), SIZE)
CD + (XB(56), IDC), (XB(16), IDC)
CD + (XB(20), TOLBUF)
CD
CD************
CD
BEGIN XLCDB
CALL OPEN TO OPEN OLD FIL
ERREXIT IF OPEN ERROR TO :ERR3:
CALL READ TO READ IN HEADER RECORD
ERREXIT IF READ ERROR TO :ERR2:
COMPUTE NUMBER OF TOC RECORDS (NOTOC) AS(# ENTRIES + 16)/ 16
IF NOTOC > 1 THEN
CALL READ TO READ IN REMAINING TOC RECORDS
ERREXIT IF READ ERROR TO :ERR2:
ENDIF
SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
CALL CREATE TO CREATE NEW FIL
ERREXIT IF CREATE ERROR TO :ERR2:
DO FOR EACH TOC ENTRY
IF REQUEST WAS 0B TO MOB THEN
ERREXIT IF NAME > 4 CHAR OR
ERREXIT IF CLASS IS 'DDB' AND NAME > 2 CHAR TO :ERR1:
APPEND MOB CONVENTION TO FRONT OF NAME
ELSE
REMOVE MOB CONVENTION FROM NAME
ENDIF
ENDDO
CALL WRITE TO WRITE NEW TOC RECORDS TO NEW FIL
ERREXIT IF WRITE ERROR TO :ERR1:
SET TOCPTR TO FIRST TOC ENTRY
SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
CONVERT SIZE FROM BLOCKS TO WORDS
DO UNTIL ALL RECORDS ARE COPIED (SIZE = 0)
SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1224)
IF SIZE < LENGTH THEN
SET LENGTH TO SIZE
ENDIF
CALL READ TO READ LENGTH WORDS OF DATA
ERREXIT IF READ ERROR TO :ERR1:
DECREMENT SIZE BY LENGTH READ
UPDATE FIRST AND LAST RECORD NUMBERS
STARTSEARCH FOR TOCPTR = TOCPTR TO LAST TOC ENTRY
EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
IF CLASS IS AN INTERFACE TABLE THEN
COMPUTE INDEX INTO DATBUFF FROM FIRST RECORD #, RECORD # IN TOC ENTRY
AM INDEX IN TOC ENTRY
SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
ENDIF
ENDESEARCH
CALL WRITE TO WRITE LENGTH WORDS OF DATA TO NEW FIL
ERREXIT IF WRITE ERROR TO :ERR1:
ENDDO
FORTRAN CALLING SEQUENCE:

CALL XLEOS ('XLEOS')

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

INPUTS IN COMMON:

XE(3) ISECU, XE(7) NUMBR,
XE(85) TOKENS, XE(142) ICR

INTERNAL VARIABLES IN COMMON

XE(3) QUAL - USER QUALIFIER REQUIRED BY XDB ROUTINES AND XRFM
XE(8) SECU - TEMPORARY SLOT FOR ISECU DURING THIS OVERLAY
XE(5) OLDLIN - OLD FILE NAME
XE(6) NEWL - NEW FILE NAME
XE(9) TOTL - SIZE OF OLDL/NEWL IN BLOCKS
XE(10) FICHR - 4 BASE CHARACTERS OF FILE NAME
XE(12) QUALF - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPEW

FDS ROUTINES USED:

XDBDA, XDBRF, XERH, XLCDB, XLPCW

COMMON USERS:

ENQUIVENCE (XE(3), ISECU),
+XE(3) QUAL), (XE(7), NUMBR),
+XE(8) SECU), (XE(85), TOKENS),
+XE(142), ICR), (XE(145), COMDF),
+XE(5), OLDLIN), (XE(6), NEWL),
+XE(9), TOTL), (XE(10), FICHR),
+XE(12), QUALF), (XE(99), IERR),
+XE(100), IDBC)
BEGIN XLFMF:
1 IF ERROR ON OPEN PDB LOG FILE
2 THEN CALL EXEC FOR NAME AND USER
3 ELSE IF OPEN ERROR SET FILE NOT FOUND
4 ELSE RETURN
5 ENDIF
6 END IF
7 CALL EXEC FOR NAME AND USER
8 IF ERROR ON OPEN PDB LOG FILE
9 THEN CALL EXEC FOR NAME AND USER
10 ELSE IF OPEN ERROR SET FILE NOT FOUND
11 ELSE RETURN
12 ENDIF
13 ELSE IF NAME TO ADD PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
14 ELSE IF FILE MANAGER ERROR
15 ELSE IF MAXIMUM NUMBER OF ENTRIES EXIST TO :FILIER:
16 ELSE IF NAME TO ADD PDB NAME TO LOG FILE
17 ELSE RETURN
18 ENDIF
19 ELSE IF NAME TO ADD PDB NAME TO LOG FILE
20 ELSE RETURN
21 ENDIF
22 ELSE IF NAME TO ADD PDB NAME TO LOG FILE
23 ELSE RETURN
24 ENDIF
25 ELSE IF NAME TO ADD PDB NAME TO LOG FILE
26 ELSE RETURN
27 ENDIF
28 ELSE IF NAME TO ADD PDB NAME TO LOG FILE
29 ELSE RETURN
30 ENDIF
31 END IF
32 CALL XLCDF TO COPY OLD FILE TO NEW FIL
33 END IF
34 END IF
35 END XLFMF
36
37 5-134
588 1 CD************
589 1 CD
590 1 CD  FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
591 1 CD
592 1 CD  CALL XELDS ('XLDEL')
593 1 CD
594 1 CD************
595 1 CD1
596 1 CD1 XLDEL DELETES A PROCESSOR FROM THE LIBRARY DIRECTORY AND
597 1 CD1 THE PROMPT FILE. IF THE PROCESSOR HAS AN INTERFACE TABLE,
598 1 CD1 IT DELETES THE DEFAULT INTERFACE TABLE FILE ALSO.
599 1 CD1
600 1 CD************
601 1 CD4
602 1 CD4  INTERNAL VARIABLES:
603 1 CD4  COMNAM - (INTEGER, 7 WORDS) IS A TEMPORARY WORK AREA
604 1 CD4  WHERE ENTRY IN LIBRARY DIRECTORY IS STRIPPED
605 1 CD4  DOWN TO PROCESSOR NAME
606 1 CD4  PRNAM - (INTEGER, 2 WORDS) PROCESSOR NAME TO BE
607 1 CD4  DELETED.
608 1 CD4
609 1 CD************
610 1 CD5
611 1 CD5  RTE/ FMGR ROUTINES USED:
612 1 CD5  IAND,XCVT,CREAT,OPEN,READF,WRITE,CLOSE,PURGE
613 1 CD5
614 1 CD5  FDS Routines USED:
615 1 CD5  XRCP, XNEXT, XRMOV, XRMSG, XRRC, XRSET, XRUPK, XTCOM
616 1 CD5
617 1 CD5
618 1 CD5  XE AND XB COMMON USED
619 1 CD5
620 1 CD5  EQUIVALENCE  (XE(3), ISECU,)
621 1 CD5  + (XE(142), ICR,)
622 1 CD5  + (XE(145), COMBUF,)
623 1 CD5  + (XS(49), LIB01,)
624 1 CD************
1 BEGIN XDEL

2 :PRMPT:

3 CALL XTCOM TO PROMPT FOR PROCESSOR NAME

4 IF XTCOM RETURN CODE IS NOT ZERO OR

5 RESPONSE IS NOT A VALID PROCESSOR NAME THEN

6 CALL XMSG TO WRITE ERROR MESSAGE

7 GO TO :PRMPT:

8 EXIT XDEL IF RETURN CODE SAYS % ENTERED

9 ENSIF

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 XMSG TO DISPLAY ERROR

14 GO TO :PRMPT:

15 ENSIF

16 DECIMATE # PROCESSORS BY 1

17 CALL PURGE TO PURGE OLD LIBRARY DIRECTORY

18 ERREXIT IF RETURN CODE < ZERO TO :FILERR:

19 1 EXIT XDEL 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 XMSG TO SAY FILE PURGED SUCCESSFULLY

26 IF PROCESSOR HAD AN INTERFACE TABLE THEN

27 CREATE DEFAULT IF NAME

28 CALL PURGE TO PURGE DEFAULT IF

29 ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:

30 CALL XMSG TO DISPLAY 'FILE SYMBOL NOT FOUND' MESSAGE

31 ENSIF

32 GO TO :PRMPT:

33 :FILERR: CALL XMSG TO WRITE FILE ACCESS ERROR

34 1 END XDEL
FORTRAN CALLING SEQUENCE:
CALL XLIFL

XLIFL CREATES THE DEFAULT INTERFACE TABLE FILE

INTERNAL VARIABLES

PROCEDURES - (INTEGER, 1 WORD) # PROCEDURES TO BE ALLOCATED TO
THE FILE

FDS ROUTINES USED:
XREX, XMSG

RTE ROUTINES USED:
CLOSE, CREAT, WRITF

COMMON USED:

EQUIVALENCE (X0(3), ISECU),
+ (X0(142), ICR ), (X0(90), LITPIR),
+ (X0(91), LITLEM),
+ (X0(96), MOPARM), (X0(101), HEPD ),
+ (X0(108), PAMS ), (X5(1), IERR ),
+ (X5(2), IDCB )
699  1 BEGIN XLFL
700  2 EXTRACT LITERAL LENGTH FROM HEADER
701  2 COMPUTE # BLOCKS FOR THIS FILE
702  2 CALL CREAT TO CREATE DEFAULT INT TABLE FILE
703  2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
704  2 CALL WRITE TO WRITE HEADER AND SPEC
705  2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
706  2 IF THERE IS A LITERAL RECORD THEN
707  3  CALL WRITE TO WRITE LITERALS
708  2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
709  2 ENDEF
710  2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
711  2 CALL XRMSG TO DISPLAY FILE (NAME) CREATED MESSAGE
712  2 EXIT XLFL
713  1 EXIT XLFL
714  2 :FILERR:
715  2 CALL XRMSG TO WRITE FILE ACCESS ERROR
716  2 CALL XEXIT TO RETURN TO MAIN PROGRAM
717  1 END XLFL
1 BEGIN XLIINT
2
3  SET MASTER STATE AS EXEC
4  SET SUBSTATE AS INTERFACE TABLE EDITOR
5  SET UP LIST OF VALID DIRECTIVES
6  SET UP LIST FLAG TO SET ENTIRE LIST
7  SET CURRENT ARGUMENT TO ZERO
8  SET PROMPT MODE TO ALL
9  SET INDEX INTO PARMS OF SHORT PROMPTS
10  SET NEW TABLE NAME TO ZERO
11  INITIALIZE ARRAY OF TYPE LENGTHS
12  IF LITERAL LENGTH IS ZERO THEN
13      SET LITERAL POINTER TO FIRST AVAILABLE WORD
14  ELSE
15      SET LITERAL POINTER TO FIRST WORD OF LITERALS
16  CALL XINT TO UNPACK LITERALS
17  ENDIF
18  CALL XINT TO ACCEPT DEFAULT VALUES
19  CALL ZERTW TO RETURN
20 END XLIINT
808 1 CDO**********
809 1 CDO
810 1 CDO
811 1 CDO
812 1 CDO
813 1 CDO
814 1 CDO**********
815 1 CBT
816 1 CBT
817 1 CBT
818 1 CBT
819 1 CBT
820 1 CDO
821 1 CDO
822 1 CDO
823 1 CDO
824 1 CDO
825 1 CDO
826 1 CDO
827 1 CDO
828 1 CDO
829 1 CDO
830 1 CDO
831 1 CDO
832 1 CDO
833 1 CDO
834 1 CDO
835 1 CDO
836 1 CDO
837 1 CDO
838 1 CDO
839 1 CDO
840 1 CDO
841 1 CDO
842 1 CDO
843 1 CDO
844 1 CDO
845 1 CDO
846 1 CDO
847 1 CDO
848 1 CDO**********

FORTRAN CALLING SEQUENCE:
CALL XMOD ('XLMOD')

XLMOD 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, POINT, READF, WRITF

FDS ROUTINES USED:
XLFL, XLIFS, XLFLS, XLPS, XREPB, XREXT,
XREOB, XAMSG, XRPCF, XRSET, XRPUCF, XTRON

COMMON USED:

EQUIVALENCE (XEC(3), XECU ),
+ (XEC(5), XASMTA), (XEC(6), XASBUA),
+ (XEC(7), XALFLA), (XEC(13), XAMHMA),
+ (XEC(14), XAPWMA), (XEC(85), XBEDS),
+ (XEC(86), XINTOK), (XEC(69), XMAKOK),
+ (XEC(97), XNCGOK), (XEC(142), XNC ),
+ (XEC(143), XCONUF), (XEC(261), XEVER ),
+ (XEC(33), XFLAGS), (XEC(35), XFLAG ),
+ (XEC(37), XRGH), (XEC(38), XIT ),
+ (XEC(43), XLM ), (XEC(44), XEMGFLA ),
+ (XEC(88), XLIWSP), (XEC(90), XLIPTB),
+ (XEC(92), XHIXS), (XEC(96), XHOPARA),
+ (XEC(101), XEDB ), (XEC(101), XEDB1 ),
+ (XEC(104), XIBD2 ), (XEC(105), XIBD3 )
BEGIN XLMOD

IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
SET RTH = 1

CALL XLMOD TO DISPLAY SPEC INSTRUCTIONS
INITIALIZE MASTER AND SUB STATE FLAGS
CALL XTCP TO PROMPT FOR PROCESSOR NAME
EXIT XLMOD IF RETURN CODE SAYS X ENTERED
ERREXIT IF XCON RETURN CODE NON-ZERO OR
ERREXIT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO :PRMERR:
CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
ERREXIT IF THERE IS A FILE ERROR TO :FILERR:
ERREXIT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO :PRMERR:
SAVE INTERFACE TABLE BIT AND VERSION NUMBER
PERFORM VERSION TO UPDATE VERSION NUMBER
IF THE PROCESSOR HAS AN INTERFACE TABLE THEN
SET NEW VERSION NUMBER IN INTERFACE TABLE
CREATE DEFAULT INTERFACE TABLE NAME
CALL OPEN AND READF TO READ IN NEDR AND SPECS
IF THERE ARE LITERALS THEN
CALL READF TO READ IN LITERALS ENDIF

CALL CLOSE TO CLOSE FILE
ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
CREATE PROMPT TABLE NAME
CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
CALL NHMF TO RENAME PROMPT FILE >XLMPS
ERREXIT IF NHMF ERROR TO :FILERR:
SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPECS
PERFORM DELPAR TO DELETE PARAMETERS
PERFORM MODPAR TO MODIFY PARAMETERS
PERFORM ADDPAR TO ADD PARAMETERS
CALL XLPLF TO CREATE NEW PROMPT FILE
PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES ELSE

CALL NHMF TO RENAME PROMPT FILE >XLMPS
ERREXIT IF NHMF ERROR TO :FILERR:
SET CODES ARRAY TO MODIFY ABSTRACT ONLY
CALL XLPLF TO CREATE NEW PROMPT FILE
PERFORM XLMOD — NO RETURN EXPECTED ENDIF

ENDIF CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
ERREXIT IF FILE ERROR TO :FILERR:
CALL XLIIF TO CREATE NEW DEFAULT INTERFACE TABLE FILE
SET VALFLG TO SAY ORIGINAL REQUEST TO MODIFY
PERFORM XLMOD — NO RETURN EXPECTED

:PRMERR:
CALL XMSG TO DISPLAY ERROR MESSAGE
GO TO (:PRM1,:PRM2,:PRM3,:PRM4,:PRM5,:PRM6,:PRM7),RTH

:FILERR:
CALL XMSG TO DISPLAY FILE ERROR
END XLMOD
905  1 BEGIN VERSION
906  2 SET RTN = 2
907  2 :PRG2;
908  2  CALL XCTOM TO PROMPT USER FOR VERSION NUMBER
909  2  IF RETURN CODE IS NOT CR THEN
910  3  ERREXIT IF RETURN CODE IS NON-ZERO ON
911  3  ERREXIT IF VERSION IS INVALID TO :PRGRR;
912  3  IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
913  4  CALL XRSET TO PUT NEW VERSION IN ENTR
914  4  CALL OPEN, WRITF, CLOSE TO UPDATE LIBRARY DIRECTORY
915  4  ERREXIT IF FILE ERROR TO :FILERR;
916  3  ENDF
917  2  ENDF
918  1  END VERSION
919  1  *=
920  1  *=
921  1  *=
922  1 BEGIN DELPRM
923  2 SET RTN = 3
924  2 :PRG3:
925  2 DO UNTIL RETURN CODE IS CR ENTERED
926  3  IF NUMBER OF PARAMETERS > 1 THEN
927  4  CALL XCTOM TO PROMPT FOR DELETE PARAMETER NAME
928  4  IF RETURN CODE IS NOT CR ENTERED THEN
929  5  PERFORM RSPMD TO INTERPRET RESPONSE
930  5  PERFORM CHDT TO CHECK FOR EXISTING DATA
931  5  SET ARNO TH WORD IN CODES TO SAY 'DELETE U'
932  5  CALL XREM TO MOVE DATA TO DELETE PARAMETER
933  5  DECUMENT # PARAMETERS BY 1
934  4  ENDF
935  3  ELSE
936  4  CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
937  3  EXIT DELPRM
938  3  ENDF
939  2  ENDDO
940  1  END DELPRM
941  1  *=
942  1  *=
943  1  *=
944  1 BEGIN MODPRM
945  2 SET RTN = 4
946  2 :PRG4:
947  2 DO UNTIL RETURN CODE IS CR ENTERED
948  3  CALL XCTOM TO PROMPT FOR MODIFY PARAMETER NAME
949  3  IF RETURN CODE IS NOT CR ENTERED THEN
950  4  PERFORM RSPMD TO INTERPRET RESPONSE
951  4  PERFORM CHDT TO CHECK FOR EXISTING DATA
952  4  SET ARNO TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
953  4  CALL VLPR TO PROMPT USER FOR SPECIFICATIONS
954  4  SET IT COMPLETE BIT OFF
955  4  ENDF
956  2  ENDDO
957  1 END MODPRM
1 BEGIN ADDPRM
2 SET RTN = 5
3
4 PROMPT:
5 DO UNTIL RETURN CODE IS CR ENTERED
6 IF NUMBER OF PARAMETERS < 63 THEN
7 CALL XCOM TO PARSE FOR ADD PARAMETER BEFORE/AFTER PARAMETER NAME
8 IF RETURN CODE IS NOT CR ENTERED THEN
9 PERFORM RSPMD TO INTERPRET RESPONSE
10 SET ARGNO TH NON-DELETED WORD IN CODES TO SAY 'ADDED'
11 INCREMENT NUMBER OF PARAMETERS BY 1
12 CALL XMov TO MOVE DATA TO MAKE SPACE FOR NEW PARAMETER
13 CALL XLSPS TO GET NEW SPECS FOR THIS PARAMETER
14 SET IT COMPLETE BIT OFF
15 ENDIF
16 ELSE
17 CALL XMEN TO DISPLAY NO MORE PARAMETERS CAN BE ADDED
18 END ADDPRM
19 ENDIF
20 END ADDPRM
21
22 PROMPT:
23 CALL XCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
24 CALL XCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
25 ERNAME IF RETURN CODE IS NON-ZERO TO :PRMERR:
26 IF RESPONSE IS YES THEN
27 SET VFLAG TO SAY CALL INTERFACE TABLE EDITOR
28 EXIT XLMOD
29 ENDIF
30 END DEFAULT
994 1 BEGIN RSPNO
995 2  ERREXIT IF RETURN CODE IS NON-ZERO TO :PRMERR:
996 2  SET NEGATIVE FLAG OFF
997 2  IF NEGATIVE IS REQUESTED THEN
998 3  SET NEGATIVE FLAG ON
999 2  ENDIF
1000 2  ERREXIT IF PARAMETER NAME NOT IN INTERFACE TABLE TO :PRMERR:
1001 2  CONVERT PARAMETER NAME TO ARGUMENT NUMBER
1002 1 END RSPNO
1003 1 *
1004 1 *
1005 1 *
1006 1 BEGIN CHOT
1007 2  EXTRACT SFLAG, CFLAG AND DISP FROM SPEC ENTRY
1008 2  IF DOUBLE SUBSCRIPT FLAG IS ON THEN
1009 3  SET LITERAL ENTRY LENGTH TO 2
1010 2  ELSE
1011 3  IF THERE IS COMPLETE LITERAL DATA THEN
1012 4  SET LITERAL ENTRY LENGTH TO SIZE
1013 3  ENDF
1014 2  ENDF
1015 1 END CHOT

XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
XLMD
**FORTRAN CALLING PROCEDURE**

**CALL XHLD ("XHMSG")**

**XHMSG PROVIDES MAINTENANCE OF THE FDG MESSAGE FILE XHMSG**

**INPUT**

**COMMON - LU, IQSEQ, FLAGS, TOKENS, ICR**

**COMMON - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES**

**COMMON - UPDATE MODE, MESSAGE NUMBER AND TEXT**

**COMMON - MESSAGE FILE - DIRECTORY AND OLD TEXT**

**OUTPUT**

**COMMON - COMBUF**

**COMMON - MESSAGE FILE - DIRECTORY AND TEXT UPDATES**

**LOCAL**

**AREA - NUMERICAL AREA INDICATOR FOR MESSAGE**

**DIRECT - MESSAGE DIRECTORY (SEE SSD 6.2.4.12)**

**INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY**

**FILE MANAGER DATA CONTROL BLOCK**

**FILE MANAGER & XLibrary RETURN CODE**

**MESSAGE NUMBER WITHIN MESSAGE AREA**

**MESSAGE NUMBER WITHIN FILE**

**CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION**

**UPDATE MODE - NUMBER OF BLOCK CONTAINING MESSAGE**

**MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 33, 65 OR 97)**

**NOTES**

**USES APSSW, CLOSE, CREAT, EXEC, IAND, KCVT, OPEN, READF, WRITF,**

**XERNS, XIG, XRMV, XHMSG, XTCOM, XUDBG**

**WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE**
**EXISTING TEXT IN PLACE.**

**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 ENDF
1083 17 ENDDO
1084 18 CREATE A CLEARED FILE OF TOTAL REQUIRED SIZE
1085 19 ELSE
1086 20 READ DIRECTORY
1087 21 ENDF
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 EXIST)
1095 29 THEN
1096 30 DISPLAY OLD MESSAGE TEXT
1097 31 ENDF
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 ENDF
1107 41 ENDF
1108 42 ELSE
1109 43 OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110 44 ENDF
1111 45 ENDDO
1112 46 REWRITE DIRECTORY BLOCK
1113 47 END XLMSG
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1115</td>
<td>CO</td>
</tr>
<tr>
<td>1116</td>
<td>CO0</td>
</tr>
<tr>
<td>1117</td>
<td>CO0</td>
</tr>
<tr>
<td>1118</td>
<td>CO0</td>
</tr>
<tr>
<td>1119</td>
<td>CO0</td>
</tr>
<tr>
<td>1120</td>
<td>CO0</td>
</tr>
<tr>
<td>1121</td>
<td>CO0</td>
</tr>
<tr>
<td>1122</td>
<td>CD1</td>
</tr>
<tr>
<td>1123</td>
<td>CD1</td>
</tr>
<tr>
<td>1124</td>
<td>CD1</td>
</tr>
<tr>
<td>1125</td>
<td>CD1</td>
</tr>
<tr>
<td>1126</td>
<td>CD2</td>
</tr>
<tr>
<td>1127</td>
<td>CD2</td>
</tr>
<tr>
<td>1128</td>
<td>CD2</td>
</tr>
<tr>
<td>1129</td>
<td>CD2</td>
</tr>
<tr>
<td>1130</td>
<td>CD2</td>
</tr>
<tr>
<td>1131</td>
<td>CD2</td>
</tr>
<tr>
<td>1132</td>
<td>CO3</td>
</tr>
<tr>
<td>1133</td>
<td>CO3</td>
</tr>
<tr>
<td>1134</td>
<td>CO3</td>
</tr>
<tr>
<td>1135</td>
<td>CO3</td>
</tr>
<tr>
<td>1136</td>
<td>CO3</td>
</tr>
<tr>
<td>1137</td>
<td>CO3</td>
</tr>
<tr>
<td>1138</td>
<td>CO3</td>
</tr>
<tr>
<td>1139</td>
<td>CO3</td>
</tr>
<tr>
<td>1140</td>
<td>CO3</td>
</tr>
<tr>
<td>1141</td>
<td>CO3</td>
</tr>
<tr>
<td>1142</td>
<td>CO3</td>
</tr>
<tr>
<td>1143</td>
<td>CO3</td>
</tr>
<tr>
<td>1144</td>
<td>CO3</td>
</tr>
<tr>
<td>1145</td>
<td>CO3</td>
</tr>
<tr>
<td>1146</td>
<td>CO3</td>
</tr>
<tr>
<td>1147</td>
<td>CO0</td>
</tr>
</tbody>
</table>

**Fortran Calling Sequence:**

CALL XLPCR

**XLPCR Creates and Initializes the PDB Log File**

**Inputs from Common:**

XE(3) ISECU, XE(142) ICR

**RTE Routines Used:**

CLOSE, CREAT, WRITF

**FD5 Routines Used:**

XRMOV, XRMSG, XTCOM

**Common Used:**

(EQV, XE(5), TOKENS, XE(142), ICR)

(EQV, XE(145), COMBUF, XE(99), IERI)

(EQV, XE(100), IDC, XE(116), IBUF)
FORTRAN CALLING SEQUENCE:

CALL XLPFL (MCOD, CODES)

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XEC(1) LU, XEC(3) ISECU, XEC(16) PROCNUM,
XEC(142) ICR, XEC(96) NOPARM, XEC(108) PARMS

RTE FUNCTIONS USED:

CLOSE, CREAT, EXEC, OPEN,
POSNT, PURGE, READ, WRIT

FDX FUNCTIONS USED:

XERTN, XICPR, XP MOV, XIRMSG,
XRPCX, XRPX, XITCOM

COMMON USED:

EQUIVALENCE XEC(1), LU, XEC(3), ISECU, XEC(16), PROCNUM,
XEC(142), ICR, XEC(96), NOPARM, XEC(108), PARMS
1227 1 BEGIN XLPF
1228 2 COMPUTE SIZE OF FILE AS 0 PARAMETERS +3
1229 3 CALL CREAT TO CREATE PROMPT FILE
1230 3 ERREXIT IF CREAT ERROR TO :FILERR:
1231 5 STUFF SYNTAX RECORD & AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS
1232 5 CALL WRITF TO WRITE SHORT PROMPT RECORD(S)
1233 3 ERREXIT IF WRITF ERROR TO :FILERR:
1234 3 IF ABSTRACT CODE IS MODIFY THEN
1235 4 CALL OPEN TO OPEN OLD PROMPT FILE >XLTMP
1236 3 ERREXIT IF OPEN ERROR TO :FILERR:
1237 3 CALL READ TO READ EXISTING ABSTRACT
1238 3 ERREXIT IF READ ERROR TO :FILERR:
1239 3 CALL EXEC TO DISPLAY EXISTING ABSTRACT
1240 3 CALL XTPM TO PROMPT USER TO MODIFY ABSTRACT
1241 3 IF RESPONSE IS CR (NO MODIFICATION) THEN
1242 4 CALL WRITF TO WRITE EXISTING ABSTRACT TO NEW FILE
1243 4 ERREXIT IF WRITF ERROR TO :FILERR:
1244 3 ELSE (NEW ABSTRACT WAS ENTERED)
1245 4 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1246 3 ENDF
1247 2 ELSE (ABSTRACT CODE IS AD)
1248 3 CALL XTTPM TO PROMPT USER TO ENTER NEW ABSTRACT
1249 3 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1250 3 ENDF
1251 2 IF PARAMETERS IS NOT ZERO THEN
1252 3 PERFORM EXTPTM TO GET DEFINITIONS FOR EACH PARAMETER
1253 3 ENDF
1254 2 CALL CLOSE TO CLOSE PROMPT FILE
1255 2 ERREXIT IF CLOSE ERROR TO :FILERR:
1256 2 IF THERE WAS AN OLD FILE THEN
1257 3 CALL CLOSE TO CLOSE OLD FILE >XLTMP
1258 3 CALL PURGE TO PURGE OLD FILE >XLTMP
1259 2 ENDF
1260 1 EXIT XLPF
1261 2 :FILERR:
1262 2 CALL XRMSP TO DISPLAY ERROR CODE
1263 2 CALL CLOSE TO CLOSE NEW FILE
1264 2 CALL CLOSE TO CLOSE OLD FILE >XLTMP
1265 2 CALL PURGE TO PURGE OLD FILE >XLTMP
1266 2 SET VALFLG = 0
1267 2 CALL XERTN TO RETURN TO MAIN ***NO RETURN TO HERE***
1268 1 END XLPF
1270 1 BEGIN FORMAT
1271 2 INITIALIZE TOKEN POINTER AND TOTAL WORD COUNT
1272 2 DO UNTIL EOS IS DETECTED IN RESPONSE
1273 2 ERREXIT IF RESPONSE IS NOT CHARACTER STRING TO :ERR1:
1274 2 ERREXIT IF RESPONSE IS TOO LONG (>128 WORDS) TO :ERR1:
1275 2 CALL XMOV TO MOVE RESPONSE TO BUFFER
1276 2 SET CONTROL CHARACTERS IN BUFFER
1277 2 INCREMENT TOTAL WORD COUNT BY THIS RESPONSE
1278 2 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
1279 2 INCREMENT TOKEN POINTER TO NEXT CHARACTER STRING
1280 2 ENDDO
1281 2 SET REMAINDER OF BUFFER TO NULL
1282 2 CALL WRITF TO WRITE NEW RESPONSE TO PROMPT FILE
1283 2 ERREXIT IF WRITF ERROR TO :FILEERR:
1284 1 END FORMAT
1285 1 :ERR1:
1286 1 CALL XMSG TO DISPLAY INVALID RESPONSE
1287 1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
1299 1 BEGIN EXTPRM
1300 2 DO FOR EACH ENTRY IN CODES ARRAY
1301 3 BEGIN CASE (:MOD,:DEL,:ADD,:EXTR), CODE$)
1302 4 :MOD: (SPECs WERE MODIFIED)
1303 4 CALL READ TO READ EXISTING DEFINITION FROM XLTMP
1304 4 ERREXIT IF READ ERROR TO :FILERR:
1305 4 CALL EXEC TO DISPLAY EXISTING DEFINITION
1306 4 IF RESPONSE IS CR (NO RESPONSE) THEN
1307 5 CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
1308 5 ELSE (NEW DEFINITION WAS ENTERED)
1309 5 PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
1310 5 ENDIF
1311 4 :ADD: (SPECs WERE ADDED)
1312 4 CALL XCTOM TO PROMPT USER TO ENTER NEW DEFINITION
1313 4 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1314 3 END CASE
1315 3 DO UNTIL RESPONSE IS CR (EXIT)
1316 4 CALL XCTOM TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
1317 3 EXIT EXTPRM IF RESPONSE IS CR
1318 4 SEARCH SHORT PROPTS FOR RESPONSE
1319 4 ERREXIT IF NOT FOUND TO :ERR:
1320 4 CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
1321 4 ERREXIT IF READ ERROR TO :FILERR:
1322 4 CALL EXEC TO DISPLAY EXISTING DEFINITION
1323 4 CALL XCTOM TO PROMPT USER TO MODIFY DEFINITION
1324 4 IF RESPONSE IS POSITIVE
1325 5 CALL POINTS TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
1326 5 ERREXIT IF POINT ERROR TO :FILERR:
1327 5 PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
1328 4 ENDIF
1329 3 ENDEXIT
1330 2 END;
1331 1 END EXTPRM
FORTAN CALLING SEQUENCE:
CALL XLPMO

XLPMO MODIFIES THE EXISTING PDB LOG FILE

INPUTS FROM COMMON:
XE(1) LU, XE(3) QAL, XE(8) SECU,
XE(42) ICA, XB(99) IERR, XB(100) IINCB,

INTERNAL VARIABLES IN COMMON:
XE(52) UNO - NUMBER OF USER ID A-Z (1-26)
XE(53) RNC - RECORD # WHERE UNO'S PDB LIST IS (2-14)
XE(52) PHAM - FILE NAME OF USER'S PDB (FULL 6 CHARs)
XE(52) RINF - NUMBER OF FILES THAT MUST BE DELETED
XE(54) WHEN NEW MAT IS CURRENTLY IN USE
XE(244) OUTFUB-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

RTE ROUTINES USED:
CLOSE, EXEC, KEVT, PURGE, READF, WRITF

FDS ROUTINES USED:
XRCP, XKEVT, XRMOV, XRMGS,
XRDFH, XRFR, XTCOM

COMMON USED:

1 CO************
1387 1 BEGIN XLPMO
1388 2 DO UNTIL USER RESPONSE IS EXIT (3)
1389 3 CALL XCTOM TO PROMPT USER FOR USER ID
1390 4 ERREXIT IF RESPONSE IS INVALID TO :ERR1:
1391 5 CALL RREAD TO READ IN THE RECORD CONTAINING THIS ID
1392 6 ERREXIT IF READ ERROR TO :FILERR:
1393 7 DISPLAY MAX # ALLOWED FOR THIS USER AND # CURRENTLY USED
1394 8 CALL XCTOM TO PROMPT USER FOR NEW MAXIMUM
1395 9 IF RESPONSE IS A MODIFICATION (NOT CR) THEN
1396 10 ERREXIT IF RESPONSE IS NOT VALID TO :ERR1:
1397 11 COMPUTE DIFFERENCE AS &CURRENTLY USED - NEW MAXIMUM
1398 12 IF DIFFERENCE <= 0 THEN
1399 13 CALL WRITF TO REWRITE UPDATED RECORD TO FILE
1400 14 ERREXIT IF WRITE ERROR TO :FILERR:
1401 15 ELSE
1402 16 DISPLAY LIST OF CURRENT FILES
1403 17 SET &PURGED = 0
1404 18 DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1405 19 CALL XCTOM TO PROMPT USER TO DELETE #PURGED FILES
1406 20 DO FOR EACH FILE NAME IN RESPONSE
1407 21 SEARCH LIST FOR FILE NAME
1408 22 ERREXIT IF INVALID NAME TO :ERR1:
1409 23 MOVE FILE NAME TO PURGE LIST
1410 24 INCREMENT &PURGED BY 1
1411 25 DECREMENT DIFFERENCE BY 1
1412 26 COMPRESS OLD FILE NAME OUT OF LIST
1413 27 ENDDO
1414 28 ENDDO
1415 29 IF RESPONSE WAS NOT EXIT THEN
1416 30 CALL WRITF TO REWRITE UPDATED LOG RECORD
1417 31 ERRREXIT IF WRITE ERROR TO :FILERR:
1418 32 DO FOR &PURGED FILES
1419 33 CALL XKMN TO CREATE FILE NAME
1420 34 CALL PURGE TO PURGE FILE
1421 35 ENDDO
1422 36 ENDFRF
1423 37 ENDFRF
1424 38 ENDFRF
1425 39 ENDFRF
1426 40 CALL CLOSE TO CLOSE NDB/PDB LOG FILE
1427 41 EXIT XLPMO
1428 42 :ERR1:
1429 43 CALL XMSG TO DISPLAY ERROR IN RESPONSE
1430 44 RETURN TO REISSUE LAST PROMPT
1431 45 :FILERR:
1432 46 CALL XMSG TO DISPLAY FILE ERROR
1433 47 CALL CLOSE TO CLOSE NDB/PDB FILE
1434 48 EXIT XLPMO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

CLASS - (INTEGER, 1 WORD, XE(15)) IS THE PARAMETER CLASS
IPMT - (INTEGER, 1 WORD) IS THE INDEX INTO PARMS WHERE
THE SHORT PROMPT IS TO BE STORED
ISPEC - (INTEGER, 1 WORD) IS THE INDEX INTO PARMS WHERE
THE SPECS 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, XRCPY, XRMOV, XRMSG, XRSET, XTCOM

XE COMMON USED:

EQUIVALENCE (XE(8), RETC ), (XE(9), RETH ),
+ (XE(10), CLASS ), (XE(11), TYPE ),
+ (XE(145), COMBIF ), (XE(37), ARGNO ),
+ (XI(96), WOPARM ), (XW(108), PARMS )
1474 1 BEGIN XLSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII

1476 2 :PRMT1:
1477 2 SET RETURN TO 1
1478 2 CALL XTCOM TO PROMPT FOR PROMPT, CLASS, TYPE AND I/O FLAGS
1479 2 ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
1480 2 INITIALIZE THIS SHORT PROMPT AND ENTRY
1481 2 ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
1482 2 ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
1483 2 CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
1484 2 ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
1485 2 CALL XRSET TO SET CLASS IN ENTRY
1486 2 ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
1487 2 ERREXIT IF CLASS IS CODE AND TYPE IS SYMBOLIC STRING TO :PRMERR:
1488 2 CALL XRSET TO SET TYPE IN ENTRY
1489 2 ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
1490 2 ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
1491 2 CALL XRSET TO SET I/O FLAGS IN ENTRY
1492 2 IF CLASS IS DAT TYPE THEN
1493 3 IF TYPE IS NOT SYMBOLIC STRING THEN
1494 4 SET RETURN TO 2

1495 4 :PRMT2:
1496 4 CALL XTCOM TO PROMPT FOR I AND J DIMENSIONS
1497 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1498 4 ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
1499 4 I: J DIMENSION IS NOT ENTERED THEN
1500 5 SET J DIMENSION TO 1
1501 4 ELSE
1502 5 ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
1503 5 IF J DIMENSION < 2 THEN
1504 6 SET J DIMENSION TO 1
1505 ELSE
1506 6 STORE I DIMENSION IN ENTRY
1507 6 :ENDIF
1508 4 :ENDIF
1509 4 SET SIZE = I DIMENSION * J DIMENSION * TYPE LENGTH
1510 4 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1511 3 ELSE
1512 4 SET RETURN TO 3

1513 4 :IC:
1514 4 CALL XTCOM TO PROMPT FOR MAXIMUM SIZE
1515 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1516 4 ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
1517 4 SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
1518 3 :ENDIF
1519 2 :ENDIF
1520 1 EXIT XLSPS

1521 2 :PRMERR: CALL XRMSG TO DISPLAY ERROR MESSAGE
1522 2 GO TO (:PRMT1,:PRMT2,:PRMT3), RETURN
1523 1 END XLSPS
1 *01 FDS MANAGER, SCHEDULED BY FDS CONFIGURATION MANAGER. ONE PROGRAM
2 *01 PER SIGNED ON USER.
3 *01
4 ******** INPUT
5 *02 FDS STATUS TABLE (SEE SDD 6.2.2)
6 *02 FDS MANAGEMENT REQUESTS (SEE SDD 6.2.6.2)
7 1-1
8 ******** OUTPUT
9 *03 FDS MANAGER RESPONSES (SEE SDD 6.2.6.3)
10 1-1
11 ******** NOTES
12 *05 USES XMGR, XMAVA, XMPAW, XMCH, XMSGT, XDUMP
13 1-1
14 *05 RMPAR, SLIB5, SLIB0, ANR, CHUN0, XEE
15 1-1
16 *05 XMAVA IS A MANAGER GLOBAL DATA AREA CONTAINING THE AWA AND VARIOUS
17 1-1
18 *05 CONTROL VALUES
19 1-1
20 1-1
21 1-1
22 1-1
23 1-1
24 1-1
25 1-1
26 1-1
27 BEGIN XMGR
28 # BEGIN INPUTS: P1 = LU
29 # P2 = XVSTO ENTRY ADDRESS
30 # P3 = USER ID
31 # P4 = OPTIONS
32 # P5 = NUMBER OF DWA TRACKS
33 # CALL RMPAR
34 # LOCK THE MANAGER INTO THE PARTITION
35 # SET XVSTO: FDS STATUS TABLE ENTRY ADDRESS
36 # RESOLVE XVSTO ADDRESS
37 # CALL XMGRN INITIALIZ DWA FUNCTIONS
38 # IF ERROR RETURN THEN
39 # EXIT TO :XREM6
40 # ENDIF
41 # GET EXEC'S CLASS NUMBER
42 # GET PROCESSOR'S CLASS NUMBER
43 # CALL SLIB5 GET Privileged
44 # SET EXEC'S CLASS NUMBER IN SBEC
45 # SET PROCESSOR'S CLASS NUMBER IN STBPC
46 # SET CURRENT TASK IN SBAT FROM EXEC ID STBEX
47 # CALL SLIB5 ENABLE
48 # FORM EXEC'S NAME FROM 'KEY' & ASCII LU(SBBLA)
49 # SCHEDULE EXEC WITH WAIT
50 # CALL XSCHN
51 # DO UNTIL EXEC REQUESTS TERMINATION OR ABENDS(P1=9 OR -32768)
52 # IF REQUEST IS IN P1
53 # IF REQUEST IS FOR AWA MANAGEMENT (P1=1) THEN
54 # PERFORM XMGR
55 # ELSE
56 # IF REQUEST IS FOR SEQUENCE TABLE EXECUTION (P1=2) THEN
57 # PERFORM 'MXOS'
58 # ELSE
59 # IF REQUEST NOT TERMINATION (P1=9 OR -32768) THEN
60 # ISSUE MESSAGE "*** XMGR INVALID REQUEST (II) FROM XEWM"
II=REQUEST & NW IS LU

SET UP FOR FDS TERMINATION
EXIT TO :XMEND
ENDIF
ENDIF
ENDDO

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

JNB XMQQT

**********

SEQUENCE TABLE EXECUTION FROM $SEQTB

**********

INPUT

XMATA, XMPRM, XVSTA

**********

OUTPUT

XMCTP, XMPRM, XVSTA

**********

ROUTINES USED

CMUND, EXEC, XMANG, XMKIL, XMPAW (XMSCH),

XMSST, XMTFC, $LIBR, $LIBQ

**********
BEGIN XNXT
CALL XNFM TO FIND BSEG TO ENTRY
FIND ADDRESS OF BSEG
CALL XNSS TO CONVERSION TO ENDING COMMAND NUMBER INTO ENDING DISPLACEMENT
CALL XNST TO CONVERT STARTING COMMAND 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 ANY INTERFACED TABLE (WORD 3 BIT 8 IS SET) THEN
EXIT TO :ERROR: IF INTERFACE TABLE NOT SPECIFIED (WORD 4 = 0) (PARM = 1)
CALL XNFM TO SEARCH AWA FOR INTERFACE TABLE (CHAIN 4)
EXIT TO :ERROR: IF TABLE NOT FOUND (PARM = 2)
IF TABLE NOT IN AWA, THEN
CALL INKRT TO ACQUIRE FROM AWA
EXIT TO :ERROR: IF SPACE IS NOT FOUND
ENDIF
EXIT TO :ERROR: IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PARM = 1)
EXIT TO :ERROR: IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PARM = 2)
EXIT TO :ERROR: IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PARM = 3)
ENDIF
SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED
EXIT TO :ERROR: IF NOT FOUND
CALL SBLIB TO BE PRIVILEGED
SET CURRENT TASK TO THIS PROCESSOR
CALL SBLIB TO BE UN-PRIVILEGED
IF PROCESSOR REQUIRE ANY INTERFACED TABLE THEN
CALCULATE LENGTH OF INTERFACE TABLE HEADER AND SPEC
CALL EXEC TO CLASS I/O WRITE HEADER AND SPEC
ENDIF
CALL EXEC TO SCHEDULE PROCESSOR WAIT
CALL XSMKE TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XNFM TO FIND BSEG TO ENTRY
FIND ADDRESS OF BSEG
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARM = 1)
CALL XHMTO HONOR AWA REQUEST
ENDO
CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
EXIT TO :ERROR: IF PROCEDURE REQUESTED TERMINATION (PARM = 8)
EXIT TO :ERROR: IF PROCESSOR ABENDED (PARM = -32768)
IF REQUEST IS TO SET CURRENT ENTRY (PARM = 3) THEN
EXIT TO :ERROR: IF RESET NUMBER IS ZERO
CALL XNST TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY DISPLACEMENT
EXIT TO :ERROR: IF NOT FOUR TIMES
IF TERMINAL ENTRY WAS JUST EXECUTED THEN
SET UP PAMS TO SHOW RESET SEQUENCE NUMBER
PERFORM END: **NO RETURN EXPECTED**
ENDIF
ELSE
IF REQUEST IS NOT NORMAL COMPLETION (PARM = 0) THEN
DISPLAY ERROR MESSAGE - INVALID REQUEST
EXIT TO :ERROR: IF CURRENT AT IS DORMANT
CALL XSMKE TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAN TO RESCHEDULE PROCESSOR
EXIT TO :ERROR: TO TERMINATE SEQUENCE
ENDIF
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDO
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>2</td>
</tr>
<tr>
<td>SET PARM = 0 (NORMAL COMPLETION)</td>
<td></td>
</tr>
<tr>
<td>PERFORM</td>
<td>ERROR = NO RETURN EXPECTED</td>
</tr>
<tr>
<td>173</td>
<td>2</td>
</tr>
<tr>
<td>ERROR: SET PARM = 1</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>2</td>
</tr>
<tr>
<td>ERROR: SET PARM = 0 AND PAM TO APPROPRIATE REASON CODE</td>
<td></td>
</tr>
</tbody>
</table>

Reproducibility of the original page is poor.
CALLINGPROCEDURE

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 WORKS IS MAINTAINED.

EXTERNAL SYMBOLS FROM XMATA
XMBCP, XMFCP, XMFCN, XMCRE

OUTPUT (EXTERNAL SYMBOLS FROM XMATA)
XMBCP, XMFCP, XMFCN, XMCRE

*****

NOTES
USES .ENTRY
213 1 BEGIN XMAFR
214 2  SET NEW FE SIZE FIELD TO MAX(SIZE, 3)
215 3  INCREMENT TOTAL FREE SPACE BY NEW FE SIZE
216 4  INDEX TO FORWARD CHAIN POINTER (FCP) HEAD
217 5  START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)
218 6  EXIT IF 'ADDR' < FCP VALUE
219 7  SET NEW FE FCP TO CURRENT FCP VALUE
220 8  SET CURRENT FCP VALUE TO 'ADDR'
221 9  SET NEW FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE
222 10  SET NEXT FE'S BCP VALUE TO 'ADDR'
223 11  PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE
224 12  OR ELSE
225 13  INDEX TO NEXT FE FCP
226 14 END LOOP
227 15  SET NEW FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
228 16  SET CURRENT FE'S FCP VALUE TO 'ADDR'
229 17  SET NEW FE BCP TO BCP HEAD VALUE
230 18  SET BCP HEAD TO 'ADDR'
231 19  END SEARCH
232 20  IF NEW FE BCP NOT = END-OF-CHAIN
233 21  THEN
234 22  PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
235 23  END IF
236 24  IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW
237 25  (OR CONSOLIDATED) FE, THEN
238 26  MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
239 27  ENDIF
240 28  1 END XMAFR
241 29  1 BEGIN MERGE
242 30  2 IF FE 1 IS ADJACENT TO FE 2
243 31  3 THEN
244 32  4 INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
245 33  5 SET FE 1 FCP TO VALUE OF FE 2 FCP
246 34  6 IF FE 1 FCP NOT = END-OF-CHAIN
247 35  7 THEN
248 36  8 SET FE 3 BCP TO ADDRESS OF FE 1
249 37  9 ELSE
250 38  10 SET BCP HEAD TO ADDRESS OF FE 1
251 39  11 ENDIF
252 40  2 ENDIF
253 41  1 END MERGE
<table>
<thead>
<tr>
<th>Line</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>CALLING PROCEDURE</td>
</tr>
<tr>
<td>256</td>
<td>JRB XNAGT</td>
</tr>
<tr>
<td>257</td>
<td>DEF *+3</td>
</tr>
<tr>
<td>258</td>
<td>DEF OPTM</td>
</tr>
<tr>
<td>259</td>
<td>DEF SIZE</td>
</tr>
<tr>
<td>260</td>
<td>FIND A BLOCK OF FREE SPACE IN THE ALFA AT LEAST 'SIZE' WORDS LARGE</td>
</tr>
<tr>
<td>261</td>
<td>INPUT</td>
</tr>
<tr>
<td>262</td>
<td>OP TM - INDICATOR OF WHICH FREE CHAIN TO SEARCH</td>
</tr>
<tr>
<td>263</td>
<td>0 = FORWARD POINTER CHAIN (FOR TDC SPACE)</td>
</tr>
<tr>
<td>264</td>
<td>1 = BACKWARD POINTER CHAIN (FOR DATA SPACE)</td>
</tr>
<tr>
<td>265</td>
<td>SIZE - NUMBER OF WORDS NEEDED (A MINIMUM OF 3 WORDS WILL BE</td>
</tr>
<tr>
<td>266</td>
<td>ALLOCATED EVEN IF 'SIZE' IS 1 OR 2)</td>
</tr>
<tr>
<td>267</td>
<td>EXTERNAL SYMBOLS FROM XNAM</td>
</tr>
<tr>
<td>268</td>
<td>XMFCP, XMFC, XMFB</td>
</tr>
<tr>
<td>269</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>270</td>
<td>A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)</td>
</tr>
<tr>
<td>271</td>
<td>INDICATING NONE AVAILABLE</td>
</tr>
<tr>
<td>272</td>
<td>EXTERNAL SYMBOLS FROM XNAM</td>
</tr>
<tr>
<td>273</td>
<td>XMFP, XMFC, XMFB</td>
</tr>
<tr>
<td>274</td>
<td>NOTES</td>
</tr>
<tr>
<td>275</td>
<td>IF SOME FE EXACTLY 'SIZE' WORDS OR &gt;= 'SIZE+3' WORDS IS NOT FOUND AN</td>
</tr>
<tr>
<td>276</td>
<td>ERROR RETURN (A-REG = -32768) IS TAKEN</td>
</tr>
<tr>
<td>277</td>
<td>USES .EMI</td>
</tr>
</tbody>
</table>
295 1 BEGIN XNAGT
296 2 IF TOTAL FREE SPACE < MAX(SIZE,3)
297 3 THEN
298 4 CALL XNPK1 PURGE DNA ELEMENTS FROM AMA
299 5 SET PHASE1_FLAG
300 6 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
301 7 RETURN VALUE= -32768
302 8 EXIT TO :XNAGR
303 9 ENDIF
304 10 ENDIF
305 11 PERFORM YMERG(OPTY,SIZE)
306 12 IF RETURN CODE IS NOT FOUND, THEN
307 13 CALL XNAPK
308 14 IF RETURN CODE IS NOT FOUND, THEN
309 15 RETURN VALUE= -32768
310 16 EXIT TO :XNAGR
311 17 ENDIF
312 18 RETURN VALUE= AREA
313 19 :XNAGR
314 20 ENDIF
315 21 CLEAR PHASE1_FLAG
316 2 END XNAGT
1 BEGIN XMSRC
2 SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FEBHEA(OPTH)
3 START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
4 WHILE TCG 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 (OPTM = 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 0('NOT FOUND)
24 END SEARCH
25 IF TCG SPACE WAS FOUND AT THE TCG SPACE FENCE, THEN
26 INCREMENT TCG SPACE FENCE OF 'SIZE'
27 ENDIF
28 END XMSRC
347 1 *00    CALLING PROCEDURE
348 1 *00
349 1 *00 JSR XMANG
350 1 *00 DEF CLSNO
351 1 *00
352 1 **********
353 1 *01 PROVIDE AWA MANAGEMENT BASED ON REQUEST LIST (SEE FDS 5DD
354 1 *01 TABLE 6.2 - III)
355 1 *01
356 1 *01
357 1 *01 **********
358 1 *02 [INPUT
359 1 *02 CLSNO - CLASS I/O NUMBER CONTAINING REQUEST LIST
360 1 *02 REQUEST LIST (SEE TABLE 6.2 - III)
361 1 *02 **********
362 1 *03 [OUTPUT
363 1 *03 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
364 1 *03 REQUEST LIST FIELD 8
365 1 *03 **********
366 1 *05 NOTES
367 1 *05 USES EXEC, XMAF, XMAG, XMAG, XMAG
368 1 *05 XDIN, XMDAL, XDOS, XDPT
369 1 *05 NEITHER EXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
370 1 *05 FOR DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
371 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
372 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
1 BEGIN XNAG

2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER

3 CLEAR RETURN PARM

4 CLEAR REWRITE FLAG

5 INITIALIZE TO FIRST REQUEST CODE

6 DO UNTIL END OF LIST OR, EIGHT REQUESTS PROCESSED OR PARM1 > ZERO

7 IF REQUEST FOR TOC (CODE 10)

8 THEN

9 IF REQUESTED SIZE .GT. TOC SIZE, THEN

10 CALL XMAG TO COLLAPSE AWA IF TOC TOO SCATTERED FOR EXEC BUFFER SIZE

11 ENDIF

12 WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O

13 STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT

14 SET REQUESTED SIZE FROM TOC SIZE

15 SET REWRITE FLAG

16 ELSE

17 IF REQUEST TO CLEAR (CODE 17)

18 THEN

19 GET A(SDMA) FROM XMDA

20 SAVE HEADER AND DIRECTORY-SIZE

21 CLEAR XMDA THRU XMDOS

22 BUILD AN AWA FOR AWA SIZE

23 CALL XMAG TO ALLOCATE A TOC ENTRY FOR AWA

24 CHAIN IN TOC ENTRY TO XMDA

25 CALL XMAG TO ALLOCATE SPACE FOR AWA

26 SET LOCATION, SIZE, & KEY IN THE TOC

27 SET DIRECTORY ADDRESS AT XMDA

28 CLEAR THE DIRECTORY

29 SET LU, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY

30 ELSE

31 CALL XMTFN TO SEARCH TOC FOR INDICATED ENTRY

32 CASE (:VERIFY; :VERALO; :VERALO; :RENAME; :DELIVER; :DELIVER; :STORE;)

33 (:RETRIEVE; :RETRIEVE; ) REQUEST CODE

34 :VERIFY:

35 IF ENTRY NOT FOUND

36 THEN

37 SET RETURN PARM1 AND PARM2 (2 & INDEX)

38 ENDIF

39 :VERALO:

40 IF ENTRY ALREADY EXISTS

41 THEN

42 IF ALLOCATE REQUEST (3)

43 THEN

44 SET RETURN PARM1 AND PARM2 (3 & INDEX)

45 ELSE

46 IF TYPE, SIZE AND I-DIM FIELDS DO NOT MATCH

47 THEN

48 SET RETURN PARM1 AND PARM2 (4 & INDEX)

49 ENSIF

50 ELSE

51 CALL XMAG TO ALLOCATE TOC SPACE

52 IF CLASS EQ 3 OR 8, THEN

53 CHAIN IN NEW TOC ENTRY

54 SET DATA SPACE ADDRESS TO ZERO

55 ELSE

56 CALL XMAG TO ALLOCATE DATA SPACE

57 ELSE
437 8 IF SPACE NOT AVAILABLE
438 8 THEN
439 9 SET RETURN PARM1 AND PARM2 (1 & INDEX)
440 8 ELSE
441 9 IF DATA ELEMENT (CLASS 2)
442 9 THEN
443 10 IF CHARACTER STRING (TYPE 4 - 8)
444 10 THEN
445 11 INITIALIZE AREA TO BLANKS
446 11 ELSE
447 11 INITIALIZE AREA TO ZEROS
448 11 ENDIF
449 9 ENDIF
450 9 CHAIN IN NEW TOC ENTRY
451 8 ENDIF
452 8 IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE
453 8 THEN CALL XMDAL DMA ALLOCATION
454 8 ENDIF
455 7 ENDIF
456 6

457 6 :RENAME:
458 6 IF ENTRY NOT FOUND
459 7 THEN
460 7 SET RETURN PARM1 AND PARM2 (2 & INDEX)
461 6 ELSE
462 7 CALL XTFSN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION
463 7 IF ENTRY FOUND
464 7 THEN
465 8 SET RETURN PARM1 AND PARM2 (3 & INDEX)
466 8 ELSE
467 8 CALL XMAGT TO ALLOCATE NEW TOC ENTRY
468 8 IF SPACE NOT AVAILABLE
469 8 THEN
470 9 SET RETURN PARM1 AND PARM2 (1 & INDEX)
471 9 ELSE
472 9 COPY OLD ENTRY ATTRIBUTES INTO NEW ENTRY AND CHAIN IN TO TOC
473 9 DECHAIN OLD ENTRY
474 9 CALL XMAFR TO RETURN OLD ENTRY TOC SPACE TO FE POOL
475 8 ENDIF
476 7 ENDIF
477 6 ENDIF

478 6 :DELIVER:
479 6 IF ENTRY FOUND
480 7 THEN
481 7 GENERATE KEY 1 LESS THAN FOUND KEY
482 7 CALL XTFSN FOR GENERATED KEY
483 7 DECHAIN TOC ENTRY
484 7 CALL XMAFR TO RETURN TOC ENTRY SPACE TO FE POOL
485 7 CALL XMAFR TO RETURN DATA SPACE TO FE POOL
486 7 IF CLASS EQ 4 OR 6, THEN
487 8 CALL XMDDA DMA DEALLOCATION
488 7 ENDIF
489 6 ELSE
490 7 IF DELETE REQUEST (5)
491 7 THEN
492 8 SET RETURN PARM1 AND PARM2 (2 & INDEX)
493 7 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 DMA FREE CLASS NUMBER IF CLASS EQ 4 OR 6, THEN CALL XMODE DMA 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. REQUESTD-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 DMA CALL XMOVT MOVE INTO AVA IF NO SPACE THEN EXIT TO :XMREX
ENDIF
BEGIN IF WRITE VALUES TO CLASS I/O STORE TYPE IN LOW BYTE OF REQUEST WORD 1 STORE CLASS NUMBER IN REQUEST WORD ELSE SET WRITE FLAG ENDIF
ELSE WRITE TOC ENTRY TO CLASS I/O STORE CLASS NUMBER IN REQUEST WORD 1
Set WRITE FLAG
:SMRER
ENDIF
CALLING PROCEDURE

JSB XNTFM
DEF KEYS
DEF KEYS

EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'

INPUT
KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC
EXTERNAL SYMBOLS FROM XNASA
XMHD

OUTPUT

A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR
ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY
WHERE 'KEYS' COULD BE CHAINED IN

NOTES
USES .ENTR

BEGIN XNTFM
ISOLATE CLASS FROM KEY AND INDEX "O A-APPROPRIATE CHAIN HEAD
START SEARCH WHILE CHAIN POINTER (.O) < END-OF-CHAIN (-32768)
COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY
EXIT IF MATCH
RETURN ADDRESS OF ENTRY
EXIT IF 'KEYS' < TOC ENTRY (EXPECTED ENTRY NOT IN CHAIN)
RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET
OR ELSE
INDEX TO NEXT TOC ENTRY ON CHAIN
END LOOP
RETURN ADDRESS OF LAST (PREVIOUS) ENTRY WITH INDIRECT BIT SET
END SEARCH
611 1 *D0   CALLING PROCEDURE
612 1 *D0
613 1 *D0 JSB XMSST
614 1 *D0
615 1 *********
616 1 *D1   ADJUST SEQUENCE LOCATION COUNTER
617 1 *D1
618 1 *********
619 1 *D2   INPUT
620 1 *D2   A-REG - INTEGER VALUE OF SEQUENCE NUMBER TO BE LOCATED
621 1 *D2
622 1 *D2   EXTERNAL SYMBOLS
623 1 *D2
624 1 *D2   SEQAD - ADDRESS OF EXECUTING SEQUENCE TABLE IN AWA
625 1 *D2
626 1 *D2   ENDTB - DISPLACEMENT TO LAST ENTRY OF EXECUTING SEQUENCE TABLE
627 1 *********
628 1 *D3   OUTPUT
629 1 *D3   A-REG - DISPLACEMENT TO ENTRY IF SEQUENCE NUMBER IS FOUND, ELSE -32768
630 1 *D3
631 1 *********
632 1 *
633 1 *
634 1 *
635 1 *
636 1 BEGIN XMSST
637 2   SAVE RESET SEQUENCE NUMBER IN LOCAL VARIABLE SEGNO
638 2   SET DISPLACEMENT IN A-REG TO ZERO
639 2   GET ADDRESS OF FIRST ENTRY IN SEQUENCE TABLE INTO X-REG
640 2   START SEARCH UNTIL CURRENT ENTRY DISPLACEMENT > LAST ENTRY DISPLACEMENT
641 2   EXIT IF INDICATED NUMBER FOUND
642 2   OR ELSE
643 2   INCREMENT DISPLACEMENT IN A-REG TO NEXT ENTRY
644 2   END LOOP
645 3   SET A-REG TO -32768 TO INDICATE NUMBER NOT FOUND
646 3   END SEARCH
647 1 END XMSST
649 1 BEGIN XMPAW
650 2 #00 ENTRY: JSB XMPAW OR CALL XMPRM
651 2 #00 DEF RETURN ADDRESS
652 2 #02 INPUTS: XMPRM HAS REPLY
653 2 #02 AND XVS0 IS UPDATED.
654 2 #02
655 2 #03 DO UNTIL GOOD PARMS RECEIVED OR TOP AT TERMINATES
656 3 1 IF ABORT CURRENT FLAG CN(- STBLU) THEN
657 4 1 SET ABORT FLAG IN XMPRM
658 4 1 TURN OFF ABORT CURRENT FLAG
659 3 1 ENDIF
660 3 1 GET CURRENT TOP ASSOCIATED TASK(AT)
661 3 1 SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
662 3 1 JSB XMPAW
663 3 1 DEF #3 RETURN
664 3 1 DFC 0 MANAGER CALL
665 3 1 DEF XMPRM PARM FIELD
666 3 1 THIS IS AN IMPLIED WAIT
667 3 1 XMPAW 1 GET CURRENT XVS0 ENTRY (XUSTA)
668 3 1 IF THERE HAS BEEN A CALL TO PRTN (P1 FIELD IS NOT TOP AT) OR
669 4 1 IL SEGMENT IS DORMANT OR
670 4 1 ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
671 4 1 IF PARN IS NOT 0,3,8,9, OR -32768 THEN
672 5 1 SET PARM TO -32768
673 5 1 PRINT ERROR "INVALID REQUEST"
674 4 1 ELSE
675 5 1 SET PARM FIELD FROM MANAGER'S ID SEGMENT
676 5 1 ENDIF
677 4 1 ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PAM)
678 5 1 IF PARM NOT 1 OR 2 THEN
679 6 1 CALL XMKE TO ABORT TOP AT
680 6 1 PRINT ERROR "INVALID REQUEST"
681 5 1 ELSE
682 6 5 1 SET PARM FROM CURRENT ID SEGMENT
683 6 5 1 ENDIF
684 5 3 ENDIF
685 3 2 ENDDD
686 1 END XMPAW
CALLING PROCEDURE

FUNCTION

INITIALIZE TOC

& DMA FUNCTIONS

OUTPUT

BREG D=COMPLETE

MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

XMTFN, XMAGT

BEGIN XMCDIN

GET NUMBER OF DMA TRACKS FROM PS

CALL XMTFN (EDWA)

CALL XMAGT (D.C) TOC ENTRY FOR EDWA

CALL XMAGT (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 '***XMCDIN "N" TRACKS NOT AVAILABLE'

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DMA MANAGEMENT

ENDIF

END XMCDIN
1 BEGIN XMBAL
2 * DMA ALLOCATION
3 * CALLING PROCEDURE
4 * JSB XMBAL
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 * = MINUS- ERROR CONDITION
14 * NOTES
15 * CALLS XMBEX, XMBST
16 * IF (5BDA).NE. 1. THEN
17 * SET START-ADDRESS TO FIRST TRACK WORD IN DMA DIRECTORY
18 * GET SIZE(IN WORDS FROM TOC ENTRY)
19 * SIZE(SC 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 XMBX 0,BITUM,START-ADDRESS
24 * STARTBIT=BITUM
25 * CALL XMBX 1,BITUM-START-ADDRESS
26 * DMA-SIZE=BITUM-STARTBIT
27 * END-DO
28 * IF BITUM .GE. 96. THEN
29 * START-ADDRESS+START ADDRESS+1 TRACK ADDRESS
30 * ENDIF
31 * IF O:O AREA FOUND. THEN
32 * START=A:O-69 & BITUM DEFINES THE TRACK & SECTOR
33 * SET DISC ADDRESS & SIZE IN TOC ENTRY
34 * CALL XPST (TOC-ENTRY)
35 * SET NORMAL RETURN
36 ELSE
37 * ISSUE MESSAGE "***XMBEX NO DMA SPACE REMAINING"
38 * SET ERROR RETURN
39 ENDIF
40 END
BEGIN XNOST
  DATA STORE DATA
  CALLING PROCEDURE
  JSB XNOST
  FUNCTION
  UPDATE DNA ELEMENT WHICH CORRESPONDS
  TO THE ANA ELEMENT
  INPUTS
  TOC ENTRY ADDRESS OF ANA ELEMENT
  IN TREG
  OUTPUTS
  UPDATE DNA ELEMENT ON DISC TRACKS
  NOTES
  USES WRITE
  IF A(EDUA) .NE. 0, THEN
  SET DISC ADDRESS FOR THE WRITE
  SET DATA ADDRESS FOR THE WRITE
  WRITE DATA
  ENDIF
END XNOST
829  1 BEGIN XH0RT
830   2 CALL DMA RETRIEVE
831   3 JNB XH0RT
832   4
833   5 FUNCTION
834   6 RETRIEVE DMA DATA INTO AVA
835   7 INPUTS
836   8 A(TOC ENTRY) IN YREG
837   9
838  10 OUTPUTS
839  11 ADDRESS OF DATA IN THE TOC
840  12 REGISTER 0, RETRIEVE SUCCESSFUL
841  13 MINUS, ERROR NO DMA DATA
842  14
843  15 NOTES
844  16 USES XHAGT, XMODA, EXEC (READ)
845  17
846  18 IF NO DMA DIRECTORY, THEN
847  19   SET ERROR CODE -5
848  20 ELSE
849  21   CALL DMA GT DATA SPACE
850  22   IF NO SPACES, THEN
851  23     CALL DMA GT DISC ADDRESS
852  24       SET ERROR CODE -1
853  25     ELSE
854  26       CALL DMA GT DISC ADDRESS
855  27       READ DATA INTO AVA
856  28       SET RETURN CODE TO 0
857  29     ENDF1
858  30     ENDF1
859  31 ENDF1
860  32 ENDF1
BEGIN XMBST
    CALLING PROCEDURE
    JSB XMBST
    FUNCTION
    SET ALLOCATION & DEALLOCATION IN DMA DIRECTORY
    INPUTS
    DMA TOC ENTRY
    OUTPUTS
    UPDATES DMA DIRECTORY
    GET TRACK-ADDRESS FROM TOC
    SUBTRACT START OF TRACKS FROM DMA 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=DMA 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=REMAING BITS (RB)
    IF RB .IE. 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 NBLW TO REMAINDER
    ENDF
    LDADR 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)
    SHIFRT RIGHT (ARITHMETIC) BY NBFW-1 PROPAGATE BITS
    IF DATA IS NEGATIVE, THEN
    SET FILL WORD TO -1
    ELSE
    SET FILL WORD TO 0
    ENDF
    ROTATE RIGHT BY BP, RESET SAVED BITS
    STORE WHERE WORD-ADDRESS POINTS
    DO WHILE NW .GT. 0
    WORD-ADDRESS=WORD-ADDRESS+8
    SET FILL WORD INTO WHERE WORD-ADDRESS POINTS
    ENDO
    IF NBLW .GT. 0, THEN
    WORD-ADDRESS=WORD-ADDRESS+8
    LOAD DATA POINTED TO BY WORD-ADDRESS
    IF NBLW .GT. 1, THEN
    ROTATE LEFT NBLW-1 BITS
    ENDF
    EXCLUSIVE OR SIGN BIT
    IF NBLW .GT. 1, THEN
    SHIFT RIGHT ARITHMETIC BY NBLW-1
    ENDF
    STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00        CALLING PROEDURE
2 *00        JSB XMAPK
2 *00        DEF OPTION
2 *00        DEF PHASE 1 FLAG
2 *00        GENERAL COLLAPSE INTERFACE
2 *00        OUTPUT
2 *00        RETURNS AREA ADDRESS FOR AREA FOUND
2 *00        RETURNS -32768 AREA NOT FOUND
2 *00        NOTES
2 *00        USES XMPK1,XMPK2,XMPK3,XMSRC
2 *05        IF PHASE1 FLAG NOT SET, THEN
2 *05        CALL XMPK1 - PURGE DWA ELEMENTS FROM ANA
2 *05        PERFORM XMSRC(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 TDC CHAIN
2 *05        CALL XMSRC(OPTN,SIZE)
2 *05        ENDIF
2 *05        IF RETURN CODE IS NOT FOUND, THEN
2 *05        CALL XMPK3 - PACK ANA DATA AREAS
2 *05        PERFORM XMSRC(OPTN,SIZE)
2 *05        ENDIF
2 *05        ENDIF
2 *05        SET RETURN VALUE TO RETURN CODE
1 END XMAPK
1 BEGIN XMPK1
2 *DO
3 JSB XMPK1
4 *DO
5 *DO
6 *DO
7 *DO
8 *DO
9 OUTPUT
10 *DO
11 *DO
12 *DO
13 *DO
14 *DO
15 *DO
16 *DO
17 *DO
18 *DO
19 *DO
20 *DO
21 *DO
22 *DO
23 *DO
24 *DO
25 *DO
26 *DO
27 *DO
28 *DO
29 *DO
30 *DO
31 *DO
32 *DO
33 *DO
34 *DO
35 *DO
36 *DO
37 *DO
38 *DO
39 *DO
40 *DO
41 *DO
42 *DO
43 *DO
44 *DO
45 *DO
46 *DO
47 *DO
48 *DO
49 *DO
50 *DO
51 *DO
52 *DO
53 *DO
54 *DO
55 *DO
56 *DO
57 *DO
58 *DO
59 *DO
60 *DO
61 *DO
62 *DO
63 *DO
64 *DO
65 *DO
66 *DO
67 *DO
68 *DO
69 *DO
70 *DO
71 *DO
72 *DO
73 *DO
74 *DO
75 *DO
76 *DO
77 *DO
78 *DO
79 *DO
80 *DO
81 *DO
82 *DO
83 *DO
84 *DO
85 *DO
86 *DO
87 *DO
88 END XMPK1
BEGIN XMPK3
2 *00 JSB XMPK3
2 *00 CALLING PROCEDURE
2 *00
2 *00 FUNCTION
2 *00 COMPRESS AMA DATA AREA BY MIGRATING DATA AREAS
2 *00 TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
2 *00
2 *03 OUTPUT
2 *03 UPDATES XMBCP,XMFNC,XMFRE,XMP3
2 *05 NOTES
2 *05 USES XMAFR
2 *05
2 *05 INCREMENT XMP3 (NUMBER OF PHASE 3'N')
2 *05 IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
2 *05 CALL XMPK2, INSURE TOC IS COMPRESSED
2 *05 DO WHILE XMBCP .GT. XMFNC UNTIL 1 FREE AREA ADJACENT TO THE FENCE
2 *05 IF BCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
2 *05 SET HIGH-WATER TO XMFNC
2 *05 ELSE
2 *05 MULTIPLE AREAS
2 *05 SET HIGH-WATER TO BCP OF LAST-FREE
2 *05 ENDIF
2 *05 PERFORM XMPK2,(XMFNC,HIGH-WATER)
2 *05 PERFORM XMPK2,LAST-FREE
2 *05 PERFORM XMAFR(TOC-ADDRESS)
2 *05 ENDDO
2 *05 ENDIF
1 END XMPK3
1088 1 BEGIN XHANV    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 XMFRD(DATA ADDRESS,SIZE)
1096 2 GET FREE AREA FROM XMBCP
1097 2 DECREASE FREE AREA LENGTH BY 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 XHANV
FORTRAN CALLING PROCEDURE

1*4
11  CALL XPATH (LU, INBUF, INLEN, NBUF, IMUN, NAME, TYPE, SIZE,
12  IDIM, DSPT)
11  *********
13  XPATH ALLOWS PROCESSORS TO OBTAIN THE ATTRIBUTES OF THE
14  PARAMETER REFERENCED BY THE INTERFACE TABLE.
11  *********
15  INPUT
16  LU - LOGICAL UNIT NUMBER OF USER TERMINAL
17  INBUF - INPUT/OUTPUT BUFFER OF 7*(8 PARAMETERS + 1) WORDS,
18  ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
19  INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ON
20  FIRST USE TO CAUSE INITIALIZATION.
21  INLEN - LENGTH OF INBUF
22  NBUF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
23  WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
24  THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS.
25  IMUN - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE WHOSE
26  ATTRIBUTES ARE REQUESTED.
11  *********
27  OUTPUT
28  NAME - AN ALPHANUMERIC NAME OF UP TO SIX CHARACTERS WHICH
29  IDENTIFIES THE DATA ELEMENT OR DDE WHERE THE DATA IS TO
30  BE OBTAINED/STORED. A ZERO ENTRY INDICATES INPUT DATA
31  WHICH IS LITERAL DATA STORED WITHIN THE INTERFACE TABLE.
32  TYPE - DATA TYPE CODE OF THE PARAMETER
33  SIZE - TOTAL NUMBER OF WORDS OF LITERAL DATA,
34  TOTAL NUMBER OF WORDS OF REFERENCED INPUT DDE,
35  TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DDE IF
36  SUBSCRIBED OR ZERO IF NOT SUBSCRIBED,
37  TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DDE OR
38  ZERO IF OUTPUT DDE
39  IDIM - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
40  AN INPUT DDE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
41  IF AN UNSUBSCRIBED OUTPUT
42  DSPT - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
43  SUBSCRIBED DATA ELEMENTS ELSE ZERO.
44  FOR AN INPUT DDE THE RTE FILE MANAGER TYPE CODE IS
45  RETURNED IN DSPT.
11  *********
46  EXTERNAL SYMBOLS
47  NOTE (SEE XPGT)
49  *********
50  INTERNAL VARIABLES
51  NOTES
53  *********
54  USES .XENT, XPTR, XPGT, XPI, XPM, XPR, 
55  APLSAME(XPGT), XPATH, ZXSTB
57  XPATH IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
58  POINTS XPATH, XPGT AND XPUT
FORTRAN CALLING PROCEDURE

CALL XFGET (LU, INBUF, INTHG, HRBUF, N, INUMS, IN(1), ..., IN(N))

**XGET**

**INPUT**

1. LU - LOGICAL UNIT NUMBER OF USER TERMINAL
2. INTHG - INPUT/OUTPUT BUFFER OF INTEGER PARAMETERS (10) WORDS.
3. INBUF - ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON FIRST USE TO CAUSE INITIALIZATION.
4. IN - LENGTH OF INTHG
5. HRBUF - MANAGER REQUEST BUFFER (6 WORDS) USED TO COMMUNICATE WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCAN TRAIL AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
6. N - NUMBER OF PARAMETERS TO BE RETRIEVED. IF N=0, ALL PARAMETERS ARE REQUESTED AND INUMS IS NOT USED.
7. INUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE INTERFACE TABLE TO BE RETRIEVED. THE ORDER OF THE NUMBERS INDICATES THE STORAGE ORDER FOR THE IN(I)

**SHARED EXTERNAL SYMBOL**

1. XGPF - FLAG INDICATING XFGET/XIPUT NORMAL PROCESSING (-1) OR XGPF/XIPU SPECIAL PROCESSING BY-PASSING SUBSCRIPT RESOLUTION (0-1)

**OUTPUT**

1. IN(I) - LABELS OR VARIABLE NAMES WHERE INPUT DATA IS TO BE STORED.

**INTERNAL VARIABLES**

1. AMUL - ARRAY OF 10 BYTES CONTAINING MULTIPLIERS FOR USE BY THE PROCEDURE XGKPX IN DETERMINING WORD DISPLACEMENTS FOR VARIOUS TYPES OF SUBSCRIPTED ARRAYS
2. CLSFL - ADDRESS OF THE CLASS FIELD (SECOND WORD) OF THE REQUEST ARRAY REQUEST
3. STPT - FLAG INDICATING GET OR PUT PROCESSING. VALUES CORRESPOND TO BIT SETTINGS FOR INPUT OR OUTPUT FLAGS IN THE FIFTH WORD OF INTERFACE TABLE ENTRIES
4. I - COUNTER OF THE NUMBER OF PASSES THROUGH THE PARAMETER.
150 1 04 PROCESSING LOOP
151 1 04 MP  = VALUE OF "P", WHERE P IS THE NUMBER OF PARAMETERS IN THE
152 1 04 INTERFACE TABLE
153 1 04 NMPFL  = ADDRESSES OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
154 1 04 NARY  = ARRAY OF REQUEST SEQUENCE
155 1 04 NARY  = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
156 1 04 REQT  = EIGHT WORD ARRAY USED FOR CONSTRUCTING AMD MANAGEMENT
157 1 04 USERID  = USER FILE IDENTIFIER CHARACTER (=IPARM(3))
158 1 04 XECFL  = EXTERNAL VARIABLES (SEE XPREQ)
159 1 04 XPLUS
160 1 04 XPPRD
161 1 04 NOTES
162 1 05 USES .ENTRY, EXEC, XPREQ, XPRET, XUSTB
163 1 05 XGET
164 1 05 XGET MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.
165 1 05 XGETMINT Must be used by XGET, XPUT, AND XPATH AND XEOS
166 1 05 TO BE INITIALIZED ONLY ONCE BY ANY OF THE TIME: ROUTINE:
167 1 05 SINCE REQUESTS FOR Input DATA FROM THE AWA MAY BE MADE FOR UP TO
168 1 05 EIGHT AT A TIME, XGET RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
169 1 05 REQUESTED IN MULTIPLES OF EIGHT.
FORTRAN CALLING PROCEDURE

CALL XPPUT (LU, INTBUF, INTLNG, NRBUFF, N, INUMS, OUT(1), ..., OUT(N))

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

XPPUT ALLOWS PROCESSORS TO STORE DATA INTO DATA ELEMENTS AND
OBEYS REFERENCED IN THE INTERFACE TABLE AS FOLLOWS:
  1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC.
  2) NAME IS SEARCHED FOR IN THE TOC. IF THE NAME IS
   NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING
   TERMINATES.
   2) NAME IS SEARCHED FOR IN THE TOC. IF THE NAME IS
   NOT FOUND, IT IS ADDED TO THE TOC. IF THE NAME IS
   FOUND OR AFTER BEING ADDED TO THE TOC, THE NAME IS
   PREFixed WITH A / SYMBOL, SUFFixed WITH A ONE

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

INPUT - LOGICAL UNIT NUMBER OF USER TERMINAL
  INTBUF - INPUT/OUTPUT BUFFER OF 7(# PARAMETERS + 1) WORDS,
  INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
  FIRST USE TO CAUSE INITIALIZATION.
  INTLNG - LENGTH OF INTBUF
  NRBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
  WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH
  AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
  N - NUMBER OF PARAMETERS TO BE STORED. IF N=0, ALL
  OUTPUT PARAMETERS ARE ASSUMED TO BE SUPPLIED.
  INUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
  INTERFACE TABLE REFERENCING DATA ELEMENTS AND OBEDE'S
  WHERE THE OUTPUT DATA IS TO BE STORED. THE ORDER OF THE
  NUMBERS INDICATES THE STORAGE ORDER FROM OUT(1) ... OUT(N)
  OUT(N) - LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE
  STORED FROM.
  XPGF - FLAG INDICATING XPPUT/XPPUT NORMAL PROCESSING (-1) OR
  XPG1/XPG1 SPECIAL PROCESSING BY PASSING SUBSCRIPT
  RESOLUTION (3-9)
  ************

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

OUTPUT
  NONE
  ************

INTERNAL VARIABLES
  (SEE XPGF)
  ************

MOVES
  (SEE XPGF)
  ************
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 XRPRD TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRDE
CALL XRPRD 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 XRPRD TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS DRDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XRPRD TO QUEUE DELETION
BUILD R/F ST TO REALLOCATE DRDE WITH NEW ATTRIBUTES
EXIT TO :ERPC**_; IF FILE TYPE NOT 1-13, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
CALL XRPRD TO QUEUE REALLOCATION OF DRDE
ENDIF
ENDIF
ENDDO
CALL XPRG To COMPLETE QUEUED REQUESTS
EXIT ACCESS

:ERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:ERR13: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:ERR14: CALL XPRQ TO PURGE QUEUED REQUESTS
:ERPC**_; TERMINATE PROCESSOR FOR INVALID DRDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>379</td>
<td>1 *00 FORTRAN CALLING PROCEDURE</td>
</tr>
<tr>
<td>380</td>
<td>1 *00</td>
</tr>
<tr>
<td>381</td>
<td>1 *00 CALL XPGT1 (LU, INBUF, INTLNG, MBUF, INUM, IN, SIZE, DISP)</td>
</tr>
<tr>
<td>382</td>
<td>1 *00</td>
</tr>
<tr>
<td>383</td>
<td>1 *00 **********</td>
</tr>
<tr>
<td>384</td>
<td>1 *01 XPGLT ALLOWS TV'S CALLING PROGRAM TO RETRIEVE A SPECIFIED AMOUNT OF DATA FROM AN INPUT PARAMETER BEGINNING AT ANY POINT IN THE</td>
</tr>
<tr>
<td>385</td>
<td>1 *01 PARAMETER ARRAY</td>
</tr>
<tr>
<td>386</td>
<td>1 *01</td>
</tr>
<tr>
<td>387</td>
<td>1 *01 INPUT</td>
</tr>
<tr>
<td>388</td>
<td>1 *02 LU - LOGICAL UNIT NUMBER OF USER TERMINAL</td>
</tr>
<tr>
<td>389</td>
<td>1 <em>02 INTBU - INPUT/OUTPUT BUFFER OF 7</em>(# PARAMETERS + 1) WORDS,</td>
</tr>
<tr>
<td>390</td>
<td>1 *02 ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE</td>
</tr>
<tr>
<td>391</td>
<td>1 *02 INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON</td>
</tr>
<tr>
<td>392</td>
<td>1 *02 FIRST USE TO CAUSE INITIALIZATION</td>
</tr>
<tr>
<td>393</td>
<td>1 *02 INTLNG - LENGTH OF INTBU</td>
</tr>
<tr>
<td>394</td>
<td>1 *02 MREF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE</td>
</tr>
<tr>
<td>395</td>
<td>1 *02 WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY</td>
</tr>
<tr>
<td>396</td>
<td>1 *02 THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS</td>
</tr>
<tr>
<td>397</td>
<td>1 *02 INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM</td>
</tr>
<tr>
<td>398</td>
<td>1 *02 WHICH TO RETURN DATA</td>
</tr>
<tr>
<td>399</td>
<td>1 *02 SIZE - TOTAL NUMBER OF WORDS TO BE RETURNED</td>
</tr>
<tr>
<td>400</td>
<td>1 *02 DISP - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER</td>
</tr>
<tr>
<td>401</td>
<td>1 *02 AT WHICH TO BEGIN DATA RETRIEVAL. A VALUE OF ZERO</td>
</tr>
<tr>
<td>402</td>
<td>1 *02 INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER</td>
</tr>
<tr>
<td>403</td>
<td>1 *02 IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT</td>
</tr>
<tr>
<td>404</td>
<td>1 *02 BE SPECIFIED, I.E., RETRIEVAL FROM AN AREA PREVIOUS TO</td>
</tr>
<tr>
<td>405</td>
<td>1 *02 THE USERS SUBSCRIPTS IS NOT SUPPORTED</td>
</tr>
<tr>
<td>406</td>
<td>1 *02 **********</td>
</tr>
<tr>
<td>407</td>
<td>1 *03 INPUT</td>
</tr>
<tr>
<td>408</td>
<td>1 *03 IN - ARRAY OF AT LEAST SIZE WORDS INTO WHICH INPUT DATA IS TO</td>
</tr>
<tr>
<td>409</td>
<td>1 *03 BE STORED</td>
</tr>
<tr>
<td>410</td>
<td>1 *03 **********</td>
</tr>
<tr>
<td>411</td>
<td>1 *04 INTERNAL</td>
</tr>
<tr>
<td>412</td>
<td>1 *04 BSDisp - DISPLACEMENT FROM THE REAL ORIGIN OF THE DATA ELEMENT AS</td>
</tr>
<tr>
<td>413</td>
<td>1 *04 A RESULT OF USER SUBSCRIPTING. DISP IS ADDED TO THIS</td>
</tr>
<tr>
<td>414</td>
<td>1 *04 VALUE IN THE REQUEST TO THE MANAGER</td>
</tr>
<tr>
<td>415</td>
<td>1 *04 GTP - DISPLACEMENT INTO JTP DEPENDING ON WHETHER GET (D) OR</td>
</tr>
<tr>
<td>416</td>
<td>1 *04 PUT (I) PROCESSING IS INVOLVED</td>
</tr>
<tr>
<td>417</td>
<td>1 *04 JGTP - TWO WORD VECTOR CONTAINING 'JSB XPGT1' OR 'JSB XPGT1'</td>
</tr>
<tr>
<td>418</td>
<td>1 *04 INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGT1 AND</td>
</tr>
<tr>
<td>419</td>
<td>1 *04 XPGT1 CALLS</td>
</tr>
<tr>
<td>420</td>
<td>1 *04 **********</td>
</tr>
<tr>
<td>421</td>
<td>1 *05 NOTES</td>
</tr>
<tr>
<td>422</td>
<td>1 *05 USES EXE, XPATH, XIE13(XPATH), XPGT(XPATH), XPRX(XPATH)</td>
</tr>
<tr>
<td>423</td>
<td>1 *05 **********</td>
</tr>
</tbody>
</table>
FORTRAN CALLING PROCEDURE

CALL XPTII (LU, INBUF, INTLG, MRBUF, INUM, OUT, SIZE, DISP)

**********

XPPTII ALLOWS THE CALLING PROGRAM TO STORE A SPECIFIED AMOUNT OF
DATA INTO AN OUTPUT PARAMETER BEGINNING AT ANY POINT IN THE
PARAMETER ARRAY

**********

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INBUF - INPUT/OUTPUT BUFFER OF 7* (# 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

INTLG - LENGTH OF INBUF

MRBUF - 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

INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE INTO
WHICH TO STORE DATA

OUT - ARRAY OF AT LEAST SIZE WORDS FROM WHICH OUTPUT DATA IS TO
BE TAKEN

SIZE - TOTAL NUMBER OF WORDS TO BE STORED

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.

**********

OUTPUT

NONE

**********

INTERNAL

SEE XPTII

**********

NOTE

XPPTII IS AN ENTRY POINT INTO XPTII

**********
BEGIN XPGET
SET FOR GET PROCESSING
PERFORM XPXXI TO RETRIEVE DATA
END XPGET
BEGIN XPPTI
SET FOR PUT PROCESSING
PERFORM XPXXI TO STORE DATA
END XPPTI
BEGIN XPXII
SET XPSF (XPSET) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
CALL XPATH TO ASSURE INITIALIZATION OF INBUF AND RETURN BASE DISPLACEMENT
EXIT TO :XPE13: (XPATH) IF INDICATED PARAMETER IS A DORE FILE
SAVE INTERFACE TABLE ENTRY CLASS/TYPE WORD, FLAG/DISP WORD AND SIZE WORD
IF LITERAL (NAME = 0)
THEN
EXIT TO :ERROR5: IF DISP + SIZE > SAVED SIZE
RESET BASE DISPLACEMENT TO ACTUAL DISP FIELD IN INTERFACE TABLE ENTRY
ELSE
INCREMENT BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
SET TYPE FIELD TO FREE
ENDIF
CLEAR ENTRY SUBSCRIPT BIT
SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
SET SIZE FIELD TO SIZE
CALL XPGET/XPPUT TO TRANSFER DATA
RESTORE ORIGINAL INTERFACE TABLE ENTRY
RESTORE XFGPF (XPGET) TO NOMINAL VALUE
EXIT XPXXI
:ERROR5: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
END XPXXI
CALLING PROCEDURE

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

DEF OPTN REQUIRED OPTION

DEF REQST REQUIRED REQUEST

DEF ADRES OPTIONAL ADDRESS (SEE OPTN)

**********

XPGET/XPPUT BUFFERED AXA MANAGEMENT SERVICE

*******

IM-UT

*********

OPTN - OPTION WORD

SIGN BIT - 0 QUEUE REQUEST AND RETURN

1 QUEUE REQUEST AND CLOSE BUFFER

RIGHT BIT - 0 QUEUE REQUEST AND TRANSFER DATA TO/FROM

ADRES

1 QUEUE REQUEST ONLY

REQST - REQUEST (SEE SDD 6.2.6.2)

ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT

VALUES (USED ONLY WHEN OPTN(RIGHT BIT) = 0)

**********

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

AADS - ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDS)

ADDS - TABLE OF ADDRESSES FOR STORAGE OF Fetched DATA

APRNS - ADDRESS OF PARNS AREA

CLASS - CLASS NUMBER USED TO TRANSMIT DATA TO MANAGER

PARNS - PARAMETER RETURN AND SCRATCH AREA

PTR - SAVE AREA FOR ADDS AND XPROB POINTER

*******

NOTES

USES _ENTR, EXEC, XPAR, XPAR

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

**********
1 BEGIN XPREQ
2 MOVE REQUEST INTO BUFFER
3 IF OPTION IS TO TRANSFER DATA
4 THEN
5 IF REQUEST IS TO RETRIEVE DATA (8)
6 THEN
7 STORE ADDRESS IN TABLE
8 ELSE SHOULD BE A REQUEST TO STORE DATA (7)
9 OUTPUT DATA TO CLASS I/O
10 STORE CLASS NUMBER IN REQUEST WORD 8
11 ENDIF
12 ENDIF
13 INCREMENT POINTER
14 IF BUFFER FULL OR OPTION IS TO CLOSE NON-EMPTY BUFFER
15 THEN
16 CLOSE BUFFER
17 TRANSMIT BUFFER TO MANAGER
18 IF MANAGER WITH REQUEST FOR AWI MANAGEMENT
19 RETRIEVE RETURN PARAMETERS
20 IF REWRITE FLAG SET (PARMS)
21 THEN
22 RETRIEVE REQUEST BUFFER
23 ENDIF
24 IF REQUESTS WERE SUCCESSFUL
25 THEN
26 DO FOR EACH REQUEST IN BUFFER
27 IF REQUEST TO RETURN DATA (8, 9 OR 10)
28 THEN
29 RETRIEVE AND STORE DATA IN ADDRESS CONTAINED IN TABLE
30 ENDIF
31 ENDDO
32 CLEAR POINTER AND LOCAL CLASS NUMBER
33 ELSE
34 OUTPUT FAILURE MESSAGE (XP11)
35 DO FOR EACH REQUEST IN BUFFER
36 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
37 THEN
38 FREE CLASS I/O NUMBER AND SAM BUFFER
39 ENDIF
40 ENDDO
41 ENDDO
42 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
43 ENDF
44 ENDF
45 END XPREQ
FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE

CALL XPDRS (LU,PRMLEN,PROMPT,TYPE,DALE.IND.IDIM,DATA,RETC)

CALLS XPRM, DECODES COMMUNICATIONS BUFFER LOOKING FOR A SPECIFIC "TYPE" AND STORES IT IN "DATA" FOR "DALE." LOGICAL ELEMENTS

INPUTS FROM CALLING SEQUENCE:

LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER STRING USED FOR THE USER PROMPT
PROMPT - (INTEGER, PRMLEN 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 5
6 - CHARACTER STRING LENGTH 8
7 - CHARACTER STRING LENGTH 18
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 A CR, THERE IS NO RESPONSE.
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.

INTERNAL VARIABLES
DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT
ELEMENT IS TO BE STORED.
EFTYP - THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER
STRING RESPONDER. IF TYPE IS PRO, IT IS THE
CLOSEST SUPPORTED LENGTH, ELSE IT IS THE LENGTH
SPECIFIED BY TYPE.
IND - CONTAINS THE INDEX INTO MSGS ARRAY OF THE CURRENT
ERROR MESSAGE.
LEN - LENGTH OF CURRENT ERROR MESSAGE.
LENGTH - 9 WORD ARRAY REPRESENTING THE LENGTH IN WORDS
OF THE 9 DATA TYPES 0-8 RESPECTIVELY.
STKIND - COUNT OF NUMBER OF NESTED REPEATS AND INDEX
TO THE CURRENT TOP OF THE STACK.
STKREP - 9 WORD ARRAY FOR STACK OF REPEAT COUNTS.
STKTOK - 4 WORD ARR. FOR STACK OF TOKEN POINTERS.
WHERE EACH REPEAT GROUP BEGINS.
TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED.
TYPE1 - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE
TYPET = TYPE+1.
SUBROUTINES AND FUNCTIONS CALLED:
EXEC, XPRDS, ERRMSG.
POL ROUTINES INCLUDED:
XPRDS, STRING, SUBSCR, REPET, ERRMSG.
1 BEGIN XPRODS
2 IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3 CALL XPRMD TO PROMPT USER AND COLLECT RESPONSE
4 IF XPRMD RETURN CODE IS ZERO THEN
5 DO UNTIL EOS TOKEN IS FOUND
6 SET MESSAGE NUMBER TO XPO2
7 CASE TOKEN (<NUMBER>,<NUMBER>,<NUMBER>,<STRING>,<NULL>,
8 ,<SUBSCR>,<REPET>):
9 :NUMBER:
10 ERREXIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERRMSG
11 :MESSAGE NUMBER TO XPO2
12 ERREXIT IF THERE IS NO ROOM IN DATA AREA PERFORM ERRMSG
13 SET PREVIOUS TOKEN TO DATA
14 CALL XRMOV TO MOVE DATA INTO DATA AREA
15 INCREMENT POINTER IN DATA AREA
16 INCREMENT POINTER TO NEXT TOKEN
17 :STRING:
18 PERFORM STRING
19 :SUBSCR:
20 PERFORM SUBSCR
21 :REPET:
22 PERFORM REPET
23 ENCASE
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 FLAG =0) OR
27 (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG =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 ENDIF
36 ENDIF
37 ENDOO
38 ENDOO
39 ENDIF
40 ENDXPRODS
41 ENDXPRODS
1 BEGIN STRING
2 ERREXIT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERRMSG
3 DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
4 ERREXIT IF TYPE OF RESPONSE > TYPE REQUESTED AND
5 ERREXIT IF TYPE IS NOT FREE PERFORM ERRMSG
6 IF TYPE IS NOT FREE THEN
7 SET EFFECTIVE LENGTH = LENGTH REQUESTED
8 END IF
9 SET MESSAGE NUMBER TO XPOS
10 ERREXIT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERRMSG
11 CALL XMOV TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH
12 CALL XMOV TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH
13 SET PREVIOUS TOKEN IS DATA
14 SET MESSAGE NUMBER TO XPOS
15 INCREMENT POINTER IN DATA AREA
16 INCREMENT TO NEXT TOKEN
17 1 END STRING
18 1 *
19 1 *
20 BEGIN SUBSCR
21 INCREMENT POINTER TO NEXT TOKEN
22 SET MESSAGE NUMBER TO XPOS
23 ERREXIT IF TOKEN IS NOT AN INTEGER TO PERFORM ERRMSG
24 IF I-DIMENSION > 1 THEN
25 SFT I TO INTEGER VALUE
26 INCREMENT POINTER TO NEXT TOKEN
27 ERREXIT IF TOKEN IS NOT AN INTEGER OR
28 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
29 SET MESSAGE NUMBER TO XPOS
30 ERREXIT IF INTEGER > I-DIMENSION TO PERFORM ERRMSG
31 ERREXIT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERRMSG
32 ELSE
33 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
34 SET MESSAGE NUMBER TO XPOS
35 ERREXIT IF SUBSCRIPT IS OUT OF RANGE
36 END IF
37 ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT
38 INCREMENT POINTER BY 3 TOKENS
39 SET PREVIOUS TOKEN = SUBSCRIPT
40 SET MESSAGE NUMBER TO XPOS
41 ERREXIT IF TOKEN IS AN EOS OR
42 ERREXIT IF TOKEN IS A REPEAT OR
43 ERREXIT IF TOKEN IS A CLOSE PAREN OR
44 ERREXIT IF TOKEN IS A SUBSCRIPT TO PERFORM ERRMSG
45 END SUBSCR
46 1 END STRING
CD*********

822 | CD00
823 | CD01
824 | CD02
825 | CD03
826 | CD04
827 | CD05
828 | CD06
829 | CD07
830 | CD08
831 | CD09
832 | CD10
833 | CD11
834 | CD12
835 | CD13
836 | CD14
837 | CD15
838 | CD16
839 | CD17
840 | CD18
841 | CD19
842 | CD20
843 | CD21
844 | CD22
845 | CD23
846 | CD24
847 | CD25
848 | CD26
849 | CD27
850 | CD28
851 | CD29
852 | CD30
853 | CD31
854 | CD32
855 | CD33
856 | CD34
857 | CD35
858 | CD36
859 | CD37
860 | CD38
861 | CD39
862 | CD40
863 | CD41
864 | CD42
865 | CD43
866 | CD44
867 | CD45
868 | CD46
869 | CD47
870 | CD48
871 | CD49
872 | CD50
873 | CD51
874 | CD52
875 | CD53
876 | CD54
877 | CD55
878 | CD56
879 | CD57
880 | CD58

822 | FORTRAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE
823 | CALL XPRM (LU,PRMLEN,PROMPT,COMLEN,COMBUF,RETC)
824 | WRITES "PRMLEN" WORDS OF "PROMPT" TO USER READS THE USER'S
825 | RESPONSE; CONVERTS RESPONSE TO TOKENS IN "COMBU" AND PASSES
826 | BACK A RETURN CODE "RETC" INCIDATING THE SUCCESS OF XPRM
827 | AND THE USEFULLNESS OF "COMBUF".

836 | INPUTS FROM CALLING SEQUENCE:
837 | LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
838 | PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
839 | STRING USED FOR THE USER PROMPT!
840 | PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
841 | USED AS THE USER PROMPT.
842 | COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE
843 | COMMUNICATIONS BUFFER (COMBUF)
844 | COMBUF - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE
845 | ENCODED USER RESPONSE
846 | RETC - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO THE
847 | CALLING PROGRAM AS FOLLOWS:
848 | 0 - NORMAL RETURN BUFFER CONTAINS USERS RESPONSE
849 | 1 - USER ENTERED INTEGER CONTENTS OF BUFFER UNPREDICTABLE.
850 | 2 - USER ENTERED CHAR. BUFFER CONTAINS NO INFORMATION
851 | 3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
852 | 34 CHARACTERS OR 17 WORDS.

861 | INTERNAL VARIABLES
862 | COUN - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN
863 | A CHARACTER STRING AND NUMBER DIGITS IN A NUMBER.
864 | DLINT - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE
865 | DLVD - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE
866 | VALUE
867 | FLCOM - CORRANS FLAG
868 | Q - LAST CHARACTER NOT A COMMA
869 | T - LAST CHARACTER WAS A COMMA
870 | FLCOM - CONTINUE FLAG
871 | Q - THIS IS NOT A RESPONSE TO A CONTINUE
872 | T - THIS IS A CONTINUED RESPONSE
873 | FLEN - NUMBER FLAG
874 | Q - POSITIVE NUMBER
875 | T - NEGATIVE NUMBER
876 | FLGPOW - POWER FLAG
877 | Q - POSITIVE POWER
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       COMMENT:
9       CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
10      CALL EXEC TO READ RESPONSE
11      CALL XRPK TO CONVERT FROM RESPONSE TO RA FORMATTED
12      IF NUMBER OF WORDS READ IS NOT ZERO THEN
13          SET COMMA FLAG ON
14          DO WHILE CONTINUE FLAG IS OFF
15              IF INPUT BUFFER IS COMPLETELY SCANNED THEN
16                  SET CONTINUE FLAG ON
17                  IF COMMA FLAG IS ON THEN
18                      CALL EXEC TO WRITE CONTINUE
19      GO TO :COMPLAP:
20      ELSE
21          EXIT IF COMUF IS FULL PERFORM COMUF
22          IF INPUT CHARACTER IS A COMMA THEN
23              EXIT IF COMUF CANNOT HOLD TOKEN PERFORM COMUF
24              STORE NULL FIELD TOKEN IN COMUF
25          INCREMENT WORDS IN COMUF BY 1
26          INCREMENT TOKENS IN COMUF BY 1
27          ENDIF
28      ENDIF
29      ENDIF
30      SET COMA FLAG ON
31      SET NEXT INPUT CHARACTER
32      ELSE
33          SET COMA FLAG OFF
34          PERFORM TOKENS
35          ENDIF
36          ENDIF
37      ELSE
38          IF CONTINUE FLAG IS OFF THEN
39              SET XPRDM RETURN CODE TO SAY USER ENTERED CR
40          ENDIF
41          ENDIF
42      ELSE
43          SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
44      ENDIF
45      ENDIF
46      END XPRDM
BEGIN TOKENS
SET NEGATIVE NUMBER FLAG OFF
SET NEGATIVE POWER FLAG OFF
SET POWER = 0
IF INPUT CHARACTER IS A DIGIT THEN
   PERFORM DIGIT
ELSE
   CASE J (:A:,:B:,:C:,:D:,:E:,:G:,:I:)

   :A:
   SET XTPRN RETURN CODE TO SAY I ENTERED

   :B:
   PERFORM QUOTE

   :C:
   SET PAREN INDICATOR = 0
   GO TO :E:

   :D:
   SET PAREN INDICATOR = 1
   GO TO :E:

   :E:
   ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN TO PERFORM COMFUL
   SINGLE TOKEN (OPAR + PAREN INDICATOR) IN COMBUF
   INCREMENT #WORDS IN COMBUF BY 1
   INCREMENT TOKENS IN COMBUF BY 1
   GET NEXT CHARACTER

   :F:
   ERREXIT IF NEXT CHARACTER IS NOT A DIGIT PERFORM INVAL

   :G:
   IF INPUT CHARACTER IS A - THEN
   SET NEGATIVE NUMBER FLAG ON
   ENDF

   :H:
   IF NEXT CHARACTER

   :I:
   IF INPUT CHARACTER IS A DIGIT THEN
   PERFORM DIGIT
   ELSE
   IF INPUT CHARACTER IS A . THEN
   GO TO :F:
   ELSE
   PERFORM INVAL - NO RETURN
   ENDF

   ENDIF

   ENDF

END CASE
ENDIF
END BEGIN TOKENS
1 BEGIN QUOTE
2 GET NEXT CHARACTER
3 SET #CHARACTERS + 0
4 DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
5 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
6 (INPUT CHARACTER IS A QUOTE AND
7 NEXT CHARACTER IS A QUOTE AND
8 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
9 INCREMENT #CHARACTERS BY 1
10 MOVE CHARACTER INTO TEMPORARY BUFFER (#CHARACTERS)
11 IF INPUT CHARACTER IS A QUOTE THEN
12 GET NEXT CHARACTER
13 ENDIF
14 GET NEXT CHARACTER
15 ENDDO
16 ERROR IF LENGTH OF CHARACTER STRING IS 0 OR
17 ERROR IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
18 ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
19 STORE CHARACTER STRING TOKEN IN COMBUF
20 STORE NUMBER OF CHARACTERS IN COMBUF
21 CALL TSPCK TO CONVERT CHARACTERS FROM R1 TO A2 FORMAT
22 INCREMENT #WORDS IN COMBUF BY 2*([#CHARACTERS + 1] / 2)
23 INCREMENT #TOKENS IN COMBUF BY 1
24 GET NEXT CHARACTER
25 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 ERERIT IF THERE IS NO ROOM IN CONBUF FOR THIS TOKEN PERFORM COMFUL
1030 12 ELSE IF INTEGER IS ZERO PERFORM INVAL
1031 13 STORE REPEAT TOKEN IN CONBUF
1032 14 INCREMENT WORDS IN CONBUF BY 2
1033 15 INCREMENT TOKENS BY 1
1034 16 GET NEXT CHARACTER
1035 17 ELSE
1036 18 PERFORM INTEGER
1037 19 ENDIF
1038 20 ENDIF
1039 21 ELSE
1040 22 PERFORM INTEGER
1041 23 ENDIF
1042 24 ENDIF
1043 25 END DIGIT
1044 26 *
1045 27 *
1046 28 BEGIN DCOL
1047 29 SET INTEGER=0
1048 30 SET COUNTER=0
1049 31 WHILE INPUT BUFFER IS NOT EXHAUSTED
1050 32 IF CHARACTER IS A DIGIT AND
1051 33 WHILE CHARACTER IS A DIGIT AND
1052 34 SET INTEGER* (INTEGER + 10) + INPUT CHARACTER + 48
1053 35 IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1054 36 INCREMENT COUNTER BY 1
1055 37 GET NEXT CHARACTER
1056 38 ENDIF
1057 39 END DCOL
1059 1 BEGIN DECPT
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 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVERFLOW
1067 3 ENDF
1068 3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069 4 PERFORM EORD
1070 3 ELSE
1071 4 PERFORM REAL
1072 3 ENDF
1073 2 ELSE
1074 3 PERFORM REAL
1075 2 ENDF
1076 1 END DECPT
1 BEGIN END IF CHARACTER IS "E" THEN
2 SET TYPE FLAG TO "E"
3 ELSE IF TYPE FLAG IS "E" THEN
4 GET NEXT CHARACTER
5 IF CHARACTER IS A THEN
6 SET NEGATIVE POWER FLAG ON
7 ELSE IF CHARACTER IS NOT A THEN
8 EXCEPT IF INPUT BUFFER IS NOT A DIGIT PERFORM INVAIL
9 ELSE IF TYPE FLAG IS "E" THEN
10 IF TYPE FLAG IS "E" THEN
11 PERFORM REAL
12 ELSE PERFORM REAL IF NEGATIVE POWER = "POWER"
13 ENDIF
14 ENDIF
15 ENDIF
16 ENDIF
17 ENDIF
18 END
121 1 BEGIN INTEGER
122 2 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
123 2 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
124 2 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVALID
125 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
126 2 CONVERT DOUBLE TO INTEGER
127 2 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
128 2 IF NEGATIVE NUMBER FLAG IS ON THEN
129 3 SET INTEGER = -INTEGER
130 2 ENDFIF
131 2 STORE INTEGER TOKEN IN COMBUF
132 2 INCREMENT #WORDS IN COMBUF BY 2
133 2 INCREMENT #TOKENS IN COMBUF BY 1
134 1 END INTEGER
135 1 *
136 1 *
137 1 *
138 1 BEGIN REAL
139 2 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
140 2 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
141 2 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVALID
142 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
143 2 SET REAL = DOUBLE x 10 ** POWER
144 2 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
145 2 IF NEGATIVE NUMBER FLAG IS ON THEN
146 3 SET REAL = -REAL
147 2 ENDFIF
148 2 STORE TOKEN IN COMBUF
149 2 INCREMENT #WORDS IN COMBUF BY 3
150 2 INCREMENT #TOKENS IN COMBUF BY 1
151 1 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 XPRDM 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 XPRDM 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 XPRDM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1172 1 END OVFLOW
FORTRAN CALLING PROCEDURE

CALL XPIIT (LU, RPARMS)

TERMINATE A PROGRAM AFTER WAITING ON ANY BUFFERED I/O TO LU TO COMPLETE THEN PASS PARAMETERS BACK TO THE FATHER TASK.

INPUT

LU - LOGICAL UNIT OF TERMINAL OR OTHER OUTPUT DEVICE FOR WHICH PRINT MAY NOT BE COMPLETED. A VALUE OF ZERO WILL INDICATE NO I/O WAIT

RPARMS - ARRAY OF FIVE PARAMETERS TO BE RETURNED TO FATHER TASK

OUTPUT

RPARMS IS RETURNED TO THE FATHER TASK

NOTES

USES EXEC, PRN

XPIIT DOES NOT RETURN TO THE CALLER.

BEGIN XPIIT

IF LU IS NON-ZERO THEN

WAIT ON ANY CLASS I/O TO COMPLETE (CLASS I/O CONTROL THEN GET)

END IF

RETURN PARAMETERS TO FATHER

TERMINATE PROGRAM

END XPIIT
<table>
<thead>
<tr>
<th>ROOM</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>CONDENSED</td>
</tr>
<tr>
<td></td>
<td>DIRECT</td>
</tr>
<tr>
<td></td>
<td>PRINT</td>
</tr>
<tr>
<td></td>
<td>REPORT</td>
</tr>
<tr>
<td></td>
<td>STORE</td>
</tr>
<tr>
<td></td>
<td>持って行こう</td>
</tr>
<tr>
<td></td>
<td>返す</td>
</tr>
<tr>
<td></td>
<td>2005</td>
</tr>
</tbody>
</table>

5-223
FORTRAN CALLING PROCEDURE

1   100 CALL XRBIT (BIT, BITNUM, STRING)

100

***********

8   101 BIT SET/CLEAR ROUTINE FOR MULTIPLE WORD BIT STRINGS. SET THE
9   101 VALUE OF BIT NUMBER 'BITNUM' IN STRING 'STRING' TO 'BIT'.
10  101

11***********

12   102 INPUT
13   102 BIT - INTEGER VALUE OF ZERO OR ONE TO BE SET IN BIT NUMBER
14   102 'BITNUM' OF 'STRING'
15   102 BITNUM - INTEGER BIT NUMBER OF BIT TO BE SET/CLEARED (FIRST BIT OF
16   102 STRING IS BIT NUMBER ZERO)
17   102 STRING - BIT STRING OF AT LEAST 'BITNUM' BITS IN LENGTH INTO WHICH
18   102 'BIT' IS TO BE SET
19   102

20***********

21   103 OUTPUT
22   103 STRING - BIT NUMBER 'BITNUM' HAS THE VALUE OF 'BIT'
23   103

24***********

25   105 NOTES
26   105 USES .ENTR, XRSET
27   105
28***********

29   1
30   1
31   1
32   1
33   BEGIN XRBIT
34   1  TRANSFER CALLING SEQUENCE
35   2  COMPUTE ADDRESS OF WORD CONTAINING BITNUM
36   2  COMPUTE BIT NUMBER WITHIN WORD
37   2  CALL XRSET TO SET/CLEAR BIT
38   1 END XRBIT
 INTEGER FUNCTION
40 1 00
41 1 01 XRCPR(LENGTH, ARRAY1, ARRAY2)
42 1 00
43 1 00
44 1 ************
45 1 01 COMPARE 'LENGTH' WORDS OF 'ARRAY1' TO 'ARRAY2'. IF THE ARRAYS
46 1 01 ARE IDENTICAL, A VALUE OF 0 IS RETURNED. IF THE COMPARE FAILS
47 1 01 A VALUE IS RETURNED INDICATING WHETHER THE FIRST UNEQUAL
48 1 01 ELEMENT OF ARRAY1 IS LESS THAN ARRAY2 (-1) OR GREATER (+1)
49 1 00
50 1 00
51 1 ************
52 1 02 INPUT
53 1 02 LENGTH - POSITIVE INTEGER NUMBER OF WORDS TO BE COMPARED
54 1 02 ARRAY1 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY2
55 1 02 ARRAY2 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY1
56 1 02
57 1 00
58 1 03 OUTPUT
59 1 03 FUNCTION VALUE - 0, ARRAYS EQUAL
60 1 03 -1, ARRAYS NOT EQUAL, ARRAY1 < ARRAY2
61 1 03 +1, ARRAYS NOT EQUAL, ARRAY1 > ARRAY2
62 1 03
63 1 ************
64 1 05 NOTES
65 1 05 USES .ENTR
66 1 05
67 1 ************
68 1 *
69 1 *
70 1 *
71 1 *
72 1 *
73 1 BEGIN XRCPR
74 2 TRANSFER CALLING SEQUENCE
75 2 SETUP COMPARE
76 2 COMPARE ARRAY1 AND ARRAY2
77 2 RETURN RESULT FLAG
78 1 END XRCPR
FORTRAN CALLING PROCEDURE.

CALL XR018 (DOUBLE, ASCII)

CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN
1P018.IY FORMAT

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

ASCII - NINE ASCII 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, IAOI, IDINT, IOR, KCVT, XREF

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 CD0
2 CD3
3 CD0
4 CD0
5 ************
6 CD1
7 MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE
8 CD1
9 ************
10 CD2
11 INPUT
12 CD2
13 COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPTR
14 CD2
15 ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER
16 CD2
17 ************
18 CD3
19 OUTPUT
20 CD3
21 COMMON XE - REQPTR
22 CD3
23 CLASS I/O WRITE/READ TO CLASS 'CLASNO'
24 CD5
25 REQBUF AND RESPONSE IS PRINTED IF REQUESTED
26 CD5
27 ************
28 CD5
29 NOTES
30 CD5
31 COMMUNICATES WITH FDS MANAGER FATHER TASK
32 CD5
33 USES EXEC, IAND, XRMOV, XRMIG, XR6G, XRSFL, XUDPL, XVPAM
34 CD5
35 ************
36 CD5
37 * *
38 CD5
39 * *
40 CD5
41 ************
42 BEGIN XREQ
43 2 PERFORM TRACE
44 2 OUTPUT REQUESTS TO MANAGER
45 2 REQUEST AWALK MANAGEMENT AND WAIT FOR RESPONSE
46 2 RETRIEVE MANAGER RESPONSE
47 2 PERFORM TRACE
48 2 RETURN RESPONSE IN REQPTR
49 2 END XREQ
50 1 BEGIN TRACE
51 2 IF TRACE REQUESTED
52 2 THEN
53 3 DO FOR EACH REQUEST
54 4 CALL XUDPL TO FORMAT LINE
55 4 OUTPUT LINE
56 3 ENDDO
57 3 OUTPUT PARM1 AND PARM2
58 2 ENDDO
59 1 ENDO TRACE
**INTEGER FUNCTION**

1  *DO  XREXT (START, LENGTH, SOURCE)
2  *DO
3  **********
4  *DO
5  EXTRACT 'LENGTH' BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
6  *DO
7  **********
8  *DO
9  INPUT
10  *DO START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
11  *DO
12  (SIGN BIT = 0)
13  *DO
14  LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
15  *DO
16  *DO SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED
17  *DO
18  **********
19  *DO NOTES
20  *DO USES .ENTR
21  *DO
22  **********
23  *DO
24  *DO
25  *DO
26  BEGIN XREXT
27  *DO
28  TRANSFER CALLING SEQUENCE
29  *DO
30  IF START NOT = 0
31  *DO
32  THEN
33  *DO
34  CONSTRUCT SHIFT
35  *DO
36  LOAD A WITH SOURCE
37  *DO
38  SHIFT BA LEFT START BITS
39  *DO
40  ELSE
41  *DO
42  LOAD A WITH SOURCE
43  *DO
44  EMOIF
45  *DO
46  SAVE A
47  *DO
48  CLEAR B
49  *DO
50  CONSTRUCT SHIFT
51  *DO
52  RESTORE A
53  *DO
54  SHIFT BA LEFT LENGTH BITS
55  *DO
56  MOVE RESULT FROM B TO A
57  *DO
58  END XREXT
FORTAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

1. A single precision real number to an ASCII string in
1PE14.6 format

2. INPUT
3. REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED

4. OUTPUT
5. ASCII - SEVEN CHARACTER STRING REPRESENTATION OF
6. 'REAL'

7. LOCAL
8. WORKING LOCATION FOR ABSOLUTE VALUE OR 'REAL' REPEATEDLY
9. MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

USES FLOAT, IAND, IFIX, IOR, KCVT, XREX, 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 DECIMAL 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 XRI6 (INTEGER, ASCII)

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

INPUT INTEGER - SIXTEEN BIT INTEGER TO BE CONVERTED

OUTPUT ASCII - THREE WORD CHARACTER STRING REPRESENTATION OF 'INTEGER'

LOCAL I - INTERNAL LOCATION FOR 'INTEGER' REPEATEDLY MODIFIED TO PRODUCE 'ASCII'

WORKING BUFFER FOR CONSTRUCTION OF 'ASCII'

USES XRMOV AND XRPCX

BEGIN XRI6

BLANK WORKING SPACE

CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING

SET SIGN OF 'INTEGER' IN 'ASCII'

CALL XRCK TO CONVERT FROM R1 TO A2 FORMAT

END XRI6
FORTRAN CALLING PROCEDURES

CALL XRLCK (RCODE)

XRLCK AND XRLULK 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

'VSTB 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 .ENTRY, RNR.

THIS ROUTINE MAY NOT BE OVERLAYED
337 1 BEGIN XRLCK
338 2 SET REQUEST FOR LOCK FUNCTION
339 2 PERFORM RLCK
340 1 END XRLCK
341 1 BEGIN XRLK
342 2 SET REQUEST FOR UNLOCK FUNCTION
343 2 PERFORM RLK
344 1 END XRLK
345 1 BEGIN RLCK
346 2 IF REQUEST IS CONSISTENT WITH STATUS
347 2 THEN
348 3 SET NEW STATUS
349 3 IF RN IN STB IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
350 3 THEN
351 4 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
352 4 ENDIF
353 3 CLEAR RETURN CODE
354 3 ELSE
355 3 SET RETURN CODE
356 2 ENDIF
357 1 END RLCK
359 1 *00 INTEGER FUNCTION
360 1 *00
361 1 *00 XRLOC(A)
362 1 *00
363 1 ************
364 1 *01 RETURN THE 16-DIT MAPPED ADDRESS OF A
365 1 *01
366 1 *01
367 1 ************
368 1 *02 INPUT
369 1 *02 A - VARIABLE, ROUTINE, ETC. FOR WHICH THE ADDRESS IS DESIRED
370 1 *02
371 1 ************
372 1 *03 OUTPUT
373 1 *03 XRLOC - 16-DIT ADDRESS OF A
374 1 *03
375 1 ************
376 1 *04 NOTES
377 1 *05 NO EXTERNAL REFERENCES
378 1 *05
379 1 ************
380 1 *
381 1 *
382 1 *
383 1 *
384 2 BEGIN XRLOC
385 2 TRANSFER CALLING SEQUENCE
386 2 LOAD THE ADDRESS OF THE CALLING PARAMETER
387 1 END XRLOC

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
389 1 *00    FORTRAN CALLING PROCEDURE
390 1 *00
391 1 *00    CALL XRMOV (LENSTH, 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 *02    INPUT
398 1 *02    LENGTH - POSITIVE INTEGER INDICATING NUMBER OF WORDS TO MOVE
399 1 *02    SOURCE - ARRAY OF WORDS TO BE MOVED
400 1 *02
401 1 *02    ************
402 1 *03    OUTPUT
403 1 *03    LENGTH - ARRAY RECEIVING MOVED WORDS
404 1 *03
405 1 *03    ************
406 1 *04    NOTES
407 1 *04    USES .ENTR
408 1 *04
409 1 *04    ************
410 1 *05
411 1 *05
412 1 *05
413 1 *05
414 1 BEGN XRMOV
415 1 Transfer Calling Sequence
416 1 INITIALIZE MOVE
417 1 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
418 1 END XRMOV
**FORTRAN CALLING PROCEDURE**

**CD0**

```
CD0 CALL XRMSG (NUMBER, LOCATE, LENGTH, SOURCE)
```

**CD2**

```
CD2 INPUT
```

**CD2**

```
CD2 NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
A - AREA INDICATOR AS FOLLOWS
   1 - AS
   2 - XD
   3 - XE
   4 - XI
   5 - XS
   6 - YT
   7 - X
   8 - X
   9 - BF
  10 - SC
NN - MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'
```

**CD2**

```
CD2 LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEDE
CD2 'SOURCE' (NOT USED IF 'N' OF 'NUMBER' IS ZERO)
CD2 LENGTH - INTEGER NUMBER OF WORDS OF 'SOURCE' TO BE INSERTED INTO
CD2 MESSAGE. ZERO INDICATES NO INSERTION
CD2 SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED
CD2 IF 'LENGTH' IS ZERO)
```

**CD2**

```
CD2 COMMON
CD2 LU - USERS LOGICAL UNIT NUMBER
```

**CD2**

```
CD2 OUTPUT
```

**CD3**

```
CD3 UP TO EIGHTY CHARACTER LINE OF TEXT TO UNIT 'LU' OF THE FORM
```

**CD2**

```
CD2 *** AANN MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE
```

**CD5**

```
CD5 NOTES
```

**CD5**

```
CD5 USES FDS SYSTEM MESSAGE FILE XRMSG
CD5 USES CLOSE, EIGC, IAND, KCVT, OPEN, READ, XRMOV, XUDOG
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```

**CD5**

```
CD5
```
1 BEGIN XMSG
2   SEPAFIRE NUMBER INTO AREA AND MESSAGE NUMBER
3   SET NUMBER IN PREFIX
4   READ MESSAGE DIRECTORY RECORD
5   IF AREA VALID
6     SET AREA CODE IN PREFIX
7     IF MESSAGE NUMBER > 0
8     THEN
9     ELSE
10    EXIT TO ERROR:
11    ENDIF
12  ENDIF
13  CALL XMTO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
14  ELSE
15  ENDIF
16  CALL XMTO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
17  THEN
18  CALL XMTO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
19  ELSE
20  ENDIF
21  SET AREA IN PREFIX
22  ERROR: CALL XMTO MOVE 'XMSG ERROR' INTO BUFFER
23  CALL XMTO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
24  ENDIF
25  OUTPUT BUFFER TO USER'S TERMINAL
26  IF OEM IS REQUESTED
27  THEN
28  CALL XUDDG
29  ENDIF
30  END XMSG
INTEGER FUNCTION

XNMB(BIT, BITNUM, STRING)

 thực hiện tìm kiếm bit trong chuỗi 'STRING' bắt đầu từ bit số 'BITNUM' và tiếp tục tìm bit tiếp theo. Sự tìm kiếm sẽ tiếp tục cho đến khi gặp bit 'BIT'.
FORTRAN CALLING PROCEDURE

CALL XRPKC (LENGTH, UNPKED, PACKED)

************
1 *01 CONVERT 'LENGTH' CHARACTERS OF 'UNPKED' FROM R1 FORMAT TO A2
2 *01 FORMAT AND STORE IN 'PACKED'
3 ************
4 *02 INPUT
5 *02 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED
6 *02 UNPKED - ARRAY OF CHARACTERS IN R1 FORMAT
7 *02
8 ************
9 *03 OUTPUT
10 *03 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS
11 *03 ODD, THE LAST WORD WILL BE BLANK FILLED.
12 *03
13 ************
14 *05 NOTES
15 *05 USES .ENTR
16 *05
17 ************
18
19
20 BEGIN XRPKC
21
22 TRANSFER CALLING SEQUENCE
23
24 SET BYTE FLAG FOR HIGH BYTE
25
26 INITIALIZE PACKED POINTER
27
28 DO FOR EACH CHARACTER IN UNPKED
29
30 IF BYTE FLAG SET HIGH
31 Then
32 LOAD A WITH NEXT WORD OF UNPKED
33 SHIFT CHARACTER INTO HIGH BYTE
34 Else
35 INCLUSIVE OR NEXT WORD OF UNPKED INTO A
36 STORE A IN PACKED
37 INCREMENT POINTER
38 ENDIF
39 FLIP BYTE FLAG
40 ENDDO
41 IF BYTE FLAG SET LOW
42 THEN
43 INCLUSIVE OR BLANK INTO LOW BYTE
44 ENDF
45 STORE A IN PACKED
46 END XRPKC
**FORTRAN CALLING PROCEDURE**

657 1 *00  
658 1 *00  
659 1 *00  CALL XROFM (PREFIX, NAME4, NAME6)  
660 1 *00  
661 1 *********  
662 1 *01  
663 1 *01  XROFM BUILDS A QUALIFIED FILE NAME OF UPTO SIX CHARACTERS IN  
664 1 *01  LENGTH BY PREFIXING THE INPUT ONE TO FOUR CHARACTER NAME WITH  
665 1 *01  THE PREFIX CHARACTER AND APPENDING A USER QUALIFIER CODE TO THE  
666 1 *01  END  
667 1 *01  
668 1 *********  
669 1 *02  
670 1 *02  PREFIX - FILE TYPE PREFIX STORED IN R1 FORMAT  
671 1 *02  NAME4 - ONE TO FOUR CHARACTER PACKED NAME TO BE QUALIFIED  
672 1 *02  COMMON XE - QUAL  
673 1 *02  
674 1 *********  
675 1 *03  
676 1 *03  NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME  
677 1 *03  
678 1 *********  
679 1 *05  NOTES  
680 1 *05  USES .ENTR  
681 1 *05  
682 1 *********  
683 1 *  
684 1 *  
685 1 *  
686 1 *  
687 1 BEGIN XROFM  
688 2 STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING  
689 2 MOVE NAME4 INTO NEXT FOUR POSITIONS  
690 2 STORE BLANK IN SIXTH POSITION  
691 2 LOCATE FIRST BLANK CHARACTER  
692 2 REPLACE BLANK WITH USER ID CHARACTER (QUAL)  
693 2 MOVE QUALIFIED NAME TO NAME6  
694 1 END XROFM
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'

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

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
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 PRifix AND SUFFIX
REMOVED

NOTES
USES .ENTR

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
FORTRAN CALLING PROCEDURE

CALL XRUPK (LENGTH, PACKED, UNPKED, COUNT)

INPUT
LENGTH - POSITIVE INTEGER NUMBER OF WORDS IN PACKED
PACKED - ARRAY OF CHARACTER DATA IN A2 FORMAT
OUTPUT
UNPKED - ARRAY OF NON-BLANK CHARACTERS IN R1 FORMAT
COUNT - NUMBER OF CHARACTERS IN UNPKED

NOTES
USES .ENTR
IF PACKED AND UNPKED ARE THE SAME ADDRESS SPACE UNPKED MAY OVERLAY
PACKED

** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTERS.

STRINGS.
839 1 BEGIN XRUPK
840 2 TRANSFER CALLING SEQUENCE
841 2 TURN ON BLANK REMOVAL
842 2 INITIALIZE COUNT
843 2 DO FOR EACH WORD OF PACKED
844 3 LOAD A WITH NEXT WORD
845 3 ROTATE A 8 BITS
846 3 DO FOR EACH BYTE OF WORD
847 4 AND OFF HIGH BYTE
848 4 IF A = QUOTE MARK
849 4 THEN
850 5 CHANGE BLANK REMOVAL OPTION
851 4 ENDIF
852 4 IF BLANK REMOVAL IS ON
853 4 THEN
854 5 IF A NOT = BLANK
855 5 THEN
856 6 INCREMENT COUNT
857 6 STORE A IN UNPKED
858 5 ENDIF
859 4 ELSE
860 5 INCREMENT COUNT
861 5 STORE A IN UNPACKED
862 4 ENDIF
863 4 RELOAD A WITH WORD
864 3 ENDDO
865 2 ENDDD
866 2 RETURN VALUE OF COUNT
867 1 END XRUPK
CALL X13P (CHSTR, LIMIT)

X13P COMPACTS A CHARACTER STRING STORED IN A2 FORMAT SUCH THAT NO MORE THAN ONE CONSECUTIVE BLANK APPEARS EXCEPT BETWEEN QUOTATION MARKS.

CHSTR = CHARACTER STRING OF LENGTH WORDS IN A2 FORMAT

LENGTH = LENGTH OF CHSTR IN WORDS

OUTPUT = CHARACTER STRING OF LENGTH WORDS IN A2 FORMAT WITH DUPLICATE BLANKS REMOVED

NEW LENGTH = NEW LENGTH OF OUTPUT IN WORDS

SPECIAL REMARKS:

THIS ROUTINE ASSUMES THE SPECIAL HANDLING OF STRINGS WITHIN QUOTATION MARKS ON EACH END. THE FIRST/LAST CHARACTER STRING WAS THE LENGTH-1 STRING. THIS BIT IS TURNED "ON" BY THE PROGRAM PRIOR TO EXIT. THIS CHANGE WAS MADE TO PREVENT A X13P/COMPACT PROCESSOR FROM MISRECOGNIZING THE END OF THE STRING.
BEGIN XR1SP

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 'THF:

REPLACE CHARACTER STRING INDICATOR WITH QUOTE MARK

FLIP CHARACTER STRING FLAG

CLEAR BLANK FLAG

ELSE

IF CHARACTER STRING FLAG IS CLEAR, THEN

IF CHARACTER IS A BLANK, THEN

IF BLANK FLAG IS SET (AT LEAST ONE PREDECESSING BLANK), THEN

SKIP THIS CHARACTER (EXIT TO ENDDO)

ELSE

SET BLANK FLAG

ENDIF

ELSE

CLEAR BLANK FLAG

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

ENDDO
1 CDO FORTRAN CALLING PROCEDURE
2 CDO
3 CDO CALL XRLBS(XSERE)
4 CDO
5 CDO
6 CDO
7 CDO
8 CDO XSERE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR
9 CDO
10 CDO
11 CDO INPUT
12 CDO
13 CDO
14 CDO
15 CDO COMMON XE - LU
16 CDO COMMON XB - DEBUG, IRET, NEWTAB, NUMENT, PRMLEN
17 CDO PROMPT, PRMTHD, WKBUF
18 CDO
19 CDO
20 CDO OUTPUT
21 CDO
22 CDO
23 CDO COMMON XE - RERBUF, REAPTR
24 CDO
25 CDO
26 CDO COMMON XB - NUMENT, PRMLEN, PRMTHD, PROMPT, WKBUF
27 CDO
28 CDO
29 CDO NOTES
30 CDO
31 CDO USES Routines
32 CDO
33 CDO
34 CDO EXEC
35 CDO
36 CDO
37 CDO
38 CDO
39 CDO
40 CDO
41 CDO
42 CDO
43 CDO
44 CDO
45 CDO
1 CD0
2 FORTRAN CALLING PROCEDURE
3
4 CD0
5 CALL XSNPT
6
7 CD0
8 PROCESS THE INPUT RESPONSES OF THE SEQUENCE
9
10 CD1
11 TABLE EDITOR
12
13 CD1
14 INPUT
15
16 CD2
17 COMMON XE - COMBUF, COMPTR, LU, TOKENS
18
19 CD2
20 COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, PRMTMD
21 SEQNO, TABNOX, WKBUF
22
23 CD2
24 COMMON XB - INSERT, IRETC, NUMENT, PRMTMD, SEQNO,
25 TABNOX, TABSIZ, WKBUF
26
27 CD3
28 OUTPUT
29
30 CD3
31 COMMON XE - COMPTR
32
33 CD3
34 COMMON XB - INSERT, IRETC, NUMENT, PRMTMD, SEQNO,
35 TABNOX, TABSIZ, WKBUF
36
37 CD3
38 NOTES
39
40 CD3
41 USES ROUTINES
42
43 CD3
44 XRMAG
45
46 CD3
47 XDELS
48
47 CD3
48 XENF
49
49 CD3
49 XSLIS
50
50 CD3
50 XSNUM
51
51 CD3
51 XSMT
52
52 CD3
52 XUDAG
53
53 CD3
53 C********
1 BEGIN XSNPT
2 IF PROMPT MODE IS UPDATE, THEN
3 IF TOKEN INPUT IS AN INTEGER, THEN
4 ERREXIT IF INTEGER < 1 :ERROR;
5 RETAIN INTEGER AS SEQUENCE NO. (SEND)
6 INCREMENT TO NEXT TOKEN
7 ERREXIT IF TOKEN IS NOT "=" :ERROR;
8 INCREMENT TO NEXT TOKEN
9 START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED
10 EXIT IF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SEND)
11 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
12 SET INSERT FLAG TO ZERO INDICATING REPLACEMENT OF ENTRY
13 EXIT SEQUENCE NO. OF ENTRY > SEQUENCE NO. INPUT (SEND)
14 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
15 SET INSERT FLAG TO 1 INDICATING INSERT NEEDED
16 GROELSE
17 INCREMENT TO NEXT TABLE ENTRY
18 ENLOOP
19 SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE
20 SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE
21 ENSEARCH
22 CALL XSENT TO "REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG"
23 ELSE
24 ERREXIT IF TOKEN IS NOT A NAME :ERROR:
25 START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED
26 EXIF NAME INPUT IS DIRECTIVE
27 SET INDEX TO DIRECTIVE LIST ENTRY
28 GROELSE
29 INCREMENT TO NEXT DIRECTIVE
30 ENLOOP
31 ENSEARCH
32 CASE (ISLIS, XSDEL, XSNML, XSNRO, XSNRM), INDEX
33 ENDF
34 ELSE, PROMPT MODE IS NOT UPDATE
35 IF TOKEN IS #, THEN
36 SET PROMPT MODE TO UPDATE
37 F -F
38 IF AN & IS INPUT, THEN
39 ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR;
40 MARK THIS TABLE ENTRY AS DELETED
41 DO FROM END OF TABLE UNTIL A NONDELETED ENTRY IS FOUND
42 IF TABLE ENTRY IS MARKED FOR DELETION, THEN
43 DECREMENT NUMBER OF TABLE ENTRIES BY ONE
44 ENDF
45 ENDDO
46 ELSE
47 IF PROMPT MODE IS ALL, THEN
48 SET INSERT FLAG TO TWO TO INDICATE EXTENSION OF TABLE
49 ELSE
50 SET INSERT FLAG TO ZERO TO INDICATE REPLACEMENT OF TABLE ENTRY
51 ENDF
52 CALL XSENT TO BUILD ENTRY BASE ON INSERT FLAG
53 ENDF
54 ENDF
55 ENDF
56 EXIT XSNPT
57 :ERROR: CALL XRMG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
263 | COO | FORTRAN CALLING PROCEDURE
264 | COO | XSENT
265 | COO | CALL XSENT
266 | COO | XSENT
267 | COO | XSENT
268 | COO | XSENT CONSTRUCTS SEQUENCE TABLE ENTRIES
269 | COO | XSENT
270 | COO | COMMON XE - COMBUF, COMPTU, TOKENS, LU
271 | COO | COMMON XE - DEBUG, LIBDZ, NMENT, PRTMND,
272 | COO | SEGO, TABIDX, WKBUF
273 | COO | OUTPUT
274 | COO | COMMON XE - COMPTU
275 | COO | COMMON XE - INTNAM, NMENT, PRCNAME, PRTMND,
276 | COO | SEGO, TABIDX, WKBUF
277 | COO | XSENT
278 | COO | XSENT
279 | COO | XSENT
280 | COO | XSENT
281 | COO | XSENT
282 | COO | XSENT
283 | COO | XSENT
284 | COO | XSENT
285 | COO | XSENT
286 | COO | XSENT
287 | COO | XSENT
288 | COO | XSENT
289 | COO | XSENT
290 | COO | XSENT
291 | COO | XSENT
292 | COO | XSENT
293 | COO | XSENT
294 | COO | XSENT
295 | COO | XSENT
296 | COO | XSENT
297 | COO | XSENT
298 | COO | XSET
299 | COO | XSENT
300 | COO | XSENT
301 | COO | XSENT
302 | COO | XSENT
303 | COO | XSENT
304 | COO | ERREXIT IF Token INPUT IS NOT A NAME :ERROR1:
305 | COO | RETAIN THIS NAME AS PROC. NAME
306 | COO | INCREMENT TO NEXT TOKEN
307 | COO | START SEARCH UNTIL ALL ENTRIES OF LIBD SEACHED
308 | COO | EXIT IF LIBD ENTRY = PROC. NAME
309 | COO | XSENT
310 | COO | XSENT
311 | COO | XSENT
312 | COO | XSENT
313 | COO | XSENT
314 | COO | XSENT
315 | COO | EXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERROR4:
316 | COO | INCREMENT TO NEXT TOKEN
317 | COO | ERROR IF NEXT TOKEN IS NOT A NAME :ERROR1:
318 | COO | RETAIN THIS NAME AS INTERFACE TABLE NAME
319 | COO | INCREMENT TO NEXT TOKEN
320 | COO | ELSE
321 | COO | INTERFACE TABLE NAME TO ZERO

ERREXIT IF Token INPUT IS NOT A NAME :ERROR1:
RETAIL THIS NAME AS PROC. NAME
INCREMENT TO NEXT TOKEN
START SEARCH UNTIL ALL ENTRIES OF LIBD SEACHED
EXIT IF LIBD ENTRY = PROC. NAME
ORELSE
INCREMENT TO NEXT ENTRY
ENDLOOP
ERREXIT :ERROR3:
ENDSEARCH
IF COMM IS NEXT TOKEN, THEN
ERREXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERROR4:
INCREMENT TO NEXT TOKEN
ERROR 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
359 2 CD0  FORTRAN CALLING PROCEDURE
360 2 CD0
360 2 CD0  CALL XSPCK
360 2 CD0
364 2 CD1
364 2 CD1  XSPCK COMPACTS THE WORKING BUFFER BY REMOVING ALL SEQUENCE
367 2 CD1  TABLE ENTRIES MARKED FOR DELETION
368 2 CD1
369 2 CD1
370 2 CD2
370 2 CD2  INPUT
370 2 CD2
375 2 CD2  COMMON XE - LU
375 2 CD2  COMMON XB - DEBUG, NUMENT, TABNOX, WBUF
375 2 CD2
377 2 CD3
377 2 CD3  OUTPUT
377 2 CD3  COMMON XB - NUMENT, TABNOX, WBUF
377 2 CD3
383 2 CD5
384 2 CD5  NOTES
385 2 CD5  USES ROUTINES
386 2 CD5  XRMOV
387 2 CD5  XRMSG
387 2 CD5  XUD8G
387 2 CD5
393 2 CD5

BEGIN XSPCK
IF THE TABLE IS NOT EMPTY, THEN
DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
IF THIS ENTRY IS MARKED DELETED, THEN
SET MOVE LENGTH (MOVLEN) TO 7
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT MOVLEN BY 7
ENDDO
MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
THE DELETED ENTRY
DECREMENT NUMENT BY MOVLEN / 7
IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
ENDIF
ENDDO
ENDIF
END XSPCK
414 2  CD0  FORTRAN CALLING PROCEDURE
415 2  CD0
416 2  CD0
417 2  CD0  CALL XSLST
418 2  CD0
419 2  CD0  ********
420 2  CD0
421 2  CD0  XSLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
422 2  CD0  CONTAINED IN THE WORKING BUFFER
423 2  CD0
424 2  CD0  ********
425 2  CD0
426 2  CD0  INPUT
427 2  CD0
428 2  CD0  COMMON XE - LU, RERBUF, SUBSTA
429 2  CD0
430 2  CD0  COMMON XB - BEGNO, DEBUG, ENDM0, LISTLU,
431 2  CD0
432 2  CD0
433 2  CD0  ********
434 2  CD0
435 2  CD0  NOTES
436 2  CD0
437 2  CD0  USES Routines
438 2  CD0
439 2  CD0
440 2  CD0
441 2  CD0
442 2  CD0
443 2  CD0
444 2  CD0  ********
445 2  CD0
446 3  BEGIN XSLST
447 4  IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
448 4  DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
449 4  SET LIMITS (BEGIN AND ENDM0) OF SEQ. ENTRIES LISTED
450 4  SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
451 3  ENDIF
452 3  WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
453 3  IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
454 4  DO FROM BEGNO TO ENDM0
455 5  MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
456 5  IF INT. TABLE NAME = 0, THEN
457 6  SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARs.)
458 6  ELSE
459 6  SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARs.)
460 6  ENDIF
461 5  CALL XRSIG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
462 4  WRITE PRINT BUFFER
463 3  ENDD0
464 2  END XSLST
BEGIN XSCAN
501  2  BEGIN XSCAN
502  3  SET LIST LIMITS (BEGINO AND ENDMO) TO ZERO
503  3  IF NEXT TOKEN IS A COMMA, THEN
504  4  INCREMENT TO NEXT TOKEN
505  4  IF NEXT TOKEN IS AN INTEGER, THEN
506  5  ERREXIT IF VALUE IS < 1 :ERROR6:
507  5  SET BEGIN LIMIT (BEGINO) TO THIS VALUE
508  5  INCREMENT TO NEXT TOKEN
509  4  ENDFI
510  4  IF NEXT TOKEN IS A COMMA, THEN
511  5  INCREMENT TO NEXT TOKEN
512  5  IF NEXT TOKEN IS AN INTEGER, THEN
513  6  ERREXIT IF VALUE IS < 1 :ERROR6:
514  6  SET END LIMIT (ENDMO) TO THIS VALUE
515  6  INCREMENT TO NEXT TOKEN
516  4  ENDFI
517  4  ENDFI
518  4  ENDFI
519  3  ERREXIT IF NEXT TOKEN IS NOT EOS :ERROR1:
520  3  IF BEGIN LIMIT (BEGINO) = 0, THEN
521  4  SET BEGIN LIMIT (BEGINO) TO 1 (BEGINO IS NOW A TABLE INDEX)
522  4  ELSE
523  4  START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
524  4  EXIT IF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGINO)
525  5  SET BEGIN LIMIT (BEGINO) TO INDEX OF THIS ENTRY
526  4  ORELSE
527  5  INCREMENT INDEX TO NEXT TABLE ENTRY
528  5  ENDOLOOP
529  5  ERREXIT :ERROR6:
530  4  ENDOSEARCH
531  3  ENDFI
532  3  IF END LIMIT (ENDMO) = 0, THEN
533  4  SET END LIMIT (ENDMO) TO INDEX OF LAST TABLE ENTRY
534  4  ELSE
535  4  START SEARCH FROM BEGIN LIMIT (BEGINO) TO LAST TABLE ENTRY
536  4  EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDMO)
537  4  SET END LIMIT (ENDMO) TO INDEX OF THIS ENTRY
538  4  ORELSE
539  5  INCREMENT INDEX TO NEXT TABLE ENTRY
540  5  ENDOLOOP
541  5  ERREXIT :ERROR6:
542  4  ENDOSEARCH
543  3  ENDFI
544  3  SET RETURN CODE TO INDICATE NO ERROR
545  3  SET RETURN CODE TO INDICATE AN ERROR
546  2  EXIT XSLIB
547  3  :ERROR1: CALL XRMNG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
548  3  :ERROR6: CALL XRMNG - 'INVALID SEQUENCE NUMBER'
549  3  :ERROR6: CALL XRMNG - 'INVALID SEQUENCE NUMBER RANGE'
550  2  END XSCAN
BEGIN XSPMT
  ERREXIT IF TOKEN IS NOT COMMA :ERROR1:
  INCREMENT TO NEXT TOKEN
  ERREXIT IF TOKEN IS NOT A NAME :ERROR1:
  INCREMENT TO NEXT TOKEN
  ERREXIT IF TOKEN IS NOT EOS :ERROR1:
  IF NAME IS 'N', THEN
    SET PROMPT MODE TO CREATE
  ELSE
    ERREXIT IF NAME IS NOT 'A' :ERROR9:
  SET PROMPT MODE TO ALL
  SET TABLE ENTRY INDEX (TABNdx) TO 0
END IF
EXIT XSPMT
:ERROR1: CALL XRMSPG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERROR9: CALL XRMSPG - 'SYNTAX ERROR - INVALID QUALIFIER'
END XSPMT
C00 FORTRAN CALLING PROCEDURE
C01 CALL XSDCL
C02
C03 COMMON XE - LU
C04 COMMON XB - BGN0, DEBUG, END00, IRET, NUM0NT
C05
C06 COMMON XM - NUNIT, WKBW
C07
C08
C09 NOTES
C10 USES Routines
C11 XSCAN
C12 XDBG
C13
C14 BEGIN XSDCL
C15 CALL XSCAN TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE
C16 IF NO ERROR INDICATED, THEN
C17 DO FROM THE BEGIN LIMIT (BGN0) TO THE END LIMIT (EN00)
C18 MARK THIS SEQ. TABLE ENTRY AS DELETED
C19 ENDDO
C20 DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR
C21 UNTIL A NON-DELETED ENTRY FOUND
C22 IF ENTRY IS MARKED DELETED, THEN
C23 DECREMENT NUMBER OF TABLE ENTRIES (NUM0NT) BY ONE
C24 ENDF1
C25 ENDDO
C26 ENDF1
C27 END XSDCL
FORTRAN CALLING PROCEDURE

CALL ISLIS

******

(XSLTS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR)

******

INPUT

COMMON XE - LU

COMMON XE - DEBUG, XE TLC

******

NOTES

USES ROUTINES

XICAN

XSLST

XUDBG

******

FORTRAN CALLING PROCEDURE

CALL XSNUM

******

(XSNUM IS THE SEQUENCE TABLE EDITOR NUMBER DIRECTIVE PROCESSOR)

******

INPUT

COMMON XE - COMBUF, COMPAR, LU, TOKENS

COMMON XE - DEBUG, XE NUM

******

OUTPUT

COMMON XE - WKBUF

******

NOTES

USES ROUTINES

XHMSG

XUDBG
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
**FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:**

**CALL XITCOM (PROMPT, PRMLEN, RETCOD)**

**WRITE** "PRMLEN" **W**ORDS **O**F **C**HARACTERS "PROMPT" **T**O **T**HE **U**SER,

**READ** **T**HE **U**SER'S **R**ESPONSE, **C**ALL **XTLAM** **T**O **A**NALYZE **T**HE **R**ESPONSE,

**AND** **R**ETURN **A** "RETCOD" **T**O **T**HE **U**SER **I**NDICATING **T**HE **S**UCCESS **O**F

**XTCOM** **A**ND **T**HE **U**SEFULNESS **O**F **T**HE **R**ESPONSE.

**INPUTS FROM CALLING SEQUENCE:**

- **PROMPT** - (INTEGER, PRMLEN WORDS) AN ARRAY OF PRMLEN WORDS
- **PRMLEN** - (INTEGER, 1 WORD) THE LENGTH IN WORDS OF
  - **THE PROMPT ARAY**. **M**AXIMUM **L**ENGTH **I**S **38** **W**ORDS.

**OUTPUTS IN CALLING SEQUENCE:**

- **RETCOD** - (INTEGER, 1 WORD) IS A COMPLETION CODE DEFINED
  - **AS FOLLOWS:**
    - 0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE
    - 1 - USER RESPONDED X. BUFFER CONTAINS RESPONSE UP TO AND INCLUDING X.
    - 2 - USER ENTERED A CR. THERE IS NO RESPONSE
    - 3 - PROMPT WAS TOO LONG. MAXIMUM LENGTH IS 76
    - 4 - CHARACTERS (38 WORDS)
    - 5 - USER REQUESTED A CONTINUATION

**INTERNAL VARIABLES**

- **COMMSG** - 6 WORD ARRAY CONTAINING THE CONTINUATION MESSAGE
- **PREFIX** - 4 CHARACTERS USED AS PREFIXES TO PROMPT,
- **CORRESPONDING TO CODES IN XE(5)
- **RETCEN** - RETURN CODE FROM XITXM EXTENDED PROMPTING
- **RETCAL** - RETURN CODE FROM XITXN LEXICAL ANALYSIS
- **SUFFIX** - CHARACTERS APPENDED TO END OF PROMPT

**XE COMMON USED:**

- EQUIVALENCE (XE(1), LU), (XE(2), ICLASS),
  (XE(3), LFVL), (XE(115), IOMNIP),
  (XE(145), NOTOKS), (XE(146), NOWDS)

**XS COMMON USED:**
BEGIN XTCOM
IF PROMPT IS NOT TOO LONG (76 CHARS) 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 LR 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
ENDIF
IF LA RETURN CODE SAYS ERROR IN RESPONSE THEN
CALL KCVT TO CONVERT OCTAL TO ASCII
CALL XRMGE TO WRITE ERROR MESSAGE
GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF
IF LA RETURN CODE SAYS OVERFLOW/UNDERFLOW THEN
CALL KCVT TO CONVERT OCTAL TO ASCII
CALL XRMGE TO WRITE ERROR MESSAGE
GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF
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 XRMGE TO WRITE ERROR MESSAGE
ENDIF
GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF
IF LA RETURN CODE SAYS COMBINE IS FULL THEN
CALL XRMGE TO WRITE ERROR MESSAGE
GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF
SET XTCOM RETURN CODE = LA RETURN CODE
ELSE
SET XTCOM RETURN CODE = PROMPT IS TOO LONG
ENDIF
END XTCOM
BEGIN READSEG
CALL XRM0V TO INITIALIZE INPUT BUFFER TO BLANKS
CALL EXEC TO READ RESPONSE TO PROMPT
CALL XRUPK ROUTINE TO CONVERT A2 FORMAT BUFFER TO R1
IF NUMBER OF WORDS READ IS NOT ZERO THEN
CALL XTLM ROUTINE TO BUILD COMMUNICATIONS BUFFER
ELSE
IF LAST LA RETURN CODE WAS A CONTINUE THEN
REMOVE TRAILING COMMAS FROM COMBUF
SET LA RETURN CODE TO NORMAL RETURN
ELSE
SET LA RETURN CODE TO SAY USER ENTERED CR
ENDIF
ENDIF
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 I. 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
DBLWD - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER
AND FRACTIONAL VALUE FOR DOUBLE PRECISION OR REM.
FLGCON - COMMA FLAG
LAST CHARACTER WAS NOT A COMMA
1 - LAST CHARACTER WAS A COMMA
FLGEND - END LOOP FLAG
0 - CONTINUE LOOP
1 - END LOOP
FLGNEG - NEGATIVE EXPONENT FLAG
0 - EXPONENT WAS POSITIVE
1 - EXPONENT WAS NEGATIVE
FLGTPR - TYPE OF REAL VALUE
0 - SINGLE PRECISION
1 - DOUBLE PRECISION
POWER - EXPONENT PART OF A REAL NUMBER
RELDW - SINGLE PRECISION LOCATION FOR REAL VALUE
SPCHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING
THE KI FORMAT REPRESENTATION FOR:
X IS AN EXCLAMATION POINT
Y IS AN OPEN BRACKET
Z IS A BACK SLASH
204  1 CD4
205  1 CD************
206  1 CD4
207  1 CD************
208  1 CD5
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************

EXTERNALS

SUBROUTINES AND FUNCTIONS CALLED

XRPC, XRMV

PDL ROUTINES INCLUDED:

XTLAM, COMMA, ALPHA, DIGIT, DCOL, DECPY, EQRD
INTEG, REAL, DBL, REPE, INVAL, SCHARS, QUOTE
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
  ENDIF
SET END FLAG ON
ELSE
  ERREXIT IF COMBUF IS FULL PERFORM COMFUL
  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
ENDIF
END DO
STORE END OF BUFFER TOKEN IN COMBUF
INCREMENT #TOKEMS BY 1
END XTLAN

BEGIN COMMA
SET LAST CHARACTER WAS A COMMA ON
ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
STORE COMMA TOKEN IN COMBUF
INCREMENT #WORDS IN COMBUF BY 1
INCREMENT #TOKEMS BY 1
GET NEXT INPUT CHARACTER
END COMMA
BEGIN ALPHA
2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
3 CALL XRMOY TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
4 DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
5 INPUT CHARACTER IS A NUMERIC OR
6 INPUT CHARACTER IS AN EXCLAMATION POINT) AND
7 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
8 MOVE CHARACTER INTO TEMPORARY BUFFER
9 GET NEXT INPUT CHARACTER
10 ENDDO
11 SET CHARACTER COUNT = 6
12 STORE CHARACTER NAME TOKEN IN COMBUF
13 CALL XRPACK ROUTINE TO PACK CHARACTERS INTO COMBUF
14 INCREMENT WORDS IN COMBUF BY 4
15 INCREMENT #TOKENS BY 1
16 END ALPHA
17
18 BEGIN DIGIT
19 INITIALIZE POWER TO ZERO
20 PERFORM DCOL
21 IF INPUT BUFFER IS NOT EXHAUSTED THEN
22 IF INPUT CHARACTER IS A "0" THEN
23 PERFORM DECP
24 ELSE
25 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
26 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
27 PERFORM EOMD
28 ELSE
29 IF INPUT CHARACTER IS AN "R" THEN
30 PERFORM REPEP
31 ELSE
32 PERFORM INTEGR
33 ENDIF
34 ENDIF
35 ELSE
36 PERFORM INTEGR
37 ENDIF
38 END DIGIT
39
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 ERRSET IF CHARACTER IS NOT A DIGIT PERFORM INVAL.
367 19 PERFORM BCOL
368 20 IF NEGATIVE FLAG IS ON THEN
369 21 SET POWER = -POWER
370 22 ENDIF
371 23 IF TYPE FLAG IS "E" THEN
372 24 PERFORM REAL
373 25 ELSE
374 26 PERFORM DBL
375 27 ENDIF
376 28 1 END EORD
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>379</td>
<td>1 BEGIN INTEGER</td>
</tr>
<tr>
<td>380</td>
<td>2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL</td>
</tr>
<tr>
<td>381</td>
<td>3 CONVERT NUMBER TO INTEGER</td>
</tr>
<tr>
<td>382</td>
<td>4 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>383</td>
<td>5 STORE INTEGER TOKEN IN COMBUF</td>
</tr>
<tr>
<td>384</td>
<td>6 INCREMENT WORDS IN COMBUF BY 2</td>
</tr>
<tr>
<td>385</td>
<td>7 INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>386</td>
<td>8 END INTEGER</td>
</tr>
<tr>
<td>387</td>
<td>9</td>
</tr>
<tr>
<td>388</td>
<td>10</td>
</tr>
<tr>
<td>389</td>
<td>11 BEGIN REAL</td>
</tr>
<tr>
<td>390</td>
<td>12 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL</td>
</tr>
<tr>
<td>391</td>
<td>13 SET REAL = DOUBLE PRECISION * 10 ** POWER</td>
</tr>
<tr>
<td>392</td>
<td>14 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>393</td>
<td>15 STORE REAL TOKEN IN COMBUF</td>
</tr>
<tr>
<td>394</td>
<td>16 INCREMENT WORDS IN COMBUF BY 3</td>
</tr>
<tr>
<td>395</td>
<td>17 INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>396</td>
<td>18 END REAL</td>
</tr>
<tr>
<td>397</td>
<td>19</td>
</tr>
<tr>
<td>398</td>
<td>20</td>
</tr>
<tr>
<td>399</td>
<td>21 BEGIN DBL</td>
</tr>
<tr>
<td>400</td>
<td>22 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL</td>
</tr>
<tr>
<td>401</td>
<td>23 SET DBL = DOUBLE PRECISION * 10 ** POWER</td>
</tr>
<tr>
<td>402</td>
<td>24 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>403</td>
<td>25 STORE DOUBLE TOKEN IN COMBUF</td>
</tr>
<tr>
<td>404</td>
<td>26 INCREMENT WORDS IN COMBUF BY 4</td>
</tr>
<tr>
<td>405</td>
<td>27 INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>406</td>
<td>28 END DBL</td>
</tr>
<tr>
<td>407</td>
<td>29</td>
</tr>
<tr>
<td>408</td>
<td>30</td>
</tr>
<tr>
<td>409</td>
<td>31 BEGIN REPET</td>
</tr>
<tr>
<td>410</td>
<td>32 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL</td>
</tr>
<tr>
<td>411</td>
<td>33 ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL</td>
</tr>
<tr>
<td>412</td>
<td>34 STORE REPEAT TOKEN IN COMBUF</td>
</tr>
<tr>
<td>413</td>
<td>35 INCREMENT WORDS IN COMBUF BY 2</td>
</tr>
<tr>
<td>414</td>
<td>36 INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>415</td>
<td>37 GET NEXT CHARACTER</td>
</tr>
<tr>
<td>416</td>
<td>38 END REPET</td>
</tr>
</tbody>
</table>
421 1 BEGIN SC Xi
422 2 2 +
423 3 3 nh ARE INVALID CHARACTERS HERE
424 4 x IS A CLOSED BRACKET
425 5 y Is An OPEN BRACKET
426 6 z IS A BACK SLASH
427 7
428 8 SET J=1
429 9 SEARCH WHILE J<#CHARACTERS IN TABLE
430 10 EXIT IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
431 11 INCREMENT J BY 1
432 12 ENDLOOP
433 13 PERFORM INVALID - NO RETURN
434 14 ENDSEARCH
435 15 SET NEXT FLAG ON
436 16 = "* + - * / < > # ?"
437 17 CASE J:
438 18 ( ) * [ ] * { } * B:
439 19 : A:
440 20 SET NEXT FLAG OFF
441 21 PERFORM QUOTE
442 22 SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED
443 23 SET END FLAG ON
444 24
445 25 :C:
446 26 IF SYMBOLIC STRING FLAG IS OFF THEN
447 27 SET SYMBOLIC STRING FLAG TO CURRENT COMBUFF INDEX + 1
448 28 ELSE
449 29 SET COMBUFF(SYMBOLIC STRING FLAG) = CURRENT COMBUFF INDEX - SYMBOLIC STRING FLAG
450 30 SET J = J+1 TO STORE SYMBOLIC STRING CLOSE TOKEN
451 31 SET SYMBOLIC STRING FLAG OFF
452 32 ENDIF
453 33
454 34 SET RETURN CODE TO SAY X ENTERED
455 35 SET END FLAG ON
456 36
457 37 :E:
458 38 IF FOLLOWING CHARACTER IS A DIGIT THEN
459 39 SET NEXT FLAG OFF
460 40 SET INTEGER = 0
461 41 SET POWER = 0
462 42 PERFORM DEC
463 43 ENDIF
464 44 :F:
465 45 ENDCASE
466 46 IF NEXT FLAG IS ON
467 47 STORE TOKEN (J) IN COMBUFF
468 48 INCREMENT WORDS IN COMBUFF BY 1
469 49 IF TOKEN IS BEGIN SYMBOLIC STRING THEN
470 50 INCREMENT WORDS IN COMBUFF BY 1
471 51 ENDIF
472 52 INCREMENT #STANCES BY 1
473 53 GET NEXT CHARACTER
BEGIN QUOTE
GET NEXT CHARACTER
GET #CHARACTERS = 0
DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
(INPUT CHARACTER IS A QUOTE AND
NEXT CHARACTER IS A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
INCREMENT #CHARACTERS BY 1
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
ENDO
ENDOQ
EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN
PERFORM COMFUL
EXIT IF LENGTH OF CHARACTER STRING IS 0 OR
EXIT IF INPUT CHARACTER IS NOT A QUOTE
PERFORM INVAL
STORE CHARACTER STRING TOKEN IN COMBUF
CALL XRCPK TO PACK CHARACTERS INTO COMBUF
INCREMENT #WORDS IN COMBUF BY 2+(#CHARACTERS+1)/2
INCREMENT #TOKENS BY 1
GET NEXT CHARACTER
END QUOTE
503    1 BEGIN INVAL
504    2 SET RETURN CODE TO SAY INVALID RESPONSE
505    2 SET END FLAG ON
506    1 EXIT XILAN
507    1 END INVAL
508    1 *
509    1 *
510    1 *
511    1 BEGIN COUNFUL
512    2 SET RETURN CODE TO SAY COUNFUL IS FULL
513    2 SET END FLAG ON
514    1 EXIT XILAN
515    1 END COUNFUL
516    1 *
517    1 *
518    1 *
519    1 BEGIN OVERFLOW
520    2 SET RETURN CODE TO SAY OVERFLOW/UNDERFLOW
521    2 SET END FLAG ON
522    1 EXIT XILAN
523    1 END OVERFLOW
CD0       FORTRAN CALLING PROCEDURE
       CALL XTPRM

CD1       XTPRM IS CALLED BY XTCOM TO PROVIDE EXTENDED PROMPTING WHEN A
 ? IS ENTERED FROM THE USER TERMINAL.

CD2       ----------
CD3 INPUT  
CD4          XML COMMON - CARTRG, COMBUF, FLAGS, LU, NOPROC, PRBKM, SUBSTA,
CD5 TOKENS
CD6          XB COMMON - AREPRT, UKBUF (FROM INTERFACE TABLE EDITOR)
CD7          XS COMMON - PRXLEM (FROM XTCOM)
CD8          ----------
CD9          VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)
CD10         ----------
CD11 OUTPUT  
CD12          XML COMMON - COMBUF (USED FOR SCRATCH SPACE)
CD13          ----------
CD14 LISTING OF APPROPRIATE EXTENDED PROMPTS
CD15         ----------
CD16 INTERNAL VARIABLES
CD17          CDA CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY
CD18          CDA FILE - FILE NAME OF CURRENT TABLE ENTRY
CD19          CDA INDEX - INDEX TO CURRENT TABLE ENTRY
CD20          CDA L   - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO
CD21          CDA A FIRST LIST ITEM IN RECORD 1 (SEE SDD SECTION 6.2.4.3)
CD22          CDA MES - MESSAGE NUMBER, IF ANY, FOR CURRENT TABLE ENTRY
CD23          CDA N   - NUMBER OF LIST ITEMS IN RECORDS 1 AND 2 (SEE SDD
CD24          CDA SECTION 6.2.4.3)
CD25          CDA R  - RECORD NUMBER WITH WHICH TO BEGIN PROCESSING FOR CURRENT
CD26          CDA TABLE ENTRY
CD27          CDA S     - LIST SEARCH FLAG (1) FOR CURRENT TABLE ENTRY
CD28          CDA T     - LIST ITEM SIZE OF CURRENT TABLE ENTRY (NEGATIVE INDICATES
CD29          CDA LAST CHARACTER TO BE MASKED)
CD30          CDA TABLE - PROCESSING CONTROL TABLE FOR VARIOUS SYNTAX CONDITIONS
CD31         ----------
CD32 SYNTAX CONDITION INDEX ENTRY AS FOLLOWS
CD33          CDAI I   I:?:  FILE REC SIZE SRC MESSAG CONTINUE
CD34          CDAI 1 I   I:?: $X$PRM  1  2  NO  NONE  NO I
CD35          CDAI 2 I   I:?:R  9 I  $XPRM  1  2  YES  XTO6  NO I
CD36          CDAI 3 I   I:?:  17 I  $XPRM  1  2  NO  NONE  YES I
CD37          CDAI 4 I   I:?: $P?:?, $P?:  25 I  XLIBD  2  -3  NO  NONE  NO I
CD38          CDAI 6 I   I:?:R, $P?:?, $P?:R  333 I  $XPRM  1  2  YES  NONE  YES I
CD39          CDAI 7 I   I:?:  41 I  $PPDC  3  128  NO  XTO7  NO I
CD40          CDAI 8 I   I:?:  49 I  $XPRM  1  2  NO  NONE  NO I
CD41          CDAI 9 I   I:?:R  65 I  $XPRM  1  2  YES  NONE  YES I
CD42          CDAI 10 I  I:?:, $P?:, $P?:  73 I  $XPRM  1  2  YES  XTO8  NO I
CD43 WHERE P AND R INDICATE A PROMPT AND RESPONSE

CD44         ----------
CD45 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 > 128), 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 XRMNG TO DISPLAY 'NOT VALID ...' MESSAGE
28 ENDIF
29 EXIT TO ENDDO
30 END SEARCH
31 ENDIF
32 PERFORM DISPLAY
33 ELSE OPEN ERROR
34 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
35 CALL XRMNG TO DISPLAY 'NOT VALID ...' MESSAGE
36 ELSE
37 :ERROR9: CALL XRMNG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
38 CLOSE FILE
39 ENDIF
40 ENDDO
41 END XTPRM
1 BEGIN SETUP
2 EXIT TO :ERROR2: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
3 CASE (:X:, :S:, :F:, :/) SUBSTA
4 IF FIRST TOKEN = ?, THEN
5 SET TABLE INDEX TO FIRST ENTRY
6 ELSE
7 SET TABLE INDEX TO SECOND ENTRY
8 ENDF
9 :S:
10 IF FIRST TOKEN = ?, THEN
11 SET TABLE INDEX TO FOURTH ENTRY
12 ELSE
13 SET TABLE INDEX TO SIXTH ENTRY
14 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
15 ENDF
16 :F:
17 IF PROMPT LENGTH = 0, THEN
18 ASSUME TABLE INDEX OF THIRD ENTRY
19 ELSE
20 ASSUME TABLE INDEX OF FOURTH ENTRY
21 ENDF
22 IF FIRST TOKEN IS A NAME, THEN
23 INCREMENT TABLE INDEX BY TWO ENTRIES (NOW FIFTH OR SIXTH)
24 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
25 ENDF
26 :E:
27 FORM TENTH ENTRY FILE NAME FROM > AND IT EDITOR CURRENT PROCESSOR NAME
28 ASSUME TABLE INDEX OF TENTH ENTRY
29 IF FIRST TOKEN = ?, THEN
30 IF PROMPT LENGTH = 0, THEN
31 CHANGE TABLE INDEX TO SEVENTH ENTRY
32 COPY FILE NAME FROM ENTRY TEN TO ENTRY EIGHT
33 ELSE
34 CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY IT EDITOR
35 ENDF
36 ELSE
37 IF PROMPT LENGTH = 0, THEN
38 CHANGE TABLE INDEX TO NINETH ENTRY
39 ENDF
40 ENDF
41 END CASE
42 EXIT SETUP
43 :ERROR2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
44 1 END SETUP
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:N</td>
<td>655</td>
</tr>
<tr>
<td>:A</td>
<td>440</td>
</tr>
<tr>
<td>ALPHA</td>
<td>274</td>
</tr>
<tr>
<td>:R</td>
<td>443</td>
</tr>
<tr>
<td>:C</td>
<td>446</td>
</tr>
<tr>
<td>COMFUL</td>
<td>511</td>
</tr>
<tr>
<td>COMMA</td>
<td>265</td>
</tr>
<tr>
<td>:D</td>
<td>454</td>
</tr>
<tr>
<td>DBL</td>
<td>401</td>
</tr>
<tr>
<td>DCOL</td>
<td>314</td>
</tr>
<tr>
<td>DECPNT</td>
<td>330</td>
</tr>
<tr>
<td>DIGIT</td>
<td>293</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>485</td>
</tr>
<tr>
<td>:E</td>
<td>457</td>
</tr>
<tr>
<td>EORD</td>
<td>349</td>
</tr>
<tr>
<td>:ERROR2</td>
<td>680</td>
</tr>
<tr>
<td>:ERROR3</td>
<td>630</td>
</tr>
<tr>
<td>:F</td>
<td>464</td>
</tr>
<tr>
<td>INTEGR</td>
<td>379</td>
</tr>
<tr>
<td>INVAL</td>
<td>505</td>
</tr>
<tr>
<td>:LOOP</td>
<td>92</td>
</tr>
<tr>
<td>OVFLOW</td>
<td>519</td>
</tr>
<tr>
<td>QUOTE</td>
<td>477</td>
</tr>
<tr>
<td>READSE</td>
<td>131</td>
</tr>
<tr>
<td>REAL</td>
<td>390</td>
</tr>
<tr>
<td>REPET</td>
<td>412</td>
</tr>
<tr>
<td>SCARS</td>
<td>421</td>
</tr>
<tr>
<td>SETUP</td>
<td>637</td>
</tr>
<tr>
<td>XCOM</td>
<td>87</td>
</tr>
<tr>
<td>XTLAN</td>
<td>230</td>
</tr>
<tr>
<td>XTPRM</td>
<td>592</td>
</tr>
<tr>
<td>:E</td>
<td>646</td>
</tr>
<tr>
<td>:X</td>
<td>648</td>
</tr>
<tr>
<td>:/</td>
<td>663</td>
</tr>
</tbody>
</table>

EXIT F.PDLIST
FORTRAN CALLING PROCEDURE

CALL XUDBG (LU, ID)

**EXTERNAL**

[XUDBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY]

[DUMP AND/OR MODIFICATION FACILITY]

**INPUT (CALLING SEQUENCE)**

[XUDBG IS A LOGICAL UNIT TO INTERACT WITH IN INVOKING XUDBG OPTIONS]

[ID] - THREE WORD ASCII ARRAY USED AS A HEADER TO IDENTIFY XUDBG CALLER

[INPUT (INTERACTIVE)]

[OPTION (S,M,E): - ONE ASCII CHARACTER IDENTIFYING XUDBG OPTION]

[S = SNAP OUT (DUMP) MEMORY]

[M = MODIFY MEMORY]

[E = EXIT XUDBG]

[START:] - OCTAL MEMORY ADDRESS IN THE USER'S MAP OF FIRST WORD TO BE SNAPPED OR MODIFIED

[STOP:] - OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPPED

[OUTPUT UNIT:] - LOGICAL UNIT NUMBER TO WHICH SNAP IS TO BE WRITTEN

[VALUES:] - ARRAY OF OCTAL VALUES TO BE STORED IN MEMORY BEGINNING AT LOCATION "START", WHEN INPUT "VALUES" MUST NOT EXCEED 50 CHARACTERS. NULL FIELDS, I.E., SUCCESSIVE COMMAS, INDICATE WORDS OF ZERO TO BE STORED.

**OUTPUT (TO 'LU')**

[HEADER] - "*** XUDBG FROM IDID" (SEE INPUT)

[PROMPTS] - (SEE INPUT)

**OUTPUT (TO 'OUTPUT UNIT')**

[HEADER] - "*** XUDBG FROM IDID"

[SNAP] - 102 WORD DUMP FORMATTED LINE (SEE XUDPL)

**INTERNAL**

[BASE] - NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE

[CLASS] - CLASS I/O NUMBER FOR TERMINAL INPUT

[LENGTH] - LENGTH OF CHARACTER STRING BEING MANIPULATED

[LIN] - EIGHT WORD BUFFER OF WORD TO BE SNAPPED

[LU] - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET

[LGO] - LOGICAL UNIT FOR SNAP OUTPUT

[OPTN] - ONE CHARACTER EXECUTION OPTION CODE

[ORIGIN] - REFERENCE POINT FOR MEMORY OFFSET COMPUTATION

[ORG] - ADDRESS OF 'ORIGIN'

[STP] - 'ORIGIN' RELATIVE END OF DATA TO BE SNAPPED

[START] - 'ORIGIN' RELATIVE BEGINNING OF MEMORY TO BE ACCESSED

[RTH] - RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND CONVERSION'

**NOTES**

[CD0 CD0 CD0 CD0 CD0]

[CD0 CALL XUDBG (LU, ID) CD0 CD0 CD0]

[CD0 XUDBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY CD0]

[CD0 DUMP AND/OR MODIFICATION FACILITY CD0]

[CD0 INPUT (CALLING SEQUENCE) CD0]

[CD0 LU - LOGICAL UNIT TO INTERACT WITH IN INVOKING XUDBG OPTIONS CD0]

[CD0 ID - THREE WORD ASCII ARRAY USED AS A HEADER TO IDENTIFY XUDBG CALLER CD0]

[CD0 INPUT (INTERACTIVE) CD0]

[CD0 OPTION (S,M,E): - ONE ASCII CHARACTER IDENTIFYING XUDBG OPTION CD0]

[CD0 S = SNAP OUT (DUMP) MEMORY CD0]

[CD0 M = MODIFY MEMORY CD0]

[CD0 E = EXIT XUDBG CD0]

[CD0 START:] - OCTAL MEMORY ADDRESS IN THE USER'S MAP OF FIRST WORD TO BE SNAPPED OR MODIFIED CD0]

[CD0 STOP:] - OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPPED CD0]

[CD0 OUTPUT UNIT:] - LOGICAL UNIT NUMBER TO WHICH SNAP IS TO BE WRITTEN CD0]

[CD0 VALUES:] - ARRAY OF OCTAL VALUES TO BE STORED IN MEMORY BEGINNING AT LOCATION "START", WHEN INPUT "VALUES" MUST NOT EXCEED 50 CHARACTERS. NULL FIELDS, I.E., SUCCESSIVE COMMAS, INDICATE WORDS OF ZERO TO BE STORED. CD0]

[CD0 OUTPUT (TO 'LU') CD0]

[CD0 HEADER] - "*** XUDBG FROM IDID" (SEE INPUT) CD0]

[CD0 PROMPTS] - (SEE INPUT) CD0]

[CD0 OUTPUT (TO 'OUTPUT UNIT') CD0]

[CD0 HEADER] - "*** XUDBG FROM IDID" CD0]

[CD0 SNAP] - 102 WORD DUMP FORMATTED LINE (SEE XUDPL) CD0]

[CD0 BASE] - NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE CD0]

[CD0 CLASS] - CLASS I/O NUMBER FOR TERMINAL INPUT CD0]

[CD0 LENGTH] - LENGTH OF CHARACTER STRING BEING MANIPULATED CD0]

[CD0 LIN] - EIGHT WORD BUFFER OF WORD TO BE SNAPPED CD0]

[CD0 LU] - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET CD0]

[CD0 LGO] - LOGICAL UNIT FOR SNAP OUTPUT CD0]

[CD0 OPTN] - ONE CHARACTER EXECUTION OPTION CODE CD0]

[CD0 ORIGIN] - REFERENCE POINT FOR MEMORY OFFSET COMPUTATION CD0]

[CD0 ORG] - ADDRESS OF 'ORIGIN' CD0]

[CD0 STP] - 'ORIGIN' RELATIVE END OF DATA TO BE SNAPPED CD0]

[CD0 START] - 'ORIGIN' RELATIVE BEGINNING OF MEMORY TO BE ACCESSED CD0]

[CD0 RTH] - RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND CONVERSION' CD0]

[CD0 NOTES CD0]

[CD0 CALL XUDBG (LU, ID) CD0 CD0 CD0 CD0 CD0]
101 BEGIN XUG8
102 WRITE SNAP HEADER TO Terminal
103 DO UNTIL OPTION IS EXIT (E)
104 PROMPT TERMINAL FOR OPTION
105 IF OPTION IS NOT EXIT (E)
106 THEN
107 PERFORM PROMPT AND CONVERSION FOR STARTING ADDRESS
108 IF OPTION IS MODIFY (%)
109 THEN
110 PERFORM PROMPT AND CONVERSION FOR VALUES AND STORE IN MEMORY
111 ELSE
112 PERFORM PROMPT AND CONVERSION FOR ENDING ADDRESS
113 PERFORM PROMPT AND CONVERSION FOR OUTPUT UNIT
114 WRITE SNAP HEADER TO OUTPUT UNIT
115 DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
116 CALL XUG9 TO FORMAT LINE
117 OUTPUT LINE
118 END DO
119 ENDIF
120 END DO
121 END XUG8
122 BEGIN PROMPT AND CONVERSION
123 ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
124 RETRIEVE RESPONSE
125 CLEAR SUM
126 DO FOR EACH CHARACTER
127 SHIFT SUM AND ADD NEXT DIGIT
128 END DO
129 END PROMPT AND CONVERSION
FORTRAN CALLING PROCEDURE

158 1 C00  CALL XUDPL (ADDRESS, LINE, BUFFER)
159 1 C00
160 1 C00
161 1 C00
162 1 C00
163 1 C01
164 1 C01  PRODUCE AN OCTAL AND ASCII PRINT FORMATTED MEMORY DUMP LINE IMAGE
165 1 C01
166 1 C01
167 1 C02  INPUT
168 1 C02  ADDRESS - TWO WORD INTEGER ARRAY CONTAINING THE ABSOLUTE AND RELATIVE ADDRESS TO BE FORMATTED WITH THE LINE
169 1 C02  LINE - EIGHT WORD ARRAY TO BE CONVERTED TO OCTAL AND ASCII
170 1 C02  BUFFER - FIFTY-ONE WORD BUFFER TO HOLD FORMATTED PRINT LINE. MUST BE BLANKED PRIOR TO FIRST CALL TO XUDPL AND NOT STORED INTO BETWEEN CALLS TO XUDPL.
171 1 C02
172 1 C02
173 1 C02
174 1 C02
175 1 C02
176 1 C02
177 1 C03  OUTPUT - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE
178 1 C03
179 1 C03
180 1 C03
181 1 C03
182 1 C03
183 1 C03
184 1 C03
185 1 C03
186 1 C05  NOTES
187 1 C05
188 1 C05
189 1 C05
190 1 *
191 1 *
192 1 *
193 1 *
194 1 BEGIN XUDPL
195 2 C06  CALL XRO6 TO CONVERT EACH WORD OF ADDRESS TO OCTAL
196 3 C06  DO FOR EACH WORD OF LINE
197 4 C06  CALL XRO6 TO CONVERT WORD TO OCTAL
198 5 C06  DO FOR EACH BYTE OF WORD
199 6 C06  IF BYTE < 40 OR BYTE > 1368
200 7 C06  THEN
201 8 C06  REPLACE BYTE WITH ASCII PERIOD
202 9 C06  ENDIF
203 10 C06  END DB
204 11 C06  ENDDO
205 1 C06 END XUDPL
CD1  FORTRAN MAIN PROGRAM XUFNT IS SCHEDULED BY XUDMP TO PRINT
CD1  A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD1
CD1  ICNTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
CD1  LRU NUMBER OF WHERE DUMP IS ON DISK
CD1  TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
CD1  CONTAINING THE DUMP
CD1
CD1  OUTPUT
CD1  FORMATTED DUMP TO L6 6
CD1  EXTERNAL REFERENCES
CD1  EXEC
CD1  BMPAR
CD1  XRCPAR
CD1  XRNV
CD1  XUDPL

BEGIN XUFNT
CALL BMPAR TO GET L6 AND STARTING TRACK NOS.
READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
.COUNT OF ID-SEGS TO BE DUMPED
.UP TO 7 ID-SEGMENT ADDRESSES
.LOW AND HIGH BASE PAGE ADDRESSES
.LOW AND HIGH MAIN MEMORY ADDRESSES
.POINT TO 2ND SECTOR OF DUMP DATA
.DO UNTIL ALL ID-SEGS PRINTED
.DO UNTIL A EIGHT-WORD LINES PRINTED
.PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
ENDDO
INCREMENT TO NEXT SECTOR OF DUMP DATA
ENDDO
COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
PERFORM COMPARE AND PRINT FUNCTION
COMPUTE TRACKS OF MAIN MEMORY TO BE READ
.DO UNTIL TRACKS ARE READ
.READ NEXT TRACK FROM DISK
.COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP
.PERFORM COMPARE AND PRINT FUNCTION
ENDDO
RELEASE THE GLOBALLY ALLOCATED TRACKS
EXIT XUFNT
BEGIN COMPARE AND PRINT FUNCTION
.DO UNTIL N LINES PROCESSED
.IF NOT 1ST LINE, THEN
.CALL XRCPAR TO COMPARE WITH PREVIOUS LINE
.IF LINES ARE IDENTICAL, THEN
.IF THIS IS 1ST OF A SERIES, THEN
.WRITE 'DUPLICATE LINE'
.ENDIF
ELSE
.CALL XUDPL TO FORMAT THE DUMP LINE
.WRITE FORMATTED DUMP LINE
.ENDIF
ELSE
.CALL XUDPL TO FORMAT THE DUMP LINE
.WRITE FORMATTED DUMP LINE
.ENDIF
ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
2  * #1 ENTRY XVABN
3  * #1 TYPE 1A ROUTINE TO ABORT CURRENT ID AND
4  * #1 REQUEST A DUMP FOR THE ID.
5  * #1 ENTRY XVABN
6  * #1 TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
7  * #1 THE REQUESTED DATA AREAS AND THE CURRENT
8  * #1 ID SEGMENT AND ITS FATHER CHAIN SEGMENTS
9  * *
10  1 BEGIN XVABN ABEND
11  2 *#2 INPUTS: NONE
12  3 *#3 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
13  4 * * EXTERNALS: SABRT, SABRE, $XEN, $LIBR
14  5  CALL SLIBR  REENTRANT ROUTINE CALL
15  6  GET CURRENT ID FROM XRT (LOCATION 1717)
17  2  INCREMENT TO ADDRESS TECHNIRES
18  3  GET ADDRESS BOUNDARIES
19  4  STORE := XVABN PARAMETER LIST
20  5  PERFORM XVABN DUMP THIS PARTITION
21  2  CALL SABRT RETRY THIS PARTITION
22  3  CALL SABRE FREE CURRENT REENTRANT DATA BLOCK
23  4  EXIT TO :XEN  GO TO DISPATCHER
24  5  END XVABN
25  6  *
26  7  *
27  8  *
28  9  *
30  1 BEGIN XVABN
31  2 #1 ENTRY XVABN
32  3 #2 INPUTS: ADDRESS OF DOUBLE WORD CONTAINING START AND END ADDRESSES
33  4 #2 DEFINING AREAS TO BE DUMPED
34  5 #3 OUTPUTS: OUTPUTS TO DISK
35  6 #3 HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW
36  7 #3 CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS
37  8 #3 MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION
38  9 #3 MEMORY IMAGE AS SPECIFIED BY INPUT ADDRESS RANGE
39  0 #3 SCHEDULES XVABN TO FORMAT DATA FROM DISK TO PRINT
40  1 *
41  2 #3 EXTERNALS: SLIBR, SLIBX, EXC, XVABN
42  3 CALL SLIBR TO NOTIFY OF REENTRANCY AND BECOME PRIVILEGED
43  4 RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH
44  5 CALL SLIBX TO BECOME NON-PRIVILEGED
45  6 CALL EXC FOR A GLOBAL ALLOCATION OF DISK
46  7 CALL EXEC TO WRITE HEADER DATA USING CLASS 1/0
47  8 DO UNTIL ALL ID SEGMENTS PROCESSED
48  9 WRITE ID SEGMENT TO DISK
49  0 END XVABN
50  1 WRITE BASE PAGE SEGMENT TO REMAINING PROTON OF THIS TRACK
51  2 DO UNTIL END ADDRESS REACHED
52  3 WRITE ONE SECTOR OF DATA
53  4 ENDDO
54  5 SCHEDULE XVABN WITH LU AND STARTING TRACK NOS.
55  6 CALL SLIBX TO RETURN FROM REENTRANT ROUTINE
56  7 END XVABN
58 1 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 XVPAW AND XVSTB
62 1 * INPUTS
63 1 * FROM AN ASSOCIATED TASK
64 1 * CALL XVPAW(PARMS)
65 1 * ASSEMBLY FORM
66 1 * JSB XVPAW
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 * 0 = NORMAL TERMINATION (P2-P5 NOT USED)
72 1 * 1 = WORK AREA REQUEST (P2-P5 NOT USED)
73 1 * 2 = EXECUTE A SEQUENCE TABLE
74 1 * 3 = RESET SEQUENCE POINTERS
75 1 * 4 = P2 HAS SEQUENCE NUMBER (P2-P5 NOT USED)
76 1 * 5 = TERMINATE SEQUENCE (P2-P5 NOT USED)
77 1 * 9 = TERMINATE FDS FUNCTION (P2-P5 NOT USED)
78 1 * -32767 = ABNORMAL TERMINATION OF ASSOCIATED TASK
79 1 * FROM AN FDS MANAGER
80 1 * ASSEMBLY FORM
81 1 * JSB XVPAW
82 1 * DEF (RETURN POINT)
83 1 * OCT 0
84 1 * DEF PARMS A(FDS MANAGER RESPONSE)
85 1 * OUTPUTS
86 1 * TO AN FDS MANAGER
87 1 * REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
88 1 * RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
89 1 * CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
90 1 * MANAGER IS ACTIVATED VIA SLIST
91 1 * TO AN ASSOCIATED TASK
92 1 * RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
93 1 * RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
94 1 * CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
95 1 * ASSOCIATE TASK IS ACTIVATED VIA SLIST
1 BEGIN XVPMAN
2 * DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
3 * FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
4  SET TARGET-ID FROM CURRENT-ID-SEGMENT
5  DO WHILE FATHER-ID-NUMBER NE. 0 OR FATHER IS WAITING
6  COMPUTE FATHER-ID-SEGMENT FROM FATHER-ID-NUMBER IN TARGET-ID
7  PERFORM MGRFND(FATHER-ID-SEGMENT,COUNT)
8  EXITIF COUNT .GT. 0
9  SET TARGET-ID TO FATHER-ID-SEGMENT
10 ENDDO
11  IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
12 CALL $SYM (12,*XVO3,SEGMENT-NAME) -*XVO3,NAME* REQUESTING PROG
13 PERFORM PUMP
14 EXIT TO $LIBX TO ENABLE AND REDISPATCH
15 ENDF
16  SET AWAKEN-ID FROM FSD-ENTRY $SYM
17 GET REQUEST PAMRS MOVE INTO ID-SEGMENT
18  SET SBTAT FROM CURRENT-ID
19 FND XVPMAN
20 *
21 *
22 *
23 SEG: XVPMAN
24 * DETERMINE IF CALLER IS A VALID FDS MANAGER
25 * PERFORM MGRFND (CURRENT-ID,COUNT)
26 * COUNT WILL BE 0 FOR NO MATCH.
27 * COUNT NOT EQUAL ZERO IMPLIES A MATCH
28 * AND FSD-ENTRY HAS MATCHING FSTAB ENTRY ADDRESS
29 * IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
30 CALL $SYM (12,*XVO1,SEGMENT-NAME) -*XVO1 PROGRAM* REQUESTING PROG.
31 PERFORM PUMP
32 EXIT TO :SKEP THE DISPATCHER
33 ENDF
34 *
35 FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
36 SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
37 IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
38 CALL $SYM (12,*XVO2,SEGMENT-NAME) -*XVO2 PROGRAM* ASSOCIATED PROG.
39 PERFORM PUMP
40 EXIT TO :SKEP THE DISPATCHER
41 ENDF
42 *
43 MOVE FDS MANAGERS INPUT PAMRS TO ASSOCIATED TASK ID SEGMENT
44 MOVE PAMRS TO ID-SEGMENT WORDS 2-6
45 END XVPMAN
172 1 BEGIN PDUMP
173 2 * SET A CALL TO XVABN AT CALLER'S SUSPEND POINT
174 3 END PDUMP
175 1 *
176 1 *
177 1 *
178 1 *
179 1 *
180 1 *
181 1 *
182 1 BEGIN MGFRND INPUT IS TARGET-ID
183 2 GET XVSTB TABLE OF ACTIVE FDS MANAGERS
184 3 SET COUNT TO NUMBER IN TABLE
185 4 SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
186 5 DO WHILE COUNT .LT. 0
187 6 EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMENT
188 7 SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
189 8 SET COUNT = COUNT - 1
190 9 ENDDO
191 10 * FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
192 11 * COUNT = 0 MEANS NO MATCH
193 12 END MGFRND
194 1 *
195 1 *
196 1 *
197 1 *
198 1 BEGIN XVSTB
199 2 N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSTB
200 3 STBNM DEF N NUMBER-IN-TABLE
201 4 STBC DEF O NUMBER-ACTIVE
202 5 STBNR DEF 0 STB RESOURCE NUMBER
203 6 STDES EQU * ENTRY START
204 7 STBLU DEF 0 LUC(STBL)'
205 8 STBLA DEF 0 LUC(STBLA)
206 9 STBMG DEF 0 A(FDS-MANAGER-ID-SEGMENT)
207 10 STBEX DEF 0 A(FDS-EXECUTIVE-ID-SEGMENT)
208 11 STBEC DEF 0 CLASS NUMBER FOR EXEC
209 12 STBAT DEF 0 (CURRENT-ASSOCIATED-TASK:ID)
210 13 STBPC DEF 0 CLASS-NUMBER FOR PROCESSOR
211 14 STBEP DEF 0 ENTRY SAVE AREA
212 15 STBEE EQU * ENTRY "NO'
213 16 STBEL EQU STBEE-STBES ENTRY LENGTH
214 17 STBNE EQU N NUMBER-OF-ENTRIES
215 18 ORG STBES
216 19 REP N DO FOR N ENTRIES
217 20 REP STBEL DEFINE STORAGE WORDS FOR AN ENTRY
218 21 DEF O
219 1 END XVSTB
C**************
C       FORTRAN CALLING PROCEDURE:
C       CALL XIAUT
C
C**************
C       XIAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE
C
C**************
C       INPUTS IN COMMON:
C       XE(5) MASSTA, XE(10) SERRSTR, XE(11) SEREND, XE(12) SEPTR,
C       XE(140) TABEND, XE(1) HOPROC, XB(2) LIBD,
C       XB(249) SERNO, XB(250) SLEN, XB(251) SENTRY
C
C**************
C       OUTPUTS IN COMMON:
C       XE(5) MASSTA, XS(1), FLGTAB
C
C**************
C       COMMON USED:
C       EQUIVALENCE (XE(5), MASSTA)
C       +(XE(10), SERRSTR), (XE(11), SEREND)
C       +(XE(12), SEPTR), (XE(140), TABEND),
C       +(XB(5), NPSPGC), (XB(2), LIBD ),
C       +(XB(235), RESIND), (XE(245), SERNO ),
C       +(XB(250), SLEN), (XB(251), SENTRY),
C       +(XS(15), FLGTAB)
C
C**************
C       FBS ROUTINES CALLED:
C       XECPR, XEXT, XREMOT, XREMVS,
C       XXE, XISTO, XITMP
C
C**************
C       RTE ROUTINES CALLED:
C       I00
C
C**************
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 ERREXIT IF PROCESSOR NOT FOUND TO :ERR1:
55  7 STUFF INTERFACE TABLE 6IT AND VERSION INTO SEQUENCE TABLE ENTRY
56  8 ENDDO
57  9 CALL XXSTO TO STORE REVISED SEQUENCE TABLE IN ARD AS &SERTB
58 10 ELSE IF AM BEING REENTERED FROM INT.
59 11 CALL XXPMP TO SET UP TEMPORARY EN-AR W:IN &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 ENDOF
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 ENDOF
69 21 1 EXIT XXAUT
70 2 :ERR1:
71 2 CALL ZRMSG TO DISPLAY INVALID PROCESSOR NAME
72 1 END XXAUT
**FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:**

**CALL XELS (IXCNT)**

**IXCNT** 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:**

**IXCNT** - (integer, 3 words) array containing the name "IXCNT" used by XELS to call EXEC ESL to load the execution controller segment.

**Internal Variables:**

**MODE** - (integer, 1 word) mode in which the execution controller was called.

- 0 - Manual
- 1 - Semi-automatic
- 2 - Automatic with trace
- 3 - Automatic

**Common Used:**

EQUIVALENCE (XED5, NASDA)

**FDS Routines Used:**

XERTW, XEXT, XRMSG, XXHAR

**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:, :AUTT:, :AUTO:)
122 3 :MANU: CALL XIMAN
123 3 :SENI: CALL XISEN
124 3 :AUTT: CALL XIMSG TO DISPLAY INVALID OPTION
125 3 :AUTO: CALL XIAUT
126 2 ENDCASE
127 2 SET MASTER STATE TO DIRECTIVE LEVEL
128 2 CALL XERM TO RETURN TO XEXEC **AND RETURN**
129 2 DUMMY CALL XEXEC
130 1 END XICNT
**FORTRAN Calling Sequence:**

CALL XXDEC (RETC)

**XXDEC Decodes a response of processor name (INT table name) into a sequence table entry.**

**Inputs in COMMON:**

- XE(85) tokens, XE(145) COMBUF, XE(1) NOPROC, XE(2) LIBD

**Outputs in Calling Sequence:**

- RETC - return code (0 is normal return)

**Outputs in COMMON:**

- XE(16) PRCNAM, XE(251) SEQTAB

**COMMON USED:**

- EQUIVALENCE (XE(16), PRCNAM),
- (XE(1), PRCNAM),
- (XE(144), TOKEH5),
- (XE(145), COMBUF),
- (XE(1), NOPROC), (XE(2), LIBD),
- (XE(251), SEQTAB)

**FDs Routines Used:**

- XRCPR, XREXT, XRMOV, XRMSG

**RTE Routines Used:**

- IANID
BEGIN XIDEC

1 BUILD XIDEC RETURN CODE TO ZERO
2 ERROR IF FIRST DECENT IS NOT FOR PROCESSOR NAME TO ERROR
3 ERROR IF NAME NOT FOUND TO ERROR
4 IP INTERFACE TABLE NAME INTO XIDEC ENTRY
5 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
6 ERROR IF LAST TOKEN IS NOT EOS TO ERROR
7 ERROR IF INTERFACE TABLE IS NOT SPLIT INTO GROUPS TO ERROR
8 SET INTERFACE TABLE IN SEQUENCE ENTRY TO $\text{EXIT}$
9 END XIDEC

END XIDEC

10 BUILD XIDEC RETURN CODE TO ERROR

5-310
**FORTRAN CALLING PROCEDURE:**

CALL XXDEF

**INPUTS FROM COMMON:**

PRCHNM - (INTEGER, 3 WORDS) NAME OF PROCESSOR IN SERTAB
    FOR WHICH NO INTERFACE TABLE WAS SUPPLIED

**INTERNAL VARIABLES:**

DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE

**COMMON USED:**

EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA),
    + (XE(15), FRMNM), (XE(16), PRCHNM),
    + (X(19), RERPTR), (XE(20), RERBUF),
    + (XE(140), CRM ),
    + (X(16), TMPTAB), (X(14), DEFNM),
    + (X(18), LEN1 ), (X(19), LEN2 ),
    + (X(20), IDC ), (X(200), RETC )

**RTE ROUTINES USED:**

CLOSE, EXEC, KCVT, OPEN, READF

**FDS ROUTINES USED:**

XERMT, XEREQ, XREXT, XRMOV, XRMSC, XRRC, XRUPK
BEGIN XDEF
CREATE DEFAULT INTERFACE TABLE NAME FROM PROCESSOR NAME
CALL OPEN TO OPEN FILE
ERREXIT IF OPEN ERROR TO :FILERR:
CALL READ TO READ RECORD 1
ERREXIT IF READ ERROR TO :FILERR:
IF THERE IS LITERAL DATA THEN
CALL READ TO READ RECORD 2
ERREXIT IF READ ERROR TO :FILERR:
ENDIF
CALL CLOSE TO CLOSE FILE
ERREXIT IF CLOSE ERROR TO :FILERR:
CALL XREG TO ALLOCATE & STORE GINTAB
ERREXIT IF RETURN CODE IS NOT ZERO TO :NGRERR:
IF DEFAULT TABLE IS INCOMPLETE THEN
SET INTERFACE TABLE NAME IN XE TO GINTAB
CALL XERMT TO INPLICITLY CALL INT EDITOR **NO RETURN**
ENDIF
EXIT XDEF

FILERR:
CALL CLOSE TO CLOSE FILE
CALL XRESG TO DISPLAY FILE ACCESS ERROR
EXIT XDEF

NGRERR:
CALL XRESG TO DISPLAY SPACE ERROR
CALL EXEC TO FREE CLASS NUMBER
SET MASTER STATE TO DIRECTIVE LEVEL
CALL XERMT TO RETURN TO EXEC **NO RETURN**
END XDEF
PROCEDURE FOR CALLING XIXMAN:

CALL XIXMAN

XXMAN IS THE MANUAL EXECUTION CONTROLLER

INPUTS FROM COMMON:

XE(5) MASSTA, XB(249) SEENO

OUTPUTS TO COMMON:

XE(10) SERSTA, XE(11) SEEND, XB(249) SEENO,

XB(250) SERLEN, XB(251) SERTAB, XE(13) FLGTAB

COMMON USED:

EQUIVALENCE (XE(5), MASSTA), (XE(10), SERSTA),

+ (XE(11), SEEND),

+ (XB(249), SEENO), (XB(250), SERLEN),

+ (XB(251), SERTAB), (XE(13), FLGTAB)

FDS ROUTINES USED:

XEXT, XMNSG, XCTOM, XAXEx,

XXDEC, XXDEF, XXSTO, XXTMP

RTE ROUTINES USED:

IOR
BEGIN XMAN
IF ENTRY IS FROM A DIRECTIVE THEN
SET HASITA TO INDICATE RE-ENTRY
DO UNTIL PERCENT IS ENTERED

PROMPT: CALL XTCOM TO PROMPT FOR PRNAME, "ITNAME"
IF PERCENT IS NOT ENTERED THEN
EREXIT IF CR ENTERED TO: PROMPT:
CALL XXDEC TO DECODE RESPONSE
EREXIT IF INVALID RESPONSE TO: PROMPT:
SET SER #S IN "E:" ZEROS
SET #ENTRIES IN SERTAB IN XE TO 1
CALL XXSTO TO STORE SEQUENCE TABLE
IF IT NAME IN SERTAB IS &INTAB THEN
PROCESSOR REQUIRE AN IT THEN
CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
ENDIF
CALL XXEXEC TO EXECUTE SECTAB
ENDIF
ENDDO
ELSE AM BEING REENTERED FROM INTERFACE TABLE EDITOR
CALL XXMP TO SET UP TO EXECUTE A TEMPORARY TABLE
PERFORM XXM AN **NO RETURN**
ENDIF
RETURN

PROMPT: CALL XRMSC TO DISPLAY ERROR
PERFORM XMAN **NO RETURN**
END XMAN
1 BEGIN XSEM:
2 IF ENTRY IS FROM A DIRECTIVE THEN
3 SET MASTER STATE TO SAT REENTRY
4 DO FOR # ENTRIS IN SEQUENCE TABLE
5 SEARCH LABEL FOR PROCESSOR NAME
6 ERREXIT IF NOT FOUND TO :ERR:
7 MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
8 ENDIF
9 SAVE ORIGINAL ENDING SEQUENCE #
10 SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
11 SET FLSTAB TO SAT EXECUTE ENTIRE SEQUENCE
12 CALL XISTO TO STORE RESSTD IN AW.
13 ELSE (/ AN BEING REENTERED FROM INTE)
14 CALL XETMP TO SET UP TEMPORARY EXECUT'D
15 CALL XTERM 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 XSEM IF TERMINAL ENTRY WAS JUST EXECUTED
20 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
21 ENDIF
22 ENDIF
23 DO UNTIL TERMINAL ENTRY 1S EXECUTED AND IT IS NOT A REQUEST TO RESET
24 CALL XCON TO PROMPT USER WITH CURRENT ENTRY
25 EXIT XSEM IF RESPONSE 1S X
26 IF RESPONSE IS CR THEN
27 IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
28 CALL XIDET TO READ UP DEFAULT TABLE
29 ENDIF
30 SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
31 SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
32 CALL XETMP TO EXECUTE
33 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
34 ELSE
35 IF RESPONSE WAS A SEQUENCE # THEN
36 ERREXIT IF NUMBER IS ZERO TO :ERR:
37 SEARCH SEQUENCE TABLE FOR SEQUENCE #
38 ERREXIT IF NUMBER IS NOT FOUND TO :ERR:
39 SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
40 ELSE
41 IF RESPONSE IS AN EMPIRICAL THEN
42 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
43 ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
44 CALL XIDET TO DECODE PROCESSOR NAME, IT NAME RESPONSE
45 IF RESPONSE IS VALID THEN
46 CALL XISTO TO STORE OVERRISING ENTRY
47 SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
48 SET CURRENT SEQUENCE # TO FIRST ENTRY
49 ENDIF
50 ENDIF
51 ENDIF
52 ENDIF
53 ENDDO
54 EXIT XSEM
55 ERR:
56 CALL XTERM TO DISPLAY ERROR MESSAGE
57 END XSEM
CALLING PROCEDURE FOR IXSTO:

CALL IXSTO

XXSTO SETS UP TO REQUEST MANAGER TO STORE GRTDB AND HANDLES ANY ERRORS RETURNED BY THE MANAGER

INPUTS FROM COMMON:

XH(250), SERLEN, XH(251), SERTAB, XH(13), FLGTAB

OUTPUTS TO COMMON:

XE(5) MASSTA

COMMON USED:

EQUIVALENCE (XE(5), MASSTA), (XE(10), REGTPR), (XE(20), REGBUT), (XH(250), SERLEN), (XH(251), SERTAB), (XH(13), FLGTAB)

FDS ROUTINES USED:

XREQ, XRMOV, XRMSG, XERTN

RTE ROUTINES USED:

EXEC

EXEC
600 1 BEGIN XXSTO
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 TB
614 6 CALL XMOV 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 XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER
619 11 ENSIF
620 12 CALL XER TO REQUEST MANAGER TO STORE .SERSTB
621 13 IF RETURN CODE IS NOT ZERO THEN
622 14 CALL XMSG 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 XXSTO

BEGIN
**FORTRAN CALLING SEQUENCE:**

CALL XITHP

**XITHP** SETS UP A ONE ENTRY SEQUENCE TABLE USING SINTAB AND

**STORES IT IN THE AREA TO EXECUTE WHEN EXECUTION WITH A**

**TEMPORARY ENTRY IS NECESSARY**

**INPUTS FROM COMMON:**

XE(12) SEPTR, XB(250) SELEN, XB(251) SECTAB

**OUTPUTS TO COMMON:**

XE(6) SUBSTA, XB(249) SEIMO, XB(250) SELEN,

**COMMON USED:**

EQUIVALENCE (XE(6), SUBSTA), (XE(12), SEPTR),

**RDS Routines Used:**

XRED, XMNOV, XSTO

**RTE Routines Used:**

EXEC
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 ANA. ASSGN SUPPORTS EXTENDED FORTRAN TYPE MIXED-MODE EXPRESSIONS AND FUNCTIONS AND ALLOWS REPEITIVE 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 ANA:

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

INTERNAL VARIABLES:

BLANK COMMON - ASCOM 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>PARM</td>
<td>5</td>
<td>1</td>
<td>PARMS(1)= LU, PARMS(2)= DEBUG FLAGS</td>
</tr>
<tr>
<td>TOKENS</td>
<td>32</td>
<td>6</td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STWRITE</td>
<td>1</td>
<td>38</td>
<td>SYMBOL TABLE WIDTH</td>
</tr>
<tr>
<td>STLONG</td>
<td>1</td>
<td>39</td>
<td>SYMBOL TABLE LENGTH</td>
</tr>
<tr>
<td>LASTSY</td>
<td>1</td>
<td>40</td>
<td>LAST SYMBOL TABLE ENTRY Defined</td>
</tr>
<tr>
<td>SYMTAB</td>
<td>1</td>
<td>12,81</td>
<td>SYMBOL TABLE (WORDS 1-8 = TOC ENTRY OR APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR SUBSCRIBED DATA ELEMENT)</td>
</tr>
<tr>
<td>SSTRING</td>
<td>247</td>
<td>1013</td>
<td>SYMBOL STRING (EXP)</td>
</tr>
<tr>
<td>RESULT</td>
<td>4,35</td>
<td>1260</td>
<td>RESULT STACK USED DURING POST-FIX STRING EVALUATION (EACH EN-</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>C24</td>
<td>1</td>
<td>1</td>
<td>C24</td>
</tr>
<tr>
<td>Token</td>
<td>Input Priority</td>
<td>Output Priority</td>
<td>Token Length / Operator Requirements</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
<td>-----------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>EOS</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INTEGER</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>REAL</td>
<td>-</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>NAME</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>EOF</td>
<td>-</td>
<td>-</td>
<td>?</td>
</tr>
<tr>
<td>&lt;</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>&gt;</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>&lt;</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>)</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>)</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>)</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>)</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>??</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SUBSCRIPT</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Unary-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Function</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**Notes:**
- OP = OPERATOR
- LB = LEFT Bracket
- CN = CONSTANT
### FMCTBL - 7X36  ABLE CONTAINING INFORMATION FOR PROCESSING FUNCTIONS

<table>
<thead>
<tr>
<th>NAME(WORDS 1-3)</th>
<th>#OPERANDS-1</th>
<th>OUTPUT TYPE</th>
<th>FIRST OP TYPE</th>
<th>SECOND OP TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>AINT</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>ALOG</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>ALOGT</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>AMOD</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>ATAN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>ATAN2</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>COS</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DABS</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATAN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATAN2</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DBLE</td>
<td>0</td>
<td>DOUBLE</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DCOS</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DINT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DLOG</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DLOGT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DMOD</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DSIGN</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>ME</td>
</tr>
<tr>
<td>DSIN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DSRFT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DTAN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DTANH</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>0</td>
<td>REAL</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>IABS</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>IDINT</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>IFIX</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>ME</td>
</tr>
<tr>
<td>ISIGN</td>
<td>1</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MOD</td>
<td>1</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SIGN</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>ME</td>
</tr>
<tr>
<td>SIN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SGN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SGNL</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>ME</td>
</tr>
<tr>
<td>SQRT</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>TAN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>TANH</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DEXP</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>ME</td>
</tr>
</tbody>
</table>

**ASSGN**
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1 CS5</td>
<td>Routines referenced - RMPAR, XPGET, XMMOV, XI?S1, XI?S2, XPIXI</td>
</tr>
<tr>
<td>202</td>
<td>1 CS5</td>
<td>BACKUS-NAUR LANGUAGE DEFINITION</td>
</tr>
</tbody>
</table>
| 203  | 1 CS5| <ASSIGNMENT> ::= <REPLACEMENT> <RANGE>  
| 204  | 1 CS5| <REPLACEMENT> ::= <NON-NUMERIC DE> = <NON-NUMERIC DE>  
| 205  | 1 CS5| <NON-NUMERIC DE> ::= "CHARACTER STRING"  
| 206  | 1 CS5| <VARIABLE> ::= FREE DATA ELEMENT  
| 207  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 208  | 1 CS5| CHARACTER DATA ELEMENT  
| 209  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  
| 210  | 1 CS5| <EVALUATION> ::= <VARIABLE> = <EXPRESSION>  
| 211  | 1 CS5| <EXPRESSION> ::= FREE DATA ELEMENT = <EXPRESSION>  
| 212  | 1 CS5| <VARIABLE> = <SUBSCRIPT LIST> <FUNCTION LIST>, <EXPRESSION>  
| 213  | 1 CS5| <FUNCTION LIST> ::= <FUNCTION LIST>, <EXPRESSION>  
| 214  | 1 CS5| <FUNCTION> ::= <FUNCTION NAME> <LB> <FUNCTION LIST> <RB>  
| 215  | 1 CS5| <OPERAND> ::= FUNCTION NAME <LB> <FUNCTION LIST> <RB>  
| 216  | 1 CS5| <VARIABLE> = FREE DATA ELEMENT  
| 217  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 218  | 1 CS5| CHARACTER DATA ELEMENT  
| 219  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  
| 220  | 1 CS5| <VARIABLE> = <EXPRESSION>  
| 221  | 1 CS5| <EXPRESSION> ::= FREE DATA ELEMENT = <EXPRESSION>  
| 222  | 1 CS5| <VARIABLE> = <SUBSCRIPT LIST> <FUNCTION LIST>, <EXPRESSION>  
| 223  | 1 CS5| <FUNCTION LIST> ::= <FUNCTION LIST>, <EXPRESSION>  
| 224  | 1 CS5| <FUNCTION> ::= <FUNCTION NAME> <LB> <FUNCTION LIST> <RB>  
| 225  | 1 CS5| <OPERAND> ::= FUNCTION NAME <LB> <FUNCTION LIST> <RB>  
| 226  | 1 CS5| <VARIABLE> = FREE DATA ELEMENT  
| 227  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 228  | 1 CS5| CHARACTER DATA ELEMENT  
| 229  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  
| 230  | 1 CS5| <EVALUATION> ::= <VARIABLE> = <EXPRESSION>  
| 231  | 1 CS5| <EXPRESSION> ::= FREE DATA ELEMENT = <EXPRESSION>  
| 232  | 1 CS5| <VARIABLE> = <SUBSCRIPT LIST> <FUNCTION LIST>, <EXPRESSION>  
| 233  | 1 CS5| <FUNCTION LIST> ::= <FUNCTION LIST>, <EXPRESSION>  
| 234  | 1 CS5| <FUNCTION> ::= <FUNCTION NAME> <LB> <FUNCTION LIST> <RB>  
| 235  | 1 CS5| <OPERAND> ::= FUNCTION NAME <LB> <FUNCTION LIST> <RB>  
| 236  | 1 CS5| <VARIABLE> = FREE DATA ELEMENT  
| 237  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 238  | 1 CS5| CHARACTER DATA ELEMENT  
| 239  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  
| 240  | 1 CS5| <EVALUATION> ::= <VARIABLE> = <EXPRESSION>  
| 241  | 1 CS5| <EXPRESSION> ::= FREE DATA ELEMENT = <EXPRESSION>  
| 242  | 1 CS5| <VARIABLE> = <SUBSCRIPT LIST> <FUNCTION LIST>, <EXPRESSION>  
| 243  | 1 CS5| <FUNCTION LIST> ::= <FUNCTION LIST>, <EXPRESSION>  
| 244  | 1 CS5| <FUNCTION> ::= <FUNCTION NAME> <LB> <FUNCTION LIST> <RB>  
| 245  | 1 CS5| <OPERAND> ::= FUNCTION NAME <LB> <FUNCTION LIST> <RB>  
| 246  | 1 CS5| <VARIABLE> = FREE DATA ELEMENT  
| 247  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 248  | 1 CS5| CHARACTER DATA ELEMENT  
| 249  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  
| 250  | 1 CS5| <EVALUATION> ::= <VARIABLE> = <EXPRESSION>  
| 251  | 1 CS5| <EXPRESSION> ::= FREE DATA ELEMENT = <EXPRESSION>  
| 252  | 1 CS5| <VARIABLE> = <SUBSCRIPT LIST> <FUNCTION LIST>, <EXPRESSION>  
| 253  | 1 CS5| <FUNCTION LIST> ::= <FUNCTION LIST>, <EXPRESSION>  
| 254  | 1 CS5| <FUNCTION> ::= <FUNCTION NAME> <LB> <FUNCTION LIST> <RB>  
| 255  | 1 CS5| <OPERAND> ::= FUNCTION NAME <LB> <FUNCTION LIST> <RB>  
| 256  | 1 CS5| <VARIABLE> = FREE DATA ELEMENT  
| 257  | 1 CS5| <NON-NUMERIC DE> ::= FREE DATA ELEMENT <SUBSCRIPT LIST>  
| 258  | 1 CS5| CHARACTER DATA ELEMENT  
| 259  | 1 CS5| CHARACTER DATA ELEMENT <SUBSCRIPT LIST>  

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
DBDSP - DATA BOX DISPLAY PROCESSOR

- SCHEDULED BY FDS

DBDSP 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

XIDP1 - READS ORDR A, VALIDATES NAMES IN INTERFACE TABLE AGAINST NAMES IN DATABOX SCAN. WITH SUBROUTINE XIDMK IT DEVELOPS CONSTRAINT MASKS FOR ARRAYS

XIDP2 - PROMPTS USER IF REQUIRED AND OUTPUTS REQUESTED PAGE. ARRAYS OF UP TO TWO VARIABLES TO THE SPECIFIED LU DEVISE WITH SUBROUTINE XIDOT

DBDSP MERELY CALLS THESE OVERLAYS IN THE PROPER ORDER AND EXITS.

INPUTS TO DBDSP FROM INTERFACE TABLE

DATABASE - DATA BOX FILE NAME

NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER

REXP - DISPLAY VARIABLE SCALE LIST SET UP BY USER

VIDEFD - CONSTRAINT VARIABLE DEFINITION LIST

INPUTS TO DBDSP FROM ORDR 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

(1M) - NO OF ENTRIES IN SUMMARY TABLE

(2) - X SCAN VARIABLE (6 CHAR)

(5) - X FIRST SUBSCRIPT (INT OR ZERO)

(6) - XSECOND SUBSCRIPT (INT OR ZERO)

(7) - X UNITS (6 CHAR)

(10) - X CENTROID (REAL)
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>338:1 CO 2</td>
<td>(12) - X INCREMENT (REAL)</td>
</tr>
<tr>
<td>339:1 CO 2</td>
<td>(14) - X NUMBER OF STEPS (INTEGER 1-5)</td>
</tr>
<tr>
<td>340:1 CO 2</td>
<td>(15) - YSCAN VARIABLE (6 CHAR)</td>
</tr>
<tr>
<td>341:1 CO 2</td>
<td>(18) - Y FIRST SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>342:1 CO 2</td>
<td>(19) - Y SECOND SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>343:1 CO 2</td>
<td>(20) - Y UNITS (6 CHAR)</td>
</tr>
<tr>
<td>344:1 CO 2</td>
<td>(23) - Y CENTROID (REAL)</td>
</tr>
<tr>
<td>345:1 CO 2</td>
<td>(25) - Y INCREMENT (REAL)</td>
</tr>
<tr>
<td>346:1 CO 2</td>
<td>(27) - Y NUMBER OF STEPS (INTEGER 1-5)</td>
</tr>
<tr>
<td></td>
<td>DEPENDANT VARIABLE NAME AND UNITS IN RECORDS 32,48 AND 5</td>
</tr>
<tr>
<td></td>
<td>SUMMARY TABLE RECORDS</td>
</tr>
<tr>
<td></td>
<td>- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH</td>
</tr>
<tr>
<td></td>
<td>DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES)</td>
</tr>
<tr>
<td></td>
<td>INCLUDING ERROR FLAG WHICH IS FIRST VALUE</td>
</tr>
<tr>
<td></td>
<td>IN SUMMARY TABLE</td>
</tr>
<tr>
<td>380:1 CO 2</td>
<td>OUTPUT FROM DDOSP</td>
</tr>
<tr>
<td>381:1 CO 2</td>
<td>- DISPLAY IS OUTPUT TO LU IDENTIFIED IN THE INTERFACE TABLE</td>
</tr>
</tbody>
</table>

**DDOSP**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>359:1 CO 2</td>
<td>CDBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPRM</td>
</tr>
<tr>
<td>360:1 CO 2</td>
<td>DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DDOSP</td>
</tr>
<tr>
<td>361:1 CO 2</td>
<td>DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS</td>
</tr>
<tr>
<td>362:1 CO 2</td>
<td>JDX1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR</td>
</tr>
<tr>
<td>363:1 CO 2</td>
<td>JDX2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR</td>
</tr>
<tr>
<td>364:1 CO 2</td>
<td>JSAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)</td>
</tr>
<tr>
<td>365:1 CO 2</td>
<td>JISN1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)</td>
</tr>
<tr>
<td>366:1 CO 2</td>
<td>JISN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)</td>
</tr>
<tr>
<td>367:1 CO 2</td>
<td>JIBUF - NAME OF COMMON AREA USED FOR INTERFACE TABLE</td>
</tr>
<tr>
<td>368:1 CO 2</td>
<td>JTRN - LENGTH OF JTRN IN 16-BIT WORDS - XPRM</td>
</tr>
<tr>
<td>369:1 CO 2</td>
<td>LDX1 - LENGTH IN WDS OF CHAR STRING USED FOR USER PROMPT - XPRM</td>
</tr>
<tr>
<td>370:1 CO 2</td>
<td>MXASK - ARRAY CONTAINING MASKS FOR UP TO 8 CONSTRAINTS (4 X 121 SIZE)</td>
</tr>
<tr>
<td>371:1 CO 2</td>
<td>NAME - NAME LIST FOR VARIABLES SCANNED BY SCANENDSCM</td>
</tr>
<tr>
<td>372:1 CO 2</td>
<td>NC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)</td>
</tr>
<tr>
<td>373:1 CO 2</td>
<td>NREL - LIST OF CONSTRAINT RELATIONS INPUT BY USER</td>
</tr>
<tr>
<td>374:1 CO 2</td>
<td>NCVARL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)</td>
</tr>
<tr>
<td>375:1 CO 2</td>
<td>NOVARL - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)</td>
</tr>
<tr>
<td>376:1 CO 2</td>
<td>NOVRC - NUMBER OF DEP DISP VAR IN NOVARL LIST (INTEGER)</td>
</tr>
<tr>
<td>377:1 CO 2</td>
<td>NOVRL - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P</td>
</tr>
<tr>
<td>378:1 CO 2</td>
<td>MXERR - SET OF INDICATORS FOR CONSTRAINTS VIOLATED = NOT D</td>
</tr>
<tr>
<td>379:1 CO 2</td>
<td>MSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)</td>
</tr>
<tr>
<td>380:1 CO 2</td>
<td>NYSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)</td>
</tr>
<tr>
<td>381:1 CO 2</td>
<td>RC - STRING CONTAINING USER PROMPT MESSAGE</td>
</tr>
<tr>
<td>382:1 CO 2</td>
<td>XCOORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)</td>
</tr>
<tr>
<td>383:1 CO 2</td>
<td>XSCNM - NAME OF XVAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)</td>
</tr>
</tbody>
</table>
397 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
398 1 CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)
399 1 CD 4 YSCNND - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
400 1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
401 1 CD 4 ZTABLE - TABLE IN COMMON FOR SMTAB VARIABLE NAMES AND UNITS
402 1 CD 4 NAMVAL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
403 1 CD 4 SMTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
404 1 CD 4 PARNS - COMMUNICATION BUFFER FOR MPRM - LU, USER ID, FLAGS
405 1 CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATM
406 1 CD 4 LUDSP - BDPSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
407 1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRM
408 1 CD 4 DEBUG
409 1 CD 4 SELECT - SELECT =0 PROMPT ; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
410 1 CD 4 WITHOUT PROMPT
411 1 CD 4 CARTAG - CARTAG USED TO LOCATE DATA BOX
412 1 CD 4 CVALE -
413 1 CD 4 I nth
414 1 CD 4 ********
415 1 CD 5
416 1 CD 5
417 1 CD 5
418 1 CD 5
419 1 CD 5
420 1 CD 5
421 1 CD 5

USES ROUTINES

XPRM, XELBS, XPIXIT, RNPAR
CO************
CD0         DEFINE IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER
CD1         CO************
CD1         DEFINE ALLOCATES DATA ELEMENTS IN THE AMA THAT WERE SPECIFIED
CD1         BY THE PARAMETER KEYWORD DEFINE. IF THE DATA ELEMENT ALREADY
CD1         EXISTS, IT IS DELETED AND REALLOCATED. DATA ELEMENTS ARE
CD1         INITIALIZED TO ZERO (CHARACTER STRINGS TO BLANKS).
CD2         CO************
CD2         INPUTS FROM THE MANAGER:
CD2         LU - LOGICAL UNIT OF THE USER'S TERMINAL
CD2         DEBUG - FLAGS FOR DEBUG
CD2         CO************
CD2         INPUTS FROM THE INTERFACE TABLE:
CD2         DEFINE - SYMBOLIC STRING CONTAINING DATA ELEMENT NAME(S),
CD2         OPTIONAL I AND J DIMENSIONS AND A REQUIRED TYPE
CD3         CO************
CD3         OUTPUTS TO THE AMA:
CD3         SET OF DATA ELEMENT(S) REQUESTED
CD3         CO************
CD4         INTERNAL VARIABLES:
CD4         INBFUH - INTERFACE TABLE HEADER
CD4         ISLENG - LENGTH OF SYMBOLIC STRING
CD4         IPKTH - POSITION WITHIN THE SYMBOLIC STRING
CD4         MAHAH - BUFFER FOR VALID NAMES TO BE ALLOCATED
CD4         MBUFF - BUFFER AREA FOR XGET AND XPUT USE
CD4         NEXT - NUMBER OF ENTRIES IN THIS AMA REQUEST
CD4         NNAM - NUMBER OF NAMES IN SYMBOLIC STRING
CD4         NPOS - TOKEN POSITION FOR NEXT NAME
CD4         NSRCH - SYMBOLIC STRING INPUT TO DEFINE
CD4         CO************
CD5         EXTERNAL ROUTINES USED:
CD5         XGET, XPUT, XPUTR, XPUTL, XPUT,
CD5         XGETH, XPUTH, XPUTHL, XPUTLH
BEGIN XIDPM
SET IDIM AND JDIM TO 1
IF TOKEN IS NOT A NAME THEN
SET ERROR CODE
ELSE
MOVE NAME INTO REQUEST
INCREMENT TO NEXT TOKEN
IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREN) THEN
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT AN INTEGER OR
TOKEN IS NOT ZERO THEN
CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
EXIT TO :PNER1:
ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT AN INTEGER OR
TOKEN IS NOT ZERO THEN
CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
EXIT TO :PNER1:
ENDIF
INCREMENT TO NEXT TOKEN
ENDIF
IF TOKEN IS NOT A BEGIN TYPE FIELD SLASH THEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNER1:
ENDIF
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT A NAME THEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNER1:
ENDIF
INCREMENT TO NEXT TOKEN
STARTSEARCH FOR ALL VALID TYPES
EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
SET TYPE AND CLASS IN REQUEST
COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
IF SIZE IS TOO LARGE (>1200 WORDS) THEN
CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
EXIT TO :PNER1:
ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
EXIT TO :PNER1:
INCREMENT TO NEXT TOKEN
EXIT TO :PNER1:
EXIT TO :PNER1:
EXIT TO :PNER1:
EXIT TO :PNER1:
INCREMENT TO NEXT TOKEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNER1:
ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
EXIT TO :PNER1:
ENDIF
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
NEXT TOKEN IS NOT A COMMA THEN
CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT A NAME THEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNER1:
ENDIF
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
NEXT TOKEN IS NOT A COMMA THEN
CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
**CD**********

**ENDSC IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER**

**CD**********

**ENDSC PROCESSOR IS USED IN CONJUNCTION WITH SCAN ONLY. IT MARKS THE END OF THE SERIES OF PROCESSORS TO BE SCANNED.**

**CD**********

**IT HAS NO INTERFACE TABLE.**

**CD**********

**INPUTS FROM THE MANAGER:**

**CD2**

**LU - LOGICAL UNIT OF THE USER'S TERMINAL**

**CD2**

**USERID - USER ID CODE**

**CD2**

**FLAGS - FLAGS FOR DEBUG**

**CD2**

**CD**********

**INPUTS FROM AMA:**

**CD2**

**%SCNTB - SCAN CONTROL TABLE CREATED BY SCAN**

**CD2**

**CD**********

**OUTPUT: TO MANAGER:**

**CD3**

**XIR - RETURN CODE TO MANAGER**

**CD3**

**CD**********

**OUTPUTS TO AWA:**

**CD3**

**XSCAN - NEW VALUE FOR X SCAN VARIABLE**

**CD3**

**YSCAN - NEW VALUE FOR Y SCAN VARIABLE**

**CD3**

**CD**********

**RTE ROUTINES USED:**

**CD5**

**CLOSE, EXEC, FLOAT, KCVT, POINT, READ, RMPAR, WRIT**

**CD5**

**FDS ROUTINES USED:**

**CD5**

**XPI/T, XREXT, XRMOV, XUDBG, XVPAM, XIFCL, XIMSG**

**CD5**

**CD**********
641 1 BEGIN ENDCR
642 CALL RMMPAR TO GET INPUTS FROM MANAGER
643 SAVE REQUEST TO RETRIEVE $SCNTB FROM ANA
644 CALL XVPAM TO REQUEST ANA MANAGEMENT
645 ERREXT IF $SCNTB NOT FOUND TO :ERR4:
646 CALL EXEC TO READ IN $SCNTB
647 SAVE REQUEST TO RETRIEVE SUNTAB IN REGBUF
648 CALL XVPAM TO REQUEST ANA MANAGEMENT
649 ERREXT IF NOT FOUND TO :ERR4:
650 CALL EXEC TO READ IN SUNTAB
651 IF SUMTAB TABLE IS LARGER THAN 32 ENTRIES THEN
652 SET SIZE OF SUMTAB TO 32 ENTRIES
653 ENDF.
654 CALL WRITF TO WRITE SUMTAB TO DATBOX
655 ERREXT IF WRITF ERROR TO :ERR4:
656 IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
657 THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
658 CALL READ TO READ HEADER RECORD
659 ERREXT IF READF ERROR TO :ERR4.
660 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
661 CALL WRITF TO WRITE UPDATED HEADER
662 ERREXT IF WRITF ERROR TO :ERR4:
663 CALL CLOSE TO CLOSE DATBOX
664 SAVE REQUEST TO DELET/VER ABS $SCNTB IN REGBUF
665 IF THERE ARE REMAINING SCANS I: $SCNTB THEN
666 CALL EXEC TO WRITE REMAINING $SCNTB
667 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCNTB
668 ENDF.
669 SET RETURN PARAMETER TO NORMAL RETURN
670 ELSE
671 PERFORM SETXY
672 ENDF.
673 CALL XVPAM TO REQUEST ANA MANAGEMENT
674 CALL XPSIT TO TERMINATE WITH RETURN PARAMETERS
675 EXIT ENDSR
676 :ERR4:
677 CALL XMSG TO DISPLAY ERROR
678 CALL XPSIT TO ABEND PROCESSOR
679 1 ENDSR
1 BEGIN SETXY
2 IF XCUR IS END STEP THEN
3 IF THERE IS 1 VARIABLE THEN
4 SET X TO CENTROID
5 CALL POSNT TO POSITION FILE TO CENTROID RECORD
6 ERREXIT IF POSNT ERROR TO :ERR4:
7 ELSE
8 IF YCUR IS END STEP THEN
9 SET YCUR TO ZERO
10 SET X TO CENTROID
11 SET Y TO CENTROID
12 CALL POSNT TO POSITION TO CENTROID RECORD
13 ERREXIT IF POSNT ERROR TO :ERR4:
14 ELSE
15 SET XCUR TO (-XSTEP)
16 IF XSTEP IS ZERO AND Y IS CENTROID THEN
17 INCREMENT YCUR BY 1
18 CALL WRIF WRITE DUMMY AS CENTROID RECORD
19 ERREXIT IF WRITE ERROR TO :ERR4:
20 ENDIF
21 COMPUTE X AS (XCENT + XINCR + FLOAT(XCUR))
22 COMPUTE Y AS (YCEN + YINCR + FLOAT(YCUR))
23 ENDIF
24 ELSE
25 INCREMENT XCUR BY 1
26 IF THERE IS 1 VARIABLE AND X IS THE CENTROID OR
27 THERE ARE 2 VARIABLES AND X IS THE CENTROID AND Y IS THE CENTROID THEN
28 INCREMENT XCUR BY 1
29 CALL WRIF WRITE DUMMY AS CENTROID RECORD
30 ERREXIT IF WRITE ERROR TO :ERR4:
31 ENDIF
32 COMPUTE X AS (XCENT + XINCR + FCOUNT(XCUR))
33 IF THERE ARE 2 VARIABLES THEN
34 COMPUTE Y AS (YCEN + YINCR + FCOUNT(YCUR))
35 ENDIF
36 ENDIF
37 DJ "OR # SCAN VARIABLES
38 CALL EXEC TO WRITE VARIABLE
39 SAVE REQUEST TO STORE VALUES FOR VARIABLE IN REBUF
40 ENDDO
41 CALL EXEC TO WRITE $SCNTB
42 SAVE REQUEST TO STORE NEW $SCNTB
43 SET RETURN PARAMETERS TO_RESET SEQUENCE NUMBER
44 1 END SETXY
789  CD3  (18) XSTEP  # STEPS OF X
790  CD3  (19) XCUR  CURRENT X STEP NUMBER
791  CD3  (20) YSCAN  NAME OF Y VARIABLE
792  CD3  (21) YDSP  DISPLACEMENT FOR Y
793  CD3  (24) YCENT  CENTROID OF Y
794  CD3  (26) YINC  INCREMENT FOR Y
795  CD3  (28) YSTEP  # STEPS OF Y
796  CD3  (29) YCUR  CURRENT Y STEP NUMBER
797  CD3  (30) IDCB  144 WORD DCB FOR DATBOX
798  CD3

799  CD5

RTE SUBROUTINES USED:
CLOSE, CREAT, EXEC, FLOAT, IAND,
800  CD5  KCVT, MOD, POSNT, PURGE, RPMAR,
801  CD5  WRITF
802  CD5

FDI SUBROUTINES USED:
XPATH, XPET, XPPUT, XPYIT, XRCPR,
803  CD5  XREXT, XRMOV, XRRFM, XRUNG, XUDBG,
804  CD5
805  CD5
806  CD5
807  CD5
808  CD5
809  CD5
810  CD5
811  CD5
812  CD5


BEGIN SCAN
CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XPRINT 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 XPAT TO GET NAME AND DISPLACEMENT
COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND IDIM
CALL XPAT TO GET UNITS, CENTROID, INCQ, # STEPS
ERREXIT IT # STEPS <0 OR >5 TO :ERR3:
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR ESETB AND ESCNTB
CALL XPATM TO REQUEST AWA MANAGEMENT
SAVE REQUEST TO READ IN ESETB
IF ESCNTB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF ESCNTB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN ESETB
ERREXIT IF # SCANS > MAXIMUM ALLOWED (4) TO :ERR3;
ERREXIT IF THIS DATA BOX NAME IS ALREADY IN USE TO :ERR3;
ENDIF
ERREXIT IF THIS IS THE LAST ENTRY IN ESETB TO :ERR3;
GET THE SEQUENCE NUMBER OF THIS SCAN FROM ESETB
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH ESETB FOR THE SEQUENCE NUMBER
ERREXIT IF THIS SCAN IS THE LAST ENTRY IN ESETB 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
ENDIF
COMPUTE SIZE OF DATABOX FILE = (2+((2 + XSTEPS + 1)*(2 + YSTEPS + 1)+1))/2
COMPUTE CENTROID RECORD NUMBER = SIZE + 3
DO FOR # SCAN VARIABLES
COMPUTE BEGINNING VALUE = (CENT + INCQ + FLOAT (CUR STEP))
ENDDO
CALL XPUT 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 WRIT TO WRITE HEADER RECORD TO DATABOX
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 ESCNTB
SAVE REQUEST TO DELETE/VERIFY ABSENT ESCNTB IN REQBUFF
SAVE REQUESTS TO ALLOC AND STORE VALUES FOR NEW ESCNTB
CALL XPATM TO REQUEST AWA MANAGEMENT
073 2 ERREXIT IF NO AMA SPACE TO :ERR1:
074 2 CALL A?EXIT TO EXIT NORMALLY
075 1 EXIT SCAN
076 2 :ERR1:
077 2 IF THERE ARE MORE ACTIVE SCANS (# SCANS > 0) THEN
078 3 CALL EXEC TO READ IN NEW BSCNTB
079 3 CALL EXEC TO WRITE OUT ORIGINAL BSCNTB
080 3 SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL BSCNTB
081 2 ENDF:
082 2 :ERR2:
083 2 CALL CLOSE TO CLOSE DATBOX
084 2 CALL PURGE TO PURGE DATBOX
085 2 SET VALUE FOR XPPUT
086 2 SAVE REQUEST TO DELETE DATBOX FROM AMA
087 2 CALL XPPAM TO REQUEST AXA MANAGEMENT
088 2 :ERR3:
089 2 CALL XIMSG TO DISPLAY ERROR
090 2 CALL XP?IT TO ABEND SCAN
091 1 END SCAN
BEGIN XICHR
  SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
  CASE (RESULT DATA TYPE ) :CHISTR:, :CHRFRE:, :CHRFRE:, :FIXERR:, :FIXERR:
  :FIXERR:
  :CHISTR:
  DETERMINE # WORDS IN CHARACTER STRING
  MOVE CHARACTER STRING TO RESULT LOCATION
  IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
  BLANK FILL AFTER CHARACTER STRING
  ENDIF
  :CHRFRE:
  IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
  POP DISPLACEMENT FROM RESULT STACK
  ELSE
    SET DISPLACEMENT = 0
  ENDIF
  IF RESULT OPERAND IS FREE, THEN
  SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
  ELSE
    CHARACTER = CHARACTER
    SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
  ENDIF
  CALL XPFR TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
  IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
  BLANK FILL REMAINING DATA
  ENDIF
  END CASE
END XICHR

SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO NUMERICAL DATA"

CALL XING TO OUTPUT MESSAGE TO USER

CALL XLIST TO LIST SYMBOLIC STRING

CALL XEXIT TO EXIT PROCESSOR
END XICHR
FORTRAN CALLING SEQUENCE:

CALL XIDFT (ARRAY, START, END, TOKEN, INDEX)

XIDFT searches 'ARRAY' beginning at 'START' and ending at 'END'
for a specified 'TOKEN'. It returns either the position of
the 'TOKEN' or the 'END' value in 'INDEX'.
1083 1 BEGIN XZDFT
1084 2 SET INDEX TO START
1085 3 DO UNTIL INDEX .GE. END
1086 4 IF ARRAY(INDEX) .EQ. TOKEN, THEN
1087 4 SET END TO INDEX
1088 4 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 \\n1110 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 %
### XD01 - DATA BOX DISPLAY OVERLAY

- Prepares data for display.
- Scheduled by DDSP.

### XD01 Reads in the Scan Control Information and the Summary Tables Produced by Scan/End Scan for Further Processing.

In addition, XD01 verifies the display dependent variable names and the constraint variable names are contained in the dependent variable name list generated during the scan.

XD01 also generates constraint masks with subroutine XD0Mk for subsequent display by XD02 and XD0T.

### Inputs from the Data Box

**Record 1**

- **(1)** - Name of FDS processor creating file
- **(2)** - Interface table variable name for this file
- **(3)** - Name of FDS processor updating file
- **(4)** - 3 ASCII words of blanks
- **(5)** - Interface table variable name for this U-date
- **(6)** - 3 ASCII words of blanks

**Record 2**

- **(1)** - No of entries in summary table
- **(2)** - X scan variable (6 char)
- **(3)** - X first subscript (int or zero)
- **(4)** - X second subscript (int or zero)
- **(5)** - X units (6 char)
- **(6)** - X centroid (real)
- **(7)** - X increment (real)
- **(8)** - X number of steps (integer 1-5)
- **(9)** - Y scan variable (6 char)
- **(10)** - Y first subscript (int or zero)
- **(11)** - Y second subscript (int or zero)
- **(12)** - Y units (6 char)
- **(13)** - Y centroid (real)
- **(14)** - Y increment (real)
- **(15)** - Y number of steps (integer 1-5)

### Dependent Variable Names and Units in Records 3, 4, & 5

### Summary Table Records

- Each summary table contains value for each
DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES
INCLUDING ERROR FLAG WHICH IS FIRST VALUE
IN SUMMARY TABLE)

**OUTPUT 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
- 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

**CD 4**

- DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
- DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
- ID1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
- ID2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.
- ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XZMSK (MAX OF 8)
- IXSCLN - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
- IXSCLN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
- XFSCLN - NAME OF COMMON AREA USCD FOR INTERFACE TABLE
- IXSCLM - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
- IXSCLM2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
- MASK1 - ARRAY CONTAINING CONSTRAINTS A THRU D
- MASK2 - ARRAY CONTAINING CONSTRAINTS E THRU G
- NAMRL - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
- NC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
- NCCREL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
- NCVARL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
- NVARL - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
- NVARY - NUMBER OF DEP DISP VAR IN NVARL LIST (INTEGER)
- NVRUL - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P
- MSTRN - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=O/NOT=1
- NSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
- NSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
- XSCNM - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
- XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
- YSCNM - NAME OF Y VAR VALUES FOR X COORDINATES (1 - 11 REAL)
- YSCNM2 - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
- YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
- ITABLE - TABLE IN COMMON FOR SUMTAB VARIABLE NAMES AND UNITS
- NUNXY - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
- SUMTAB - VALUES FOR SCAN VARIABLES(S) - 1 TO 32 VALUES/RECORD
- PARMS - COMMUNICATION BUFFER FOR RMAPR - LU, USER ID, FLGS
- LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATN
- LUDSP - DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
- PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRM
- DEBUG - DEBUG
1257 1 CD 4 WITHOUT PROMPT
1258 1 CD 4 CARTGRG - CARTAGE USED TO LOCATE DATA BOX
1259 1 CD 4 CVALUE - NUMERIC (REAL) VALUES FOR EACH CONSTRAINT (UP TO 8)
1260 1 CD 4 IDC8 - DCB BLOCK FOR DBE READ
1261 1 CD 5
1262 1 CD 5 USES ROUTINES
1263 1 CD 5
1264 1 CD 5
1265 1 CD 5 XPRM, XZDMK, EXEC, OPEN, READF
1266 1 CD 5 XCMOV, XRCPR, XR18
1267 1 CD 5
1268 1 CD 5
1269 1 CD 5
1306  1 CD*****
1307  1 CD0      XZDMK - CONSTRAINT MASKING ROUTINE
1308  1 CD0
1309  1 CD0
1310  1 CD*****
1311  1 CD1
1312  1 CD1      XZDMK IS CALLED ONCE BY XZDP1 TO BUILD THE CONSTRAINT
1313  1 CD1      VIOLATION MASKS FOR ALL VALUE POSITIONS OF THE DISPLAY GRID.
1314  1 CD1
1315  1 CD*****
1316  1 CD2
1317  1 CD2      INPUTS
1318  1 CD2
1319  1 CD2      COMMON - ATABLE, NCVARL, NCPELL, CVVALUE
1320  1 CD2
1321  1 CD*****
1322  1 CD3
1323  1 CD3      OUTPUTS
1324  1 CD3
1325  1 CD3      COMMON - MSKERR, MASK1, MASK2
1326  1 CD3      ISAVE
1327  1 CD3
1328  1 CD*****
1329  1 CD4
1330  1 CD4      NOTES
1331  1 CD4
1332  1 CD4      USES Routines
1333  1 CD4
1334  1 CD4      XRCFR
1335  1 CD4      XRMOV
1336  1 CD4      XRSET
1337  1 CD4
1338  1 CD*****
1402 1 CD ******
1403 1 CD 0
1404 1 CD 0
1405 1 CD 0
1406 1 CD 0
1407 1 CD 0
1408 1 CD 0
1409 1 CD 0
1410 1 CD 0
1411 1 CD 0
1412 1 CD 0
1413 1 CD 0
1414 1 CD 0
1415 1 CD 0
1416 1 CD 0
1417 1 CD 0
1418 1 CD 0
1419 1 CD 0
1420 1 CD 0
1421 1 CD 0
1422 1 CD 0
1423 1 CD 0
1424 1 CD 0
1425 1 CD 0
1426 1 CD 0
1427 1 CD 0
1428 1 CD 0
1429 1 CD 0
1430 1 CD 0
1431 1 CD 0
1432 1 CD 0
1433 1 CD 0
1434 1 CD 0
1435 1 CD 0
1436 1 CD 0
1437 1 CD 0
1438 1 CD 0
1439 1 CD 0
1440 1 CD 0
1441 1 CD 0
1442 1 CD 0
1443 1 CD 0
1444 1 CD 0
1445 1 CD 0
1446 1 CD 0
1447 1 CD 0
1448 1 CD 0
1449 1 CD 0
1450 1 CD 0
1451 1 CD 0
1452 1 CD 0
1453 1 CD 0
1454 1 CD 0
1455 1 CD 0
1456 1 CD 0
1457 1 CD 0
1458 1 CD 0
1459 1 CD 0
1460 1 CD 0

XZDP2 - DATA BOX DISPLAY OVERLAY - PROMPTS USER FOR DISPLAY
DEDICATED, THEN FORMATS AND DISPLAYS DATA ACCORDINGLY

- SCHEDULED BY DDSP

XZDP2 ISSUES PROMPTS TO THE USER REQUESTING PAGE NUMBER,
STARTING ROW NUMBER, NUMBER OF ROWS, AND DESIRED
LJ.) SELECT OPTION CAN BE SET WHEN ENTERING
DDSP IN THE INTERFACE TABLE TO OUTPUT ALL PAGES
TO NAMED LU.
XZDP2 UTILIZES SUBROUTINE XZDOT TO ACTUALLY FORMAT
AND DISPLAY THE SCAN RESULTS.

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

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

COMBOF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XROM
DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DDSP
DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
IDV1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
IDV2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.
ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XZMSK (MAX OF 8)
ISCN1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O)
ISCN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O)
IFBUF - NAME OF COMMON AREA USED FOR INTERFACE TABLE
ITYS1 - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
ITYS2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
LENBF - LENGTH OF COMMON BUFFER IN 16-BIT WORDS - XROM
LENST - LENGTH IN DDS OF CHAR STRING USED FOR USER PROMPT - XROM
NAMES - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
NC - NUMBER OF CONSTRAINTS INPUT BY USER (INT/USER)
NREL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
NVARL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
1461 1 CD 4 NDVARL - LIST OF DEP Lisp VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
1462 1 CD 4 NDVARC - NUMBER OF DEP DISP VAR IN NDVARL LIST (INTEGER)
1463 1 CD 4 NDVARU - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P
1464 1 CD 4 NSKERR - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT 0
1465 1 CD 4 MXSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1466 1 CD 4 MXSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1467 1 CD 4 RETC - RTN CODE FM XPRDMX; 0=WRML, 1=RTN TO EXEC, 2=NULL BFR, 3=ERR
1468 1 CD 4 STRINC - 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 XSCMMN - 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 X COORDINATES (1 - 11 REAL)
1473 1 CD 4 YSCMMN - 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 SUNITAB VARIABLE NAMES AND UNITS
1476 1 CD 4 NRVARL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
1477 1 CD 4 SUNITAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
1478 1 CD 4 PARMS - COMMUNICATION BUFFER FOR RMPAR - LU, USER ID, FLAGS
1479 1 CD 4 LU - LOGICAL UNIT # FOR XPRDM CALLING SEQUENCE - USER LOCATM
1480 1 CD 4 LUDISP - OBDSIP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
1481 1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRDM
1482 1 CD 4 SELECT - SELECT =0 PROMPT; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
1483 1 CD 4 WITHOUT PROMPT
1484 1 CD 4 CARTBG - CARTAGANE USED TO LOCATE DATA BOX
1485 1 CD 4 ******** USES ROUTINES
1486 1 CD 5 XPRDM, EXEC, XZDTP
1487 1 CD 5
1488 1 CD 5
1489 1 CD 5
1490 1 CD 5
1491 1 CD 5
1492 1 CD 4 ********
XZDOT - DISPLAY OUTPUT ROUTINE

XZDOT IS CALLED ONCE BY XZDP2 TO OUTPUT THE DISPLAY PAGE

COMMON - MCVARL, MDVARL, MCERLL, EVVALUE, K1, K2, ZTABLE
ATABLE, NDVURL, MPAGE, DT6X, IXSCHM1, IYSCNH
XUNITS, YUNITS, IXSCN1, IXSCN2, IYSCN1, IYSCN2

INPUTS

OUTPUTS

NONE

NOTES

USES ROUTINES

XRPOV
EXEC
XISTP
XRPK
XRUPK
XRIG
XRE16
IABS

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1591 1 CD******
1592 1 CD0
1593 1 CD0   FORTRAN CALLING PROCEDURE
1594 1 CD0   CALL XZ1SP (STRING, LEN)
1595 1 CD0
1596 1 CD*****
1597 1 CD1
1598 1 CD1   XZ1SP REMOVES DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1599 1 CD1   A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1600 1 CD1   WITH BLANKS
1601 1 CD1
1602 1 CD*****
1603 1 CD2
1604 1 CD2   INPUT
1605 1 CD2
1606 1 CD2   CALLING SEQUENCE
1607 1 CD2
1608 1 CD2   STRING - INPUT CHARACTER STRING
1609 1 CD2   LEN - NUMBER OF WORDS IN STRING
1610 1 CD2
1611 1 CD*****
1612 1 CD3
1613 1 CD3   OUTPUT
1614 1 CD3
1615 1 CD3   CALLING SEQUENCE
1616 1 CD3
1617 1 CD3   STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
1618 1 CD3   BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILL
1619 1 CD3   LEN - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
1620 1 CD3
1621 1 CD*****
1622 1 CD4
1623 1 CD4   NOTES
1624 1 CD4
1625 1 CD4   USES ROUTINES
1626 1 CD4
1627 1 CD4   XR1SP
1628 1 CD4
1629 1 CD*****
1630 1 BEGIN XZ1SP
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 END DO
1635 1 END XZ1SP
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1437</td>
<td>C0************</td>
</tr>
<tr>
<td>1438</td>
<td>C0 FORTRAN CALLING SEQUENCE:</td>
</tr>
<tr>
<td>1439</td>
<td>C0 CALL XIFCL (LU)</td>
</tr>
<tr>
<td>1440</td>
<td></td>
</tr>
<tr>
<td>1441</td>
<td>C0************</td>
</tr>
<tr>
<td>1442</td>
<td>C0 XIFCL SEARCHES XVSTB LOOKING FOR LU SO THAT THE</td>
</tr>
<tr>
<td>1443</td>
<td>C0 PROCESSOR'S CLASS NUMBER CAN BE EXTRACTED. IT</td>
</tr>
<tr>
<td>1444</td>
<td>C0 SAVES THE CLASS NUMBER IN EXTERNAL XPCLS.</td>
</tr>
<tr>
<td>1445</td>
<td>C0************</td>
</tr>
<tr>
<td>1446</td>
<td>C0 INPUTS FROM CALLING SEQUENCE:</td>
</tr>
<tr>
<td>1447</td>
<td>C0 LU - LOGICAL UNIT OF TERMINAL BEING USED.</td>
</tr>
<tr>
<td>1448</td>
<td>C0************</td>
</tr>
<tr>
<td>1449</td>
<td>C0 NGTE: CALLING PROGRAM MUST HAVE XPCLS DEFINED</td>
</tr>
<tr>
<td>1450</td>
<td>C0 AS AN EXTERNAL reference.</td>
</tr>
<tr>
<td>1451</td>
<td>C0************</td>
</tr>
<tr>
<td>1452</td>
<td>C0 BEGIN XIFCL</td>
</tr>
<tr>
<td>1453</td>
<td>C0 CALL .ENTR TO RESOLVE PARAMETER ADDRESS</td>
</tr>
<tr>
<td>1454</td>
<td>C0 START SEARCH FOR ALL XVSTB ENTRIES</td>
</tr>
<tr>
<td>1455</td>
<td>C0 EXIT IF THIS ENTRY'S LU IS MINE</td>
</tr>
<tr>
<td>1456</td>
<td>C0 ENDSEARCH</td>
</tr>
<tr>
<td>1457</td>
<td>C0 GET CLASS NUMBER FROM XVSTB</td>
</tr>
<tr>
<td>1458</td>
<td>C0 SAVE CLASS NUMBER IN XPCLS</td>
</tr>
<tr>
<td>1459</td>
<td>C0 END XIFCL</td>
</tr>
</tbody>
</table>
1674  1 CD0 FORTRAN CALLING PROCEDURE:
1675  1 CD0
1676  1 CD0 CALL XZFCN(ENTRY)
1677  1 CD0
1678  1 CD0
1679  1 CD0-----------------------------------
1680  1 CD0
1681  1 CD0 XZFCN IS USED BY THE ASSGN ROUTINE XZPS2 TO EVALUATE FUNCTION
1682  1 CD0 OPERATIONS
1683  1 CD0
1684  1 CD0-----------------------------------
1685  1 CD0
1686  1 CD0 INPUTS
1687  1 CD0
1688  1 CD0 ENTRY - FUNCTION TOKEN CURRENTLY BEING PROCESSED
1689  1 CD0
1690  1 CD0 FROM ASGCOM - LU,STRNG,OPRND,OPINFO
1691  1 CD0
1692  1 CD0-----------------------------------
1693  1 CD0
1694  1 CD0 OUTPUTS TO ASGCOM
1695  1 CD0
1696  1 CD0 RESULT,RLTPT,OPRND
1697  1 CD0
1698  1 CD0-----------------------------------
1699  1 CD0
1700  1 CD0 EXTERNAL REFERENCES
1701  1 CD0
1702  1 CD0 FDS - XPXIT, XRMOV, XILSS, XM56
1703  1 CD0
1704  1 CD0 RTE - ABS, ABST, ALGB, ALGOT, AMOD, ATAN, ATAN2, COS, DABS, DATAN,
1705  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1706  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1707  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1708  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1709  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1710  1 CD0 DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG, DLOG,
1755 1 BEGIN XZFER
1756 2 * CASE (RESULT DATA TYPE) :RESTR:, :FREFE:, :FREFE:, :FREFIX:,
1757 3 * 2 -3 :RESTR:
1758 3 * 3 -2 :FREFE:
1759 3 * 4 -1 :FREFE:
1760 3 1 :FREFIX:
1761 3 2 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1762 3 3 MOVE CHARACTER STRING TO RESULT LOCATION
1763 3 :FREFE:
1764 3 1 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1765 3 2 POP DISPLACEMENT FROM RESULT STACK
1766 3 3 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPTED
1767 3 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 3 IF RESULT OPERAND IS FREE, THEN
1772 4 CALL XFRGR TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1773 4 SET # WORDS TO BE STORED IN OBJECT = 1
1774 3 ELSE FREE = CHARACTER DATA ELEMENT
1775 4 CALL XFRGR 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 :FREFIX:
1780 3 1 CALL XZPCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781 3 2 SET # WORDS TO BE STORED = RESULT DATA TYPE
1782 2 END CASE
1783 1 END XZFER
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1785</td>
<td>1 CD0</td>
</tr>
<tr>
<td>1786</td>
<td>FORTRAN CALLING PROCEDURE:</td>
</tr>
<tr>
<td>1787</td>
<td>CALL XIFXD</td>
</tr>
<tr>
<td>1788</td>
<td>XIFXD IS USED BY THE ASSGN ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1789</td>
<td>FOR FIXED-TYPE OBJECT DATA ELEMENTS</td>
</tr>
<tr>
<td>1790</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1791</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1792</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1793</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1794</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1795</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1796</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1797</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1798</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1799</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1800</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1801</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1802</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1803</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1804</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1805</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1806</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1807</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1808</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1809</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1810</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1811</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1812</td>
<td>XIFXD</td>
</tr>
<tr>
<td>1813</td>
<td>XIFXD</td>
</tr>
</tbody>
</table>
BEGIN XIFXD

1. Set # words to be stored in object = object data type

2. Case (result data type) :
   -2. NoChar
   -1. FixReal
   0. FixFix
   3. FixFix

3. FixReal: Error if type in symbol table for result operand is not free
4. If pop entry on result stack is a displacement (type = -2), then
5. Pop displacement from result stack
6. Else free element has not been subscripted
7. Set displacement = 0
8. Endif
9. Pop result operand from result stack (symbol table index)
10. Call xpreq to retrieve data from result operand at displacement determined
11. (If words retrieved = object data type)
12. FixFix: Set target type to object data type
13. Call xpcs to pop result value, convert if necessary, and set up for store
14. Endcase
15. Exit xifxd

16. NoChar:
17. Set message to be output to "Numerical data element cannot be set equal to
18. Character data"
19. Call ximg to output message to user
20. Call xlist to list symbolic string
21. Call xprint to exit processor
22. End xifxd
INTERNAL VARIABLES

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

ENTRY 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER
ENTRY 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER
ENTRY 3 (TOKSZ) = NO. OF WORDS IN SYMBOLIC STRINGS

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD4</td>
<td>CD4</td>
<td>CD4</td>
</tr>
<tr>
<td>1871</td>
<td>1872</td>
<td>1873</td>
</tr>
<tr>
<td>1874</td>
<td>1875</td>
<td>1876</td>
</tr>
<tr>
<td>1877</td>
<td>1878</td>
<td>1879</td>
</tr>
<tr>
<td>1880</td>
<td>1881</td>
<td>1882</td>
</tr>
<tr>
<td>1883</td>
<td>1884</td>
<td>1885</td>
</tr>
<tr>
<td>1886</td>
<td>1887</td>
<td>1888</td>
</tr>
<tr>
<td>1889</td>
<td>1890</td>
<td>1891</td>
</tr>
<tr>
<td>1892</td>
<td>1893</td>
<td>1894</td>
</tr>
<tr>
<td>1895</td>
<td>1896</td>
<td>1897</td>
</tr>
<tr>
<td>1898</td>
<td>1899</td>
<td>1900</td>
</tr>
<tr>
<td>1901</td>
<td>1902</td>
<td>1903</td>
</tr>
<tr>
<td>1904</td>
<td>1905</td>
<td>1906</td>
</tr>
<tr>
<td>1907</td>
<td>1908</td>
<td>1909</td>
</tr>
<tr>
<td>1910</td>
<td>1911</td>
<td>1912</td>
</tr>
<tr>
<td>1913</td>
<td>1914</td>
<td>1915</td>
</tr>
<tr>
<td>1916</td>
<td>1917</td>
<td>1918</td>
</tr>
<tr>
<td>1919</td>
<td>1920</td>
<td>1921</td>
</tr>
<tr>
<td>1922</td>
<td>1923</td>
<td>1924</td>
</tr>
</tbody>
</table>
1 ***
2 CO5
3 CO5
4 NOTES
5 CO5
6 USES ROUTINES
7 1937
8 1938
9 BEGIN XILLS
10 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
11 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
12 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
13 USE TOKEN VALUE TO RETRIEVE 3 control words (SIZF, FIELD, TOKSZ)
14 EXIT TO ERROR 2 IF FIELD = O
15 IF SIZE < O, THEN
16 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYM. STRING
17 ENDIF
18 IF TOKSZ < O, THEN
19 SET TOKSZ TO SIZE + 2
20 ENDIF
21 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
22 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
23 SET BUFFER POINTER TO 1ST POSITION FOR DATA
24 IF INDICATED TOKEN PROCESSED, THEN
25 OUTPUT LINE WITH INDICATOR
26 ENDIF
27 IF FIELD > O, THEN
28 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
29 ELSE
30 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
31 SOME: CALL XR16 WITH VALUE IN NEXT WORD OF SYM. STRING
32 AND PUT RESULTS INTO PRINT BUFFER
33 TWO: CALL XR14 WITH VALUE IN NEXT 2 WORDS OF SYM. STRING
34 AND PUT RESULTS INTO PRINT BUFFER
35 THREE: CALL XR18 WITH VALUE IN NEXT 3 WORDS OF SYM. STRING
36 AND PUT RESULTS INTO PRINT BUFFER
37 FOUR: MOVE THE NEXT 3 WORDS OF SYM. STRING INTO PRINT BUFFER
38 FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYM. STRING
39 SIX: CALL XR16 WITH VALUE IN NEXT WORD OF SYM. STRING AND PUT RESULTS
40 INTO PRINT BUFFER FOLLOWED BY AN "R"
41 EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
42 INDEX BY 1
43 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
44 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
45 INDICATOR LINE NOT YET OUTPUT, THEN
46
FORTRAN CALLING PROCEDURE

CALL XMSG (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

NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'AHH' WHERE
A - AREA INDICATOR AS FollowS

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'
WORDS OF 'SOURCE' TO BE OUTPUT

LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED

'SOURCE' (NOT USED IF 'NN' 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 ZERO)

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

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

* NOTES
USES FOS SYSTEM MESSAGE FILE XRMSS
USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XRMV, XUDBG
BEGIN XZMSG
  SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
  SET MESSAGE 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 XRM0V TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
            ELSE
              EXIT TO :ERROR:
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
  CALL XRM0V TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
  IF MESSAGE NUMBER > 0
    THEN
      CALL XRM0V TO MOVE 2 REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
  ENDIF
  ELSE
    SET AREA IN PREFIX
    ERROR: CALL XRM0V TO MOVE 'XZMSG ERROR' INTO BUFFER
  ENDIF
  CALL XRM0V TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
  ENDIF
  OUTPUT BUFFER TO USER'S TERMINAL
  IF DEBUG IS REQUESTED
    THEN
      CALL XUDEBUG
  ENDIF
END XZMSG
<table>
<thead>
<tr>
<th>Line</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CD0</td>
</tr>
<tr>
<td>2</td>
<td>CD0</td>
</tr>
<tr>
<td>3</td>
<td>CD0</td>
</tr>
<tr>
<td>4</td>
<td>CD0</td>
</tr>
<tr>
<td>5</td>
<td>CD0</td>
</tr>
<tr>
<td>6</td>
<td>CD0</td>
</tr>
<tr>
<td>7</td>
<td>CD0</td>
</tr>
<tr>
<td>8</td>
<td>CD0</td>
</tr>
<tr>
<td>9</td>
<td>CALL XIOPR(ENTRY)</td>
</tr>
<tr>
<td>10</td>
<td>CD0</td>
</tr>
<tr>
<td>11</td>
<td>CD0</td>
</tr>
<tr>
<td>12</td>
<td>CD0</td>
</tr>
<tr>
<td>13</td>
<td>CD0</td>
</tr>
<tr>
<td>14</td>
<td>CD0</td>
</tr>
<tr>
<td>15</td>
<td>CD0</td>
</tr>
<tr>
<td>16</td>
<td>CD0</td>
</tr>
<tr>
<td>17</td>
<td>CD0</td>
</tr>
<tr>
<td>18</td>
<td>CD0</td>
</tr>
<tr>
<td>19</td>
<td>CD0</td>
</tr>
<tr>
<td>20</td>
<td>CD0</td>
</tr>
<tr>
<td>21</td>
<td>CD0</td>
</tr>
<tr>
<td>22</td>
<td>CD0</td>
</tr>
<tr>
<td>23</td>
<td>CD0</td>
</tr>
<tr>
<td>24</td>
<td>CD0</td>
</tr>
<tr>
<td>25</td>
<td>CD0</td>
</tr>
<tr>
<td>26</td>
<td>CD0</td>
</tr>
<tr>
<td>27</td>
<td>CD0</td>
</tr>
<tr>
<td>28</td>
<td>CD0</td>
</tr>
<tr>
<td>29</td>
<td>CD0</td>
</tr>
<tr>
<td>30</td>
<td>CD0</td>
</tr>
<tr>
<td>31</td>
<td>CD0</td>
</tr>
<tr>
<td>32</td>
<td>CD0</td>
</tr>
<tr>
<td>33</td>
<td>CD0</td>
</tr>
<tr>
<td>34</td>
<td>CD0</td>
</tr>
<tr>
<td>35</td>
<td>CD0</td>
</tr>
<tr>
<td>36</td>
<td>CD0</td>
</tr>
<tr>
<td>37</td>
<td>CD0</td>
</tr>
<tr>
<td>38</td>
<td>CD0</td>
</tr>
<tr>
<td>39</td>
<td>CD0</td>
</tr>
<tr>
<td>40</td>
<td>CD0</td>
</tr>
<tr>
<td>41</td>
<td>CD0</td>
</tr>
<tr>
<td>42</td>
<td>CD0</td>
</tr>
<tr>
<td>43</td>
<td>CD0</td>
</tr>
<tr>
<td>44</td>
<td>CD0</td>
</tr>
<tr>
<td>45</td>
<td>CD0</td>
</tr>
<tr>
<td>46</td>
<td>CD0</td>
</tr>
<tr>
<td>47</td>
<td>CD0</td>
</tr>
<tr>
<td>48</td>
<td>CD0</td>
</tr>
<tr>
<td>49</td>
<td>CD0</td>
</tr>
<tr>
<td>50</td>
<td>CD0</td>
</tr>
<tr>
<td>51</td>
<td>CD0</td>
</tr>
<tr>
<td>52</td>
<td>CD0</td>
</tr>
<tr>
<td>53</td>
<td>CD0</td>
</tr>
<tr>
<td>54</td>
<td>CD0</td>
</tr>
<tr>
<td>55</td>
<td>CD0</td>
</tr>
<tr>
<td>56</td>
<td>CD0</td>
</tr>
<tr>
<td>57</td>
<td>CD0</td>
</tr>
<tr>
<td>58</td>
<td>CD0</td>
</tr>
<tr>
<td>59</td>
<td>CD0</td>
</tr>
<tr>
<td>60</td>
<td>CD0</td>
</tr>
<tr>
<td>61</td>
<td>CD0</td>
</tr>
<tr>
<td>62</td>
<td>CD0</td>
</tr>
<tr>
<td>63</td>
<td>CD0</td>
</tr>
<tr>
<td>64</td>
<td>CD0</td>
</tr>
<tr>
<td>65</td>
<td>CD0</td>
</tr>
<tr>
<td>66</td>
<td>CD0</td>
</tr>
<tr>
<td>67</td>
<td>CD0</td>
</tr>
<tr>
<td>68</td>
<td>CD0</td>
</tr>
<tr>
<td>69</td>
<td>CD0</td>
</tr>
<tr>
<td>70</td>
<td>CD0</td>
</tr>
<tr>
<td>71</td>
<td>CD0</td>
</tr>
<tr>
<td>72</td>
<td>CD0</td>
</tr>
<tr>
<td>73</td>
<td>CD0</td>
</tr>
<tr>
<td>74</td>
<td>CD0</td>
</tr>
<tr>
<td>75</td>
<td>CD0</td>
</tr>
<tr>
<td>76</td>
<td>CD0</td>
</tr>
<tr>
<td>77</td>
<td>CD0</td>
</tr>
<tr>
<td>78</td>
<td>CD0</td>
</tr>
<tr>
<td>79</td>
<td>CD0</td>
</tr>
<tr>
<td>80</td>
<td>CD0</td>
</tr>
<tr>
<td>81</td>
<td>CD0</td>
</tr>
<tr>
<td>82</td>
<td>CD0</td>
</tr>
<tr>
<td>83</td>
<td>CD0</td>
</tr>
<tr>
<td>84</td>
<td>CD0</td>
</tr>
<tr>
<td>85</td>
<td>CD0</td>
</tr>
<tr>
<td>86</td>
<td>CD0</td>
</tr>
<tr>
<td>87</td>
<td>CD0</td>
</tr>
<tr>
<td>88</td>
<td>CD0</td>
</tr>
<tr>
<td>89</td>
<td>CD0</td>
</tr>
<tr>
<td>90</td>
<td>CD0</td>
</tr>
<tr>
<td>91</td>
<td>CD0</td>
</tr>
<tr>
<td>92</td>
<td>CD0</td>
</tr>
<tr>
<td>93</td>
<td>CD0</td>
</tr>
<tr>
<td>94</td>
<td>CD0</td>
</tr>
<tr>
<td>95</td>
<td>CD0</td>
</tr>
<tr>
<td>96</td>
<td>CD0</td>
</tr>
<tr>
<td>97</td>
<td>CD0</td>
</tr>
<tr>
<td>98</td>
<td>CD0</td>
</tr>
<tr>
<td>99</td>
<td>CD0</td>
</tr>
<tr>
<td>100</td>
<td>CD0</td>
</tr>
<tr>
<td>101</td>
<td>CD0</td>
</tr>
<tr>
<td>102</td>
<td>CD0</td>
</tr>
<tr>
<td>103</td>
<td>CD0</td>
</tr>
<tr>
<td>104</td>
<td>CD0</td>
</tr>
<tr>
<td>105</td>
<td>CD0</td>
</tr>
<tr>
<td>106</td>
<td>CD0</td>
</tr>
<tr>
<td>107</td>
<td>CD0</td>
</tr>
</tbody>
</table>

**FORTRAN CALLING PROCEDURE:**

**Inputs:**
- ENTRY - OPERATOR TOKEN CURRENTLY BEING PROCESSED
- FROM ASGCOM - LU, SSTRING, SYNTAX, DATYPV, RSLTP, CLSTM, MAPHS, RESULT

**Outputs:**
- TO ASGCOM
- RESULT, OPANDS, REGST, RSLTP

**Internal Variables:**

**External References:**
- XPREM, XKXIT, XRMV, XILS, XKMSG
- RTE - IANK, OUF
1: BEGIN XOPR
3: EXIT XOPR

4: :ADD:
5: PERFORM ADDITION
6: SET RESULT'S DATA TYPE AS APPROPRIATE

7: :SUBTR:
8: PERFORM SUBTRACTION
9: SET RESULT'S DATA TYPE AS APPROPRIATE

10: :MULT:
11: PERFORM MULTIPLICATION
12: SET RESULT'S DATA TYPE AS APPROPRIATE

13: :DIVIDE:
14: PERFORM DIVISION
15: SET RESULT'S DATA TYPE AS APPROPRIATE

16: :INDEX:
17: IF RESULT STACK IS NOT EMPTY, THEN
18: GET 1-DIM FOR TOP OPERAND IN RESULT STACK
19: SET # WORDS PER ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK
20: ELSE EIGELE SUBSCRIPT FOR OBJECT IS BEING EVALUATED
21: GET 1-DIM FOR FIRST ENTRY IN SYMBOL TABLE
22: SET # WORDS PER ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
23: ENDF
24: CALCULATE RESULT (I.E., INDEX = IDIM*(SECOND OPERAND-1)+FIRST OPERAND)

25: :SUBSCR:
26: IF RESULT STACK IS NOT EMPTY, THEN
27: SET # WORDS/ELEMENT BASED ON TOP ENTRY OF RESULT STACK ENTRY
28: ELSE OBJECT IS BEING SUBSCRIPTED
29: SET # WORDS/ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
30: ENDF
31: CALCULATE RESULT (I.E., DISPLACEMENT = (OPERAND-1)*#WORDS PER ENTRY)
32: IF RESULT STACK IS NOT EMPTY AND TOP ENTRY IS NOT FREE OR CHARACTER, THEN
33: CALL FREE TO RETRIEVE A LOGICAL VALUE FOR TOP OPERAND AT DISPLACEMENT
34: DECREMENT RESULT STACK POINTER TO DISCARD ARRAY NAME
35: ELSE OBJECT IS BEING SUBSCRIPTED
36: SET TYPE = -2
37: ENDF

38: :EXP:
39: PERFORM EXPONENTIATION
40: SET RESULT'S DATA TYPE AS APPROPRIATE

41: :UNIM:
42: CHANGE SIGN OF OPERAND FOR RESULT

43: ENDCASE
44: EXIT IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
45: PUSH RESULT AND TYPE ONTO RESULT STACK
46: EXIT XOPR

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

CALL XIPS1

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

INPUTS FROM ASGCOM

LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTBL

OUTPUTS TO ASGCOM

EXPTR, RNGSTK, POLISH, SYNTAX

INTERNAL VARIABLES

GRPSK - 2X120 ARRAY USED TO TRACK FUNCTIONS, SUBSCRIPTS, AND PARENTHESELICAL GROUPINGS

OPSTK - 2X120 ARRAY; EACH ENTRY CONTAINS OPERATOR TOKEN AND ITS OUTPUT PRIORITY

PRCONT - PRECEDENT TYPE FOR PRECEDING CHARACTER; USED FOR SYNTAX CHECK

TKPTR - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING

TOKEN - TOKEN CURRENTLY BEING PROCESSED

EXTERNAL REFERENCES

RTE - IAMD, MINO

FDS - XPXII, XILSS, XIMSG, XSYM
<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2273</td>
<td>BEGIN SETUP</td>
</tr>
<tr>
<td>2274</td>
<td>CASE TOKEN</td>
</tr>
<tr>
<td>2275</td>
<td>ESS: INTEGRAL REAL DOUBLE</td>
</tr>
<tr>
<td>2276</td>
<td>SYMBOL: SYMBOL: SYMBOL:</td>
</tr>
<tr>
<td>2277</td>
<td>NAME CHARACTER + -</td>
</tr>
<tr>
<td>2278</td>
<td>SYMBOL: CHAR: PLUS: MINUS: ASTERISK: &lt; &gt;</td>
</tr>
<tr>
<td>2279</td>
<td>ENDCASE: INVLDP: INVLDP: INVLDP: INVLDP: INVLDP:</td>
</tr>
<tr>
<td>2280</td>
<td>IF TOKEN IS A NAME AND NEXT TOKEN IS A LEFT BRACKET, THEN</td>
</tr>
<tr>
<td>2281</td>
<td>START SEARCH UNTIL FUNCTION TABLE IS CHECKED</td>
</tr>
<tr>
<td>2282</td>
<td>EXIT IF TOKEN NAME MATCHES FUNCTION NAME</td>
</tr>
<tr>
<td>2283</td>
<td>CHANGE TOKEN TO FUNCTION'S INDEX IN TABLE + 128</td>
</tr>
<tr>
<td>2284</td>
<td>PUSH COUNT=BINARY/UNARY FLAG AND LIMIT=0 ONTO GROUPING STACK</td>
</tr>
<tr>
<td>2285</td>
<td>ENDF</td>
</tr>
</tbody>
</table>
2325  2  ENDCAS:
2326  1  EXIT SETUP
2327  2  :INVLD:
2328  2  SET MESSAGE TO BE OUTPUT TO "INVALID CHARACTER"
2329  2  :BADFUN:
2330  2  SET MESSAGE TO BE OUTPUT TO "FUNCTION NOT SUPPORTED BY THIS PROCESSOR"
2331  2  CALL XIMSG TO OUTPUT MESSAGE TO USER
2332  2  CALL XILSS TO LIST SYMBOLIC STRING
2333  2  CALL XPIXT TO EXIT PROCESSOR
2334  1  END SETUP
BEGIN STRING

SET INDEX INTO SYNTAX TABLE TO MIN(TOKEN,40)

ERREXIT TO :SYNTX1: IF THIS TOKEN IS NOT VALID ACCORDING TO SYNTAX TABLE

IF TOKEN IS NOT AN OPERAND, THEN

DO UNTIL TOKEN IS PUSHD ONTO OPERATOR STACK OR DISCARDED

IF INPUT PRIORITY OF THIS TOKEN > OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR STACK, THEN

INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK

ERREXIT TO :SYNTX1: IF COUNT > COMMA LIMIT FOR (TOP ENTRY IN GRPPING STK

IF COMMA LIMIT SHOOWS SUBSCRIPTING IN F..,G...ESS (LIMIT > 0), THEN

PUSH TOKEN AND OUTPUT PRIORITY ONTO OPERATOR STACK

ELSE

DISCARD >> FUNCTION LIST IS BEING PROCESSED

ENDIF

ELSE OPERATOR IS NOT :

PUSH TOKEN AND ITS OUTPUT PRIORITY ONTO OPERATOR STACK

ENDIF

ELSE INPUT PRIORITY IS < OR = OUTPUT PRIORITY

IF INPUT PRIORITY < OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR STACK OR

INPUT PRIORITY = OUTPUT PRIORITY NOT = 2, THEN

ERREXIT TO :SYNTX1: IF INPUT PRIORITY = 0 AND OPERATOR STACK IS EMPTY

IF TOP ENTRY OF OPERATOR STACK IS #, THEN

ERREXIT IF THIS IS NOT LAST ENTRY ON OPERATOR STACK TO :BADEL:

ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTX2:

DISCARD TOKEN (; OR ESS)

ENDIF

POP OPERATOR STACK

POP OPERATOR ONTO EXPRESSION STACK

ELSE BRACKETS OR PARENTHESES HAVE BEEN MATCHED

IF TOP ENTRY OF OPERATOR STACK IS ( THEN

ERREXIT TO :SYNTX2:

IF TOP ENTRY OF GROUPING STACK INDICATES SUBSCRIPTING (LIMIT > 0), THEN

PUSH SUBSCRIPTING OPERATOR ONTO EXPRESSION ARRAY

ENDIF

ELSE TOP OPERATOR ENTRY IS OPEN Bracket (FUNCTION LIST)

ERREXIT IF CURRENT TOKEN IS NOT CLOSE Bracket TO :SYNTX2:

ERREXIT TO :FLSTK: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GRPPING STK COUNT=0)

ENDIF

POP OPERATOR STACK

POP GROUPING STACK

DISCARD CURRENT TOKEN

ENDIF

END DEFINITION

INCREMENT TO NEXT TOKEN USING TOKEN LENGTH FROM SYNTAX TABLE

EXIT STRIN

:SYNTX1:

SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"

:SYNTX2:

SET MESSAGE TO BE OUTPUT TO "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY"

:COMERR:

SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
BEGIN RANGE
INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
IF LAST TOKEN INPUT HAS 7, THEN
DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
CALL XSYM TO INSERT NAME IN SYMBOL TABLE, IF NECESSARY, FLAG ENTRY AS
AN INDEX, AND RETURN SYMBOL TABLE INDEX
ERREXIT IF NEXT TOKENS ARE NOT "=INTEGER,INTEGER" TO :RNGSYM:
IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
SET INCREMENT TO -1
ENDIF
PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
END DO
ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
ENDIF
EXIT RANGE
:RNGSYM:
SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ";NAME=
* INTEGER,INTEGER"
CALL XMSG TO OUTPUT MESSAGE TO USER
CALL XLSS TO LIST SYMBOLIC STRING
CALL XPRT TO EXIT PROCESSOR
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, SSTRING, RESULT, OPRNDs

OPINFO, NUMWDs, DATIPS, RSLTPy, EXPRx, CLSRex, CLSRM, DECxAS,

RNGxSTx, POLISH, SYNTAX, FNCyBL

OUTPUTS TO ASGCOM

SYMTAB, RESULT, RERxST, RSLTPY, OPINFO

INTERNAL VARIABLES

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

EXTERNAL REFERENCES

FDS - XPRyER, XPIIT, XRMxLV, XICXHxR, XIFXMC, XIFxRE, XIFxFOD, XILSS, XIMxSG,

XIOPRx, XPRSxS, XIRET

RTE - IABs, IANO, MAXB
1 BEGIN XIPS2
2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
3 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
4 INITIALIZE RANGE VALUES FOR ITERATION
5 DO UNTIL ALL RANGES ARE FINISHED
6 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
7 POP ENTRY FROM POLISH STRING
8 IF ENTRY IS AN OPERAND, THEN
9 PERFORM RPUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH INTO RESULT STACK
10 ELSE ENTRY IS AN OPERATOR
11 IF OPERATOR IS NOT =, THEN
12 PERFORM EXEVAL TO EVALUATE POLISH STRING
13 ELSE OPERATOR IS =
14 PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
15 ENDEF
16 ENDF
17 END DO
18 END DO
19 END XIPS2

REPEREABILITY OF THE
ORIGINAL PAGE IS POOR
2486 1 BEGIN TOC
2487 2 "DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED"
2488 3 "IF SYMBOL NOT = O (SYMBOL IS DE OR INDEX), THEN"
2489 4 "IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN"
2490 5 "CALL XPREQ TO RETRIEVE TOC ENTRY"
2491 6 "ELSE SYMBOL IS A RANGE INDEX"
2492 7 "SET DATA TYPE TO INTEGER"
2493 8 "ENDIF"
2494 9 "ENDIF"
2495 10 "END DO"
2496 11 "CALL XPREQ WITH A CLOSE BUFFER REQUEST"
2497 12 "END TOC"
2499 1 BEGIN DATA1
2500 2 DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
2501 3 IF SYMBOL IS A NON-SUBSCRIBED FIXED-TYPE DATA ELEMENT, THEN
2502 4 CALL XPREQ TO QUEUE REQUEST FOR DATA RETRIEVAL
2503 5 ENDIF
2504 6 END DO
2505 7 CALL XPREQ WITH A CLOSE BUFFER REQUEST
2506 8 END DATA1
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 IF 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 ENDIF
2515 1 END RPUSH
1 BEGIN EVALUATE
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 RECOVER DATA AND TYPE FOR #1 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 XPCE TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
20 END DO
21 CALL XZPMC TO PERFORM FUNCTION FOR RESULTS
22 SET DATA TYPE FOR RESULTS FROM FUNCTION TABLE
23 ELSE IF THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "=
24 DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
25 CASE (TYPE REQUIREMENT+1) :SAME:, :INTERC:
26 :SAME:
27 IF # OPERANDS > 1, THEN
28 SET TARGET TYPE TO MAX OF TWO DATA TYPES
29 ELSE # OPERANDS = 1
30 SET TARGET TYPE TO OPERAND'S DATA TYPE
31 ENDIF
32 :INTEGER:
33 SET TARGET TYPE TO INTEGER
34 ENDCASE
35 DO FOR EACH OPERAND
36 CALL XPCE TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
37 END DO
38 CALL XZOPR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
39 ENDIF
40 EXIT EVALUATE
41
42 :MIXERR:
43 SET MESSAGE TO BE OUTPUT TO "FREE OR CHARACTER DATA ELEMENT FOUND IN AN
44 EXPRESSION"
45 CALL XPMSG TO OUTPUT MESSAGE TO USER
46 CALL XLSS TO LIST SYMBOLIC STRING
47 CALL XPEXIT TO EXIT PROCESSOR
48 1 END EVALUATE
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 XPREG TO RETRIEVE FIRST ELEMENT OF THE ARRAY
2569 5 STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
2570 3 EMDIF
2571 2 * EMDIF
2572 2 * CASE (OBJECT DATA TYPE ) :FREE,:FIXED,:FIXED,:FIXED,:CHAR,:CHAR,
2573 3 * :CHAR,:CHAR,:CHAR:
2574 3 * :FREE:
2575 3 * CALL XPFRF TO RETRIEVE DATA AND SET UP FOR STORE
2576 3 :FIXED:
2577 3 CALL XPFRF TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
2578 3 :CHAR:
2579 3 CALL XCHROM 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 EMDIF
2586 2 ENDIF
2587 2 BUILD XPFRF REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S
2588 2 DISPLACEMENT
2589 2 CALL XPREG TO STORE DATA IN OBJECT
2590 1 END REPLAC
2593 1 BEGIN RNGSET
2594 2 DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
2595 3 * RANGES ARE PROCESSED
2596 4 IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
2597 5 INCREMENT RANGE VALUE
2598 6 ELSE
2599 7 SET RANGE INDEX VALUE TO BEGIN VALUE
2600 8 ENDF
2601 9 END DO
2602 1 END RNGSET
FORTRAN CALLING PROCEDURE:
CALL XIRET(ENTRY)

XIRET is used by the assign routine XIPS2 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 ASGCOM - SYNTAB,RSLTPT

OUTPUTS TO ASGCOM
RESULT,RSLTPT

EXTERNAL REFERENCES
FDS - XRMOV
RTE - IAND
REPRODUCIBILITY OF
ORIGINAL PAGE IS POOR.
FORTRAN CALLING PROCEDURE

CALL XSYM (TKMPTR, FLAG, SYMIND)

XSYM PROVIDES SYMBOL TABLE BUILDING AND ACCESS FOR ASSGN. IT
ACCOMPLISHES RANGE AND SUBSCRIPT PROCESSING DIRECTLY

INPUT
TKMPTR - INDEX INTO INPUT SYMBOLIC STRING (SSTRNG) OF TOKEN TO BE
ENTERED OR LOCATED IN SYMBOL TABLE (SYMTAB)

FLAG - SPECIAL PROCESSING FLAG
0, NORMAL PROCESSING
1, SYMBOL IS A RANGE
2, SYMBOL IS SUBSCRIPTED

ASGCOM COMMON - LU, SSTRNG, STWIDE, SYMTAB, TOKENS,

OUTPUT

SYMIND - INDEX INTO SYMBOL TABLE (SYMTAB) WHERE SYMBOL IS LOCATED.
VALUE OF SYMIND INDICATES ENTRY NUMBER, I.E., 1, 2, ...

ASGCOM COMMON - SYMTAB

INTERNAL

SYMTPR - INDEX TO FLAG WORD OF ENTRY IN SYMTAB. VALUE IS EQUAL TO
SYMIND + STWIDE

EXTERNAL REFERENCES
XPIIT, XILSS, XIMSG, XIST
2647 BSTM XSTM
2648 IF T0KEND TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY
2649 2691 IF T0KEND IS A MAN8, THEN FLAG IS SET (1) (RANGE INDEX, 2-SUBSCRIPTED), THEN
2650 IF FLAG INDICATES RANGE INDEX, THEN EN6T TO SEARCH SUBCLAS TABLED
2651 IF ERROR ALREADY EXISTS, THEN EN6T TO SEARCH SUBCLAS TABLED
2652 EN6T TO SEARCH SUBCLAS TABLED
2653 EN6T TO SEARCH SUBCLAS TABLED
2654 EN6T TO SEARCH SUBCLAS TABLED
2655 EN6T TO SEARCH SUBCLAS TABLED
2656 EN6T TO SEARCH SUBCLAS TABLED
2657 IF ENTRY FLAG WORD TO FLAG VALUE
2658 AND D1AS OF 256 TO SYMBOL INDEX
2659 2700 1 XSTM
2701 2 XSTM
2702 3 XSTM
2703 4 XSTM
2704 5 XSTM
2705 6 XSTM
2706 7 XSTM
2707 8 XSTM
2708 9 XSTM
2709 10 XSTM
2710 11 XSTM
2711 12 XSTM
2712 13 XSTM
2713 14 XSTM
2714 15 XSTM
2715 16 XSTM
2716 17 XSTM
2717 18 XSTM
2718 19 XSTM
2719 20 XSTM
2720 21 XSTM
2721 22 XSTM
2722 23 XSTM
2723 24 XSTM
2724 25 XSTM
2725 26 XSTM
2726 27 XSTM
2727 28 XSTM
2728 29 XSTM
2729 30 XSTM
2730 31 XSTM
2731 32 XSTM
2732 33 XSTM
2733 34 XSTM
2734 35 XSTM
2735 36 XSTM
2736 37 XSTM
2737 38 XSTM
2738 39 XSTM
2739 40 XSTM
2740 41 XSTM
2741 42 XSTM
2742 43 XSTM
2743 44 XSTM
2744 45 XSTM
2745 46 XSTM
2746 47 XSTM
2747 48 XSTM
2748 49 XSTM
2749 50 XSTM
2750 51 XSTM
2751 52 XSTM
2752 53 XSTM
2753 54 XSTM
2754 55 XSTM
2755 56 XSTM
2756 57 XSTM
2757 58 XSTM
2758 59 XSTM
2759 60 XSTM
2760 61 XSTM
2761 62 XSTM
2762 63 XSTM
2763 64 XSTM
2764 65 XSTM
2765 66 XSTM
2766 67 XSTM
2767 68 XSTM
2768 69 XSTM
2769 70 XSTM
2770 71 XSTM
2771 72 XSTM
2772 73 XSTM
2773 74 XSTM
2774 75 XSTM
2775 76 XSTM
2776 77 XSTM
2777 78 XSTM
2778 79 XSTM
2779 80 XSTM
2780 81 XSTM
2781 82 XSTM
2782 83 XSTM
2783 84 XSTM
2784 85 XSTM
2785 86 XSTM
2786 87 XSTM
2787 88 XSTM
2788 89 XSTM
2789 90 XSTM
2790 91 XSTM
2791 92 XSTM
2792 93 XSTM
2793 94 XSTM
2794 95 XSTM
2795 96 XSTM
2796 97 XSTM
2797 98 XSTM
2798 99 XSTM
2799 100 XSTM
2800 101 XSTM
2801 102 XSTM
2802 103 XSTM
2803 104 XSTM
2804 105 XSTM
2805 106 XSTM
2806 107 XSTM
2807 108 XSTM
2808 109 XSTM
2809 110 XSTM
2810 111 XSTM
2811 112 XSTM
2812 113 XSTM
2813 114 XSTM
2814 115 XSTM
2815 116 XSTM
2816 117 XSTM
2817 118 XSTM
2818 119 XSTM
2819 120 XSTM
2820 121 XSTM
2821 122 XSTM
2822 123 XSTM
2823 124 XSTM
2824 125 XSTM
2825 126 XSTM
2826 127 XSTM
2827 128 XSTM
2828 129 XSTM
2829 130 XSTM
2830 131 XSTM
2831 132 XSTM
2832 133 XSTM
2833 134 XSTM
2834 135 XSTM
2835 136 XSTM
2836 137 XSTM
2837 138 XSTM
2838 139 XSTM
2839 140 XSTM
2840 141 XSTM
2841 142 XSTM
2842 143 XSTM
2843 144 XSTM
2844 145 XSTM
2845 146 XSTM
2846 147 XSTM
2847 148 XSTM
2848 149 XSTM
2849 150 XSTM
2850 151 XSTM
2851 152 XSTM
2852 153 XSTM
2853 154 XSTM
2854 155 XSTM
2855 156 XSTM
2856 157 XSTM
2857 158 XSTM
2858 159 XSTM
2859 160 XSTM
2860 161 XSTM
2861 162 XSTM
2862 163 XSTM
2863 164 XSTM
2864 165 XSTM
2865 166 XSTM
2866 167 XSTM
2867 168 XSTM
2868 169 XSTM
2869 170 XSTM
2870 171 XSTM
2871 172 XSTM
2872 173 XSTM
2873 174 XSTM
2874 175 XSTM
2875 176 XSTM
2876 177 XSTM
2877 178 XSTM
2878 179 XSTM
2879 180 XSTM
2880 181 XSTM
2881 182 XSTM
2882 183 XSTM
2883 184 XSTM
2884 185 XSTM
2885 186 XSTM
2886 187 XSTM
2887 188 XSTM
2888 189 XSTM
2889 190 XSTM
2890 191 XSTM
2891 192 XSTM
2892 193 XSTM
2893 194 XSTM
2894 195 XSTM
2895 196 XSTM
2896 197 XSTM
2897 198 XSTM
2898 199 XSTM
2900 EXIT XSTM
FORTRAN CALLING PROCEDURE

CALL XISYT (TKNPR, SYMIND)

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
TKNPR - index into input symbolic string (SSTRING) of token to be
entered or located in symbol table (SYMTAB)

ASGCOM COMMON - LASTSY, SSTRING, STWIDE, SYMTAB, TOKENS

OUTPUT
SYMIND - index into symbol table (SYMTAB) where symbol is located.
VALUE OF SYMIND INDICATES ENTRY NUMBER, i.e., 1, 2, ...

ASGCOM COMMON - LASTSY, 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 0.

EDLOOP - INDEX OF LAST ALLOCATED WORD IN SYMTAB. VALUE IS EQUAL
TO LASTSY*STWIDE

IBUF - INTERNAL BUFFER FOR SYMBOL VALUE USED TO ASSURE LAST
WORDS OF INTEGER AND REAL CONSTANTS ARE ZERO.

EXTERNAL REFERENCES
XRCP, XRNW

SPECIAL REMARKS
THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS
I TYPE (1 WORD) I NAME I UNUSED (4 WORDS) I VALUE I OTHER I
I = TOKEN CODE I (3 WORDS) I RESERVED FOR TOC ENTRY I (3 WORDS) I (IN WORDS) I

### SYMBOL DEFINITION TABLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>2114</td>
</tr>
<tr>
<td>ADD1</td>
<td>1106</td>
</tr>
<tr>
<td>ADD1</td>
<td>1105</td>
</tr>
<tr>
<td>ADD1</td>
<td>1103</td>
</tr>
<tr>
<td>ADD1</td>
<td>1124</td>
</tr>
<tr>
<td>ADD1</td>
<td>1112</td>
</tr>
<tr>
<td>ADD1</td>
<td>1123</td>
</tr>
<tr>
<td>ADD1</td>
<td>1122</td>
</tr>
<tr>
<td>ADD1</td>
<td>1102</td>
</tr>
<tr>
<td>ADD1</td>
<td>1120</td>
</tr>
<tr>
<td>ADD1</td>
<td>1119</td>
</tr>
<tr>
<td>ADD1</td>
<td>1118</td>
</tr>
<tr>
<td>ADD1</td>
<td>1117</td>
</tr>
<tr>
<td>ADD1</td>
<td>1116</td>
</tr>
<tr>
<td>ADD1</td>
<td>1115</td>
</tr>
<tr>
<td>ADD1</td>
<td>1114</td>
</tr>
<tr>
<td>ADD1</td>
<td>1101</td>
</tr>
<tr>
<td>ADD1</td>
<td>1100</td>
</tr>
<tr>
<td>ADD1</td>
<td>1111</td>
</tr>
<tr>
<td>ADD1</td>
<td>1110</td>
</tr>
<tr>
<td>ADD1</td>
<td>1109</td>
</tr>
<tr>
<td>ADD1</td>
<td>1108</td>
</tr>
<tr>
<td>ADD1</td>
<td>1090</td>
</tr>
<tr>
<td>ADD1</td>
<td>1094</td>
</tr>
<tr>
<td>ADD1</td>
<td>1099</td>
</tr>
<tr>
<td>ADD1</td>
<td>1104</td>
</tr>
<tr>
<td>ADD1</td>
<td>1107</td>
</tr>
<tr>
<td>ADD1</td>
<td>1097</td>
</tr>
<tr>
<td>ADD1</td>
<td>1096</td>
</tr>
<tr>
<td>ADD2</td>
<td>1126</td>
</tr>
<tr>
<td>ADD2</td>
<td>1091</td>
</tr>
<tr>
<td>ADD2</td>
<td>1113</td>
</tr>
<tr>
<td>ADD2</td>
<td>1121</td>
</tr>
<tr>
<td>ADD3</td>
<td>1092</td>
</tr>
<tr>
<td>ADD3</td>
<td>1128</td>
</tr>
<tr>
<td>ADD4</td>
<td>1095</td>
</tr>
<tr>
<td>ADD4</td>
<td>1093</td>
</tr>
<tr>
<td>ADD4</td>
<td>1130</td>
</tr>
<tr>
<td>ASGN</td>
<td>272</td>
</tr>
<tr>
<td>ASTEM</td>
<td>2334</td>
</tr>
<tr>
<td>BADMLE</td>
<td>2393</td>
</tr>
<tr>
<td>BADMUL</td>
<td>2329</td>
</tr>
<tr>
<td>BADRNG</td>
<td>2701</td>
</tr>
<tr>
<td>CALSTY</td>
<td>1098</td>
</tr>
<tr>
<td>CALSTY</td>
<td>1132</td>
</tr>
<tr>
<td>CHAR</td>
<td>2304</td>
</tr>
<tr>
<td>CHAR</td>
<td>2590</td>
</tr>
<tr>
<td>CHRFB</td>
<td>937</td>
</tr>
<tr>
<td>CHRSTR</td>
<td>931</td>
</tr>
<tr>
<td>COMPRR</td>
<td>2389</td>
</tr>
<tr>
<td>DAT1</td>
<td>2499</td>
</tr>
<tr>
<td>DQOSP</td>
<td>423</td>
</tr>
<tr>
<td>DREF</td>
<td>2208</td>
</tr>
<tr>
<td>DEFINE</td>
<td>483</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>2125</td>
</tr>
<tr>
<td>ENDCAS</td>
<td>2325</td>
</tr>
<tr>
<td>ENDO</td>
<td>2382</td>
</tr>
<tr>
<td>ENDOC</td>
<td>641</td>
</tr>
</tbody>
</table>
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 BSETDB ENTRY

INPUT FROM INTERFACE TABLE

DOTYPE - 4CH CHARACTER STRING SPECIFYING LOOP CONDITION TYPE
UNTIL - 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 AWA ON REQUEST TO MANAGER

INTBUF - INTERFACE TABLE (LESS LITERAL AREA)
LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE LITERAL AREA
BOSTK - SEE OUTPUT DEFINITION
BSETDB - 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 - $FRMREW RESET NUMBER OF ENDDO IF RETURN = 3

INPUT/OUTPUT FROM TO AWA

BOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27,1) WHERE N 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 SD, VOLUME IV, FIGURE 1.2-17)
WORDS 8-14 - INTERFACE TABLE ENTRY FOR OPRMD1 (SAME AS
IN ORIGINAL TABLE EXCEPT FOR POSSIBLE NEW
VALUES POINTING INTO LITERAL AREA)
WORDS 15-21 - INTERFACE TABLE ENTRY FOR OPRMD2 (SEE
ABOVE)
WORDS 22-25 - LITERAL AREA FOR VALUE(S)/SUBSCRIPT(S) OF
OPRMD1 & 2
WORD 26 - RELATION ID CODE AS FOLLOWS

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

WORD 27 - SEQUENCE NUMBER OF TOP OF LOOP
124 1 BEGIN DO
125 2 CALL XPGET TO INITIALIZE ACCESS TO XPRER AND TO RETRIEVE DTYPE AND RELAT
126 3 CALL XVPA TO RETRIEVE BSTOK INTO BUFFER
127 4 IF RETRIEVAL FAILED, THEN
128 5 INITIALIZE BUFFER FOR BUILDING FIRST BSTOK ENTRY
129 6 ENDF
130 7 IF BSTOK IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
131 8 IF XPRER INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXIST, THEN
132 9 CALL XPRER TO RETRIEVE LITERALS
133 10 ENDF
134 11 IF RELAT IS A VALID RELATION OPERATOR, THEN
135 12 SET RELATION CODE IN NEW ENTRY IN BUFFER
136 13 SET INTERFACE TABLE HEADER WITH NAME OF BINTAB AND NUMBER OF PARAMETER; OF 2
137 14 DO FOR EACH OPND
138 15 MOVE OPND ENTRY INTO NEW INTERFACE TABLE BUFFER
139 16 IF OPND HAS LITERAL VALUE OR DOUBLE SUBSCRIPTS, THEN
140 17 MOVE LITERAL DATA
141 18 ADJUST LITERAL POINTERS
142 19 ENDF
143 20 ENDDO
144 21 CALL XPRER TO RETRIEVE @SERTB (EXECUTING SEQUENCE TABLE)
145 22 IF @SERTB DISPLACEMENT (@SEQDSP) < 0, THEN
146 23 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (0 IF END OF TABLE)
147 24 ELSE INSERTED COMMAND
148 25 EXIT TO :ERROR2: IF SEQUENCE NUMBER IS ZERO (MANUAL)
149 26 LOCATE ORIGINAL SEQUENCE ENTRY
150 27 IF ORIGINAL ENTRY WAS ALSO A DO (OVERWRITE CONDITION), THEN
151 28 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (OR ZERO)
152 29 ELSE (INSERT)
153 30 SET TOP OF LOOP TO CURRENT NUMBER
154 31 ENDF
155 32 ENDF
156 33 CASE (:WHILE;, :UNTIL;, :OTHER;) DTYPE
157 34 :WHILE:
158 35 INVERT RELATION CODE
159 36 INITIALIZE NEST COUNTER TO 1
160 37 START SEARCH FROM TOP OF LOOP ENTRY UNTIL ALL ENTRIES HAVE BEEN EXAMINED
161 38 IF COMMAND IS ENDDO, THEN
162 39 DECREMENT NEST COUNTER
163 40 ELSE
164 41 IF COMMAND IS ANOTHER DO, THEN
165 42 INCREMENT NEST COUNTER
166 43 ENDF
167 44 ENDF
168 45 EXIT IF NEST COUNTER IS ZERO
169 46 SET RESET NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
170 47 END LOOP
171 48 EXIT TO :ERROR4: FOR NO MATCHING ENDDO
172 49 END SEARCH
173 50 :UNTIL:
174 51 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
175 52 :OTHER:
176 53 TERMINATE WITH ERROR FOR UNRECOGNIZED DTYPE
177 54 END CASE
178 55 CALL XPRER TO OUTPUT NEW EXPANDED BSTOK
179 56 ELSE INVALID RELATION
1 CD#####
1 CDO
1 CDO
1 CDO
1 CDO
1 CDO
1 CD#####
1 DO1
1 DO1
1 DO1
1 DO1
1 DO1
1 DO1
1 DO1
1 CD#####
1 CD2
1 CD2
1 CD2
1 CD2
1 CD2
1 CD2
1 CD#####
1 CD3
1 CD3
1 CD3
1 CD3
1 CD#####
1 CD5
1 CD5
1 CD5
1 CD5
1 CD5
1 CD5
1 CD5
1 CD#####
1 CD6
1 CD6
1 CD6
1 CD6
1 CD#####
1 CD7
1 CD7
1 CD7
1 CD7
1 CD#####
1 CD8
1 CD8
1 CD8
1 CD8
1 CD#####
1 CD9
1 CD9
1 CD9
1 CD9
1 CD#####
1 CD10
1 CD10
1 CD10
1 CD10
1 CD#####
1 CD11
1 CD11
1 CD11
1 CD11
1 CD#####
1 CD12
1 CD12
1 CD12
1 CD12
1 CD#####
1 CD13
1 CD13
1 CD13
1 CD13
1 CD#####
1 CD14
1 CD14
1 CD14
1 CD14
1 CD#####
1 CD15
1 CD15
1 CD15
1 CD15
1 CD####
BEGIN ELSE
2 CALL RMPAR TO GET SCHEDULING PARAMETERS
3 CALL ZIFCL TO ESTABLISH FDS MANAGER'S CLASS NO. (XPCLS)
4 CALL ZISCM TO SEARCH FOR MATCHING ENOIF COMMAND
5 CALL XPIT TO RETURN PARAMETERS TO FDS MANAGER
6 END ELSE
**ENDDO - LOOP TERMINATION PROCESSOR**

SCHEDULED BY FD

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FDSD 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

**INPUT/OUTPUT FROM/TO AMA**

**BDOSTK** - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED (27,N) WHERE N IS THE CURRENT NUMBER OF ACTIVE LOOPS.

**SEE DO PROCESSOR FOR CONTENTS**

**THE FDSD SYSTEM STATUS TABLE, VYSTB, IS ACCESSED FOR THE MANAGER/PROCESSOR CLASS 1/0 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

**MRBUFF** - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD ENTRIES ARE OF THE FORM

**REX** - AWS 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

**CLASSNO** - CLASS 1/0 NUMBER THROUGH WHICH DATA IS TRANSMITTED

**XPCLNS** - CLASS 1/0 NUMBER FOR MANAGER COMMUNICATIONS

**REFERENCES ROUTINES**

EXEC, IAND, SMPAR, XPGET, XPREG(XPGET), XPXIT, XRLOC, XRMV,

XUPD6, XVPAW, XIMSG
1 BEGIN ENDDO
2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XVSTB
3 IF RETRIEVAL WAS SUCCESSFUL, THEN
4 WRITE GINTAB TO CLASS I/O FROM LAST &DOSTK ENTRY
5 WRITE NON-LITERAL PORTION OF GINTAB TO CLASS I/O (LEAVE FOR XPGET)
6 CALL XVPAW TO RESTORE &INTAB FROM FIRST CLASS BUFFER INTO AWA
7 IF RESTORE SUCCESSFUL, THEN
8 CALL XPGET TO RETRIEVE OPRMD
9 CALL XEVIL TO EVALUATE RELATION
10 IF RELATION IS TRUE, THEN
11 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
12 IF THIS IS LAST &DOSTK ENTRY, THEN
13 DELETE &DOSTK FROM AWA
14 ELSE
15 CALL XPREG TO REPLACE &DOSTK LESS LAST ENTRY
16 ENDIF
17 ELSE
18 SET RESET NUMBER TO TOP OF LOOP
19 ENDIF
20 ELSE
21 TERMINATE WITH ERR06 FOR AWA OVERFLOW
22 ENDIF
23 ELSE
24 TERMINATE WITH ERR03 FOR ENDDO WITH OUT DO
25 ENDIF
26 END ENDDO
BEGIN IF
CALL DMPAR TO GET INPUT (SHEPPING) PARAMETERS
CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSLATE IT TO A CODE
ERREXIT IF RELATIONAL OPERATOR INVALID :ERR03:
CALL XIEVL TO EVALUATE THE RELATIONAL EXPRESSION
IF THE EXPRESSION IS FALSE, THEN
CALL XICHC TO LOCATE THE ELSE OR ENL': CORRESPONDING TO THIS IF
AND SET SEQUENCE RESET NUMBER
ELSE
CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE
ENDIF
ENDIF
:ERR03: CALL XIMSG 'INVALID RELATIONAL OPERATOR - MUST BE #,>,>,#,=,<=,OR =>
END IF
FORTRAN CALLING PROCEDURE

IVAL = XIEV(L, RELAT, OPRND2)

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

INPUT
OPRND1 - FIRST REAL VALUE TO COMPARE
RELAT - RELATION CODE AS FOLLOWS
1 - GREATER THAN
2 - GREATER THAN OR EQUAL
3 - EQUAL
4 - LESS THAN OR EQUAL
5 - LESS THAN
OPRND2 - SECOND REAL VALUE TO COMPARE

OUTPUT
XIEV - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

INTERNAL VARIABLES
TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS
FOLLOWS
OPRND1 - OPRND2
OPRND1 - OPRND2

RELATION + 0 -

0 1 2 3 4 5
T F F T F F
T F F T F F
F T F T F T
F T F T F T
F T F T F T

(by complementing code > 2)

THEREFORE FOR OPRND1 - OPRND2 POSITIVE (BITS 0-2) 1, 2, 1
FOR OPRND1 - OPRND2 ZERO (BITS 3-5) 0, 0, 1
AND FOR OPRND1 - OPRND2 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
XEXIT

NO CHECKS FOR VALID RELATION CODES OR OVERFLOW/UNDER FLOW ARE MADE
1 BEGIN X:EV.
2 CASE (:+;, :0;, :=-) DIFFERENCE OF OPRND1 AND OPRND2
3 :+; SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)
4 :0; SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
5 :=- SET FIELD OFFSET TO SIX (BITS 6-8 OF TTABLE)
6 END CASE.
7 IF RELATH > 2 (BOTTWN OF TRUTH TABLE), THEN
8 COMPLEMENT TTABLE
9 DECREMENT RELATH BY 3
10 ENDIF
11 ADD RELATH TO FIELD OFFSET (INDEXES TO CORRECT BIT FOR RELATH AND DIFFERENCE)
12 SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
13 END X:EV.
**FORTRAN CALLING PROCEDURE**

**CALL ZISCH** (SRCHFG, PPARMS)

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

**INPUTS**

**SRCHFG** - Flag indicating origin 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

**OUTPUTS**

**RPARMS** - Parameters to be returned to FBS manager via XPPIT

**Routines Used**

- EXEC
- RPAR
- XEVPA
- CUSR
- XRCR
- XINS
1 BEGIN XISCH
2 RETRIEVE &SETTAB FROM THE AWA USING XPAW
3 EXIT IF CURRENT COMMAND IS FOUND
4 SET NUMBER OF IF NESTS TO 1
5 STARTSEARCH FROM NEXT COMMAND IN &SETTAB UNTIL ALL FOLLOWING COMMANDS
6 IF COMMAND IS FOR ENDIF PROCESSOR, THEN
7 DECREMENT NUMBER OF IF NESTS BY 1
8 ELSE
9 IF COMMAND IS FOR IF PROCESSOR, THEN
10 INCREMENT NUMBER OF IF NESTS BY 1
11 ELSE
12 IF CALLED BY IF PROCESSOR, AND
13 COMMAND IS FOR ELSE PROCESSOR, THEN
14 EREREIT IF THIS IS THE END OF &SETTAB :ERRO1:
15 IF NUMBER OF IF NESTS IS 1, THEN
16 DECREMENT NUMBER OF IF NESTS TO 0
17 ENDIF
18 ENDIF
19 EXIT IF NUMBER OF IF NESTS IS 0
20 SET SEQUENCE RESET NUMBER (RPARAMS(2)) TO SEQUENCE NUMBER OF THE
21 NEXT COMMAND IN THE TABLE
22 ENDOLOOP
23 EREREIT :ERRO1:
24 ENDSCH
25 ENDSCH
26 EREREIT :ERRO4:
27 ENDSCH
28 ENDSCH
29 1 EXIT XISCH
30 :ERRO1: CALL XZMSG - "IF CANNOT BE EXECUTED WITHOUT MATCHING ENDIF"
31 :ERRO4: CALL X MSG - "SYSTEM ERROR - NO &SETTAB"
32 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.
<table>
<thead>
<tr>
<th>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>PSRNODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOR2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>55</td>
<td>0 0 1</td>
<td>1793</td>
<td></td>
</tr>
<tr>
<td>FISTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:07</td>
<td>2</td>
<td>30</td>
<td>0 0 1</td>
<td>1843</td>
<td></td>
</tr>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:32</td>
<td>2</td>
<td>30</td>
<td>0 0 1</td>
<td>1875</td>
<td></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>0 0 1</td>
<td>1932</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>0 2 3</td>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>0 1 2</td>
<td>2103</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ABSOLUTE</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>280</td>
<td>SET</td>
<td>2105</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>ELT SYMB</td>
<td>08 NOV 77</td>
<td>05:47:05</td>
<td>8</td>
<td>159</td>
<td>6 5</td>
<td>2345</td>
<td></td>
</tr>
<tr>
<td>XS</td>
<td></td>
<td>ELT SYMB</td>
<td>10 NOV 77</td>
<td>06:19:22</td>
<td>9</td>
<td>394</td>
<td>2 3</td>
<td>2494</td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>23:22:28</td>
<td>10</td>
<td>35</td>
<td>6 5</td>
<td>2868</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>373</td>
<td>8 5</td>
<td>2923</td>
<td></td>
</tr>
<tr>
<td>XL</td>
<td></td>
<td>ELT SYMB</td>
<td>15 FEB 78</td>
<td>19:17:22</td>
<td>12</td>
<td>824</td>
<td>9 5</td>
<td>3296</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>12:48:36</td>
<td>13</td>
<td>627</td>
<td>14 5</td>
<td>4120</td>
<td></td>
</tr>
<tr>
<td>XV</td>
<td></td>
<td>ELT SYMB</td>
<td>22 FEB 78</td>
<td>23:50:36</td>
<td>14</td>
<td>320</td>
<td>20 5</td>
<td>4747</td>
<td></td>
</tr>
<tr>
<td>COMM:CN</td>
<td></td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>22:20:04</td>
<td>15</td>
<td>200</td>
<td>20 5</td>
<td>4857</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>RELOCATABLE</td>
<td>11 MAR 78</td>
<td>03:16:51</td>
<td>16</td>
<td>2</td>
<td>19</td>
<td>5057</td>
<td></td>
</tr>
<tr>
<td>FISTURD</td>
<td></td>
<td>RELOCATABLE</td>
<td>11 MAR 78</td>
<td>03:17:36</td>
<td>17</td>
<td>1</td>
<td>5</td>
<td>5088</td>
<td></td>
</tr>
<tr>
<td>XA</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:29</td>
<td>18</td>
<td>101</td>
<td>10 5</td>
<td>5094</td>
<td></td>
</tr>
<tr>
<td>XP</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:52</td>
<td>19</td>
<td>534</td>
<td>22 5</td>
<td>5195</td>
<td></td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>11:43:48</td>
<td>20</td>
<td>384</td>
<td>6 5</td>
<td>5729</td>
<td></td>
</tr>
<tr>
<td>ZF</td>
<td></td>
<td>ELT SYMB</td>
<td>18 MAY 78</td>
<td>09:12:58</td>
<td>21</td>
<td>1548</td>
<td>4 5</td>
<td>6113</td>
<td></td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>22</td>
<td>403</td>
<td>20 5</td>
<td>7661</td>
<td></td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:39</td>
<td>23</td>
<td>504</td>
<td>20 5</td>
<td>8064</td>
<td></td>
</tr>
<tr>
<td>XU</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 5</td>
<td>8572</td>
<td></td>
</tr>
<tr>
<td>EREXTC</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>25</td>
<td>118</td>
<td>13 5</td>
<td>8723</td>
<td></td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>26</td>
<td>1201</td>
<td>21 5</td>
<td>8841</td>
<td></td>
</tr>
<tr>
<td>XN</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:39</td>
<td>27</td>
<td>600</td>
<td>29 5</td>
<td>10042</td>
<td></td>
</tr>
<tr>
<td>XXI</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:57</td>
<td>28</td>
<td>326</td>
<td>2 3</td>
<td>10968</td>
<td></td>
</tr>
</tbody>
</table>

**NEXT AVAILABLE LOCATION:**

- ASSEMBLER PROCEDURE TABLE EMPTY
- COBOL PROCEDURE TABLE EMPTY
- FORTRAN PROCEDURE TABLE EMPTY
- ENTRY POINT TABLE EMPTY

**DXQT F.POLIST**