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
PREFACE

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

*Combined as one volume with title: Volume III FDS-1 Processor Library*
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2.0 COMMON FOR FDS EXECUTIVE</td>
<td>2-1</td>
</tr>
<tr>
<td>3.0 FDS EXECUTIVE MESSAGES</td>
<td>3-1</td>
</tr>
<tr>
<td>4.0 PDL LISTING PROGRAM</td>
<td>4-1</td>
</tr>
<tr>
<td>5.0 FDS EXECUTIVE DETAILED LOGIC FLOW</td>
<td>5-1</td>
</tr>
<tr>
<td>6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION</td>
<td>6-1</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

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

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

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
  # COMPTX
  # FLAGS
  # RCHNM(3)
  # REBUF(64)
  # REINS
  # SUBSTA
  # TKMLNG
  # TOKEMS(32)
  # XE

DIMENSION
  * INTNAM(3)

EQUIVALENCE
  # (XE(1)) (CLASMD)
  # (XE(2)) (COMPTX)
  # (XE(3)) (FLAGS)
  # (XE(4)) (RCHNM)
  # (XE(5)) (SUBSTA)
  # (XE(6)) (REBUF)
  # (XE(7)) (REINS)
  # (XE(8)) (TKMLNG)
  # (XE(9)) (TOKEMS)
  # (XE(10)) (INTNAM)
  # (XE(11)) (EXTNAM)
  # (XE(12)) (TABEND)
  # (XE(13)) (OLDIND)
  # (XE(14)) (NPROC)
  # (XE(15)) (COMPTX)
  # (XE(16)) (COMBUF)

CARTPG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES
CLASMD - EXECUTIVE/PROCESSOR REQUEST BLOCK CLASS I/O NUMBER
COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER
(1) - NUMBER OF TOKENS IN BUFFER
(2) - NUMBER OF USED WORDS IN BUFFER
(3-256) - TOKENS REPRESENTING USER'S RESPONSE
COMPTX - POINTER TO TOKEN CURRENTLY BEING PROCESSED FROM COMBUF
EXTNAM - SEQUENCE # WHERE EXECUTION IS TO END (RETURN TO X)
FRAGS - EXECUTIVE FLAG WORD (0-OFF, 1-ON)
INTNAM - 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

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

BITS 0-9 - NOT USED

BITS 10-13 - DIRECTIVE CONTROL NOBLE IF BITS 14-15 = 0

0 - LIST
1 - TEC
2 - SAVE
3 - RECALL
4 - DELETE
5 - RENAME
6 - COPY
7 - CLEAR
8 - OFF
9 - STORE
10 - RESTORE
11 - UNLOAD
12 - LOAD
13 - BATCH

BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR

IF BITS 14-15 = 1
0 - INITIALIZATION FROM DIRECTIVE
1 - INITIALIZATION FOR REENTRY

BITS 12-13 - EXECUTION CONTROL NOBLE IF BITS 14-15 = 1

0 - MANUAL
1 - SEMI-AUTOMATIC
2 - AUTOMATIC-T
3 - AUTOMATIC

BITS 14-15 - EXECUTIVE STATE

0 - DIRECTIVE LEVEL
1 - EXECUTION CONTROL LEVEL
2 - SEQUENCE TABLE EDIT LEVEL
3 - INTERFACE TABLE EDIT LEVEL

NPROC - NUMBER OF PROCESSORS IN LIBRARY

OLDIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

PROMAN - NAME OF PROCESSOR FOR WHICH INTERFACE TABLE EDITOR WAS INVOKED OR BEING EXECUTED IN MANUAL, SEMI OR AUTO-WITH-TRACE MODE

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

REBUY - BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SSD 6.2.7.7)

REPTH - POINTER TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUFY (O INDICATES REBUFY EMPTY) OR RETURN CODE FORM XREQ

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

SEGMA - NAME OF SEQUENCE TABLE INPUT TO SEQUENCE TABLE EDITOR OR EXECUTED IN SEMI OR AUTO MODE

SEPNTE - POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SEMI

SIBSTH - INITIAL SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN SEMI OR AUTO MODE

SIBSTA - EXECUTIVE SUB-STATE FLAG (LEVEL IN COMMUNICATION WITH USER TERMINAL). SET TO LEVEL TO BE INITIALIZED NEXT OR ZERO IF LEVEL INITIALIZATION FAILS.
<table>
<thead>
<tr>
<th>INTEGER</th>
<th>NAME LENGTH</th>
<th>NAME SPACE</th>
<th>SIZE</th>
<th>ASCENT</th>
<th>SEEXTAB</th>
<th>R100(150)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ASCE** = SEQUENCE TABLE ENTRY IN ASCII TO PROMPT USER FOR UNIQUE IDENTIFIER OF ENTRY IN PROCESSING SEQUENCE TABLE

**LTRED** = LIBRARY DIRECTORIES AVAILABLE

**NREL** = NAME OF SEQUENCE TABLE

**RELE** = NUMBER OF SEQUENCE TABLES

**SECK** = SEQUENCE TABLE CURRENT ENTRY SELECTED

**NO** = NO OF SEQUENCE TABLES REQUESTED
XU CONTENTS (SEQUENCE TABLE EDIT LEVEL)

191  1 (C)   INTEGER
192  1 (C)   * BEGNO
193  1 (C)   * ENDNO
194  1 (C)   * PROMPT
195  1 (C)   * SEHNO
196  1 (C)   * TABSIZ
197  1 (C)   * MKBUF
198  1 (C)   * XLIBD
199  1 (C)   * OLTAB
200  1 (C)   * NEWTAB
201  1 (C)   * PRCH
202  1 (C)   * PRMLN
203  1 (C)   * PROB
204  1 (C)   * SUBM
205  1 (C)   * SUBR
206  1 (C)   * PRD
207  1 (C)   * LST
208  1 (C)   * EXEC
209  1 (C)   * PGM
210  1 (C)   * PROG
211  1 (C)   * FAC
212  1 (C)   * FAM
213  1 (C)   * FAM
214  1 (C)   * FAM
215  1 (C)   * FAM
216  1 (C)   * FAM
217  1 (C)   * FAM
218  1 (C)   * FAM
219  1 (C)   * FAM
220  1 (C)   * FAM
221  1 (C)   * FAM
222  1 (C)   * FAM
223  1 (C)   * FAM
224  1 (C)   * FAM
225  1 (C)   * FAM
226  1 (C)   * FAM
227  1 (C)   * FAM
228  1 (C)   * FAM
229  1 (C)   * FAM
230  1 (C)   * FAM
231  1 (C)   * FAM
232  1 (C)   * FAM
233  1 (C)   * FAM
234  1 (C)   * FAM
235  1 (C)   * FAM
236  1 (C)   * FAM
237  1 (C)   * FAM
238  1 (C)   * FAM
239  1 (C)   * FAM
240  1 (C)   * FAM
241  1 (C)   * FAM
242  1 (C)   * FAM
243  1 (C)   * FAM
244  1 (C)   * FAM
245  1 (C)   * FAM
246  1 (C)   * FAM
247  1 (C)   * FAM
248  1 (C)   * FAM
249  1 (C)   * FAM

DEBUG = FLAG SET TO VALUE OF EXECUTIVE DEBUG OPTION (BITS 13 - 15)
OF FLAGS (EK(4))

BEGNO = NUMBER IN WBUFF TO FIRST TABLE ENTRY TO BE LISTED OR DELETED

ENDNO = INDEX IN WBUFF TO LAST TABLE ENTRY TO BE LISTED OR DELETED

INSERT = FLAG INDICATING TO INSERT WHAT TYPE OF EDIT TO PERFORM:
= 0 => REPLACE ENTRY AT TABDIX
= 1 => INSERT A NEW ENTRY IN FRONT OF TABDIX
= 2 => ADD AN ENTRY AT THE BOTTOM OF THE TABLE

INTHAN = INTERFACE TABLE NAME INPUT OR 0

IZETC = RETURN CODE SET TO -1 WHEN THERE IS NO TERMINATE

LTDOSZ = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBD

NEWTAB = NAME OF SEQUENCE TABLE BEING CREATED

NUMER = NUMBER OF ENTRIES (INCLUDING THESE MARKS FOR DELETION)

CURRENTLY IN WBUFF

OLDDB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

PRMLN = NAME OF SEQUENCE NUMBER PROMPTED BY XSPR

PRINTM = CURRENT PROMPTING MODE :
= 1 => UPDATE MODE ( # )
= 2 => CREATE OR MODIFY MODE ( # 200: )
= 3 => ALL MODE ( # 200=PROC, TABLE: )

PROMPT = PROMPT BUILT BY XSPR

SEHNO = CURRENT SEQUENCE NUMBER BEING PROMPTED

TABDIX = INDEX IN WBUFF TO CURRENT TABLE ENTRY

TABDIZ = SIZE IN WORDS OF TABLE IN WBUFF

TABDIX = INDEX IN WBUFF TO CURRENT TABLE ENTRY

XKBUFN = MAXIMUM SIZE IN WORDS OF WBUFF

XLIBD = PROCESSOR LIBRARY DIRECTORY
XB CONTENTS (INTERFACE TABLE EDIT LEVEL)

1 CD INTEGER
2 INTEGER
3 CD ARGNO
4 CD BITINO
5 CD BITNUM
6 CD DEBUG
7 CD DIRECT
8 CD DFLAG
9 CD EQNAT(3)
10 CD PRMLEM
11 CD PROMPT(30)
12 CD SFLAG
13 CD VERSION
14 CD WKBUFF(1300)
15 CD DIMENSION
16 CD ISIZE(9)
17 CD NEWTAG(3)
18 CD EQUIVALENCE
19 CD COMFLG
20 CD ICCLASS
21 CD XB(26)
22 CD VERSOM
23 CD XB(28)
24 CD LISTL
25 CD XB(29)
26 CD MODES
27 CD IDIM
28 CD XB(33)
29 CD SFLAG
30 CD XB(36)
31 CD DFLAG
32 CD XB(37)
33 CD ARENO
34 CD XB(38)
35 CD ARPTR
36 CD XB(39)
37 CD ISUB
38 CD XB(40)
39 CD IOFLAG
40 CD XB(41)
41 CD PRMTHD
42 CD XB(42)
43 CD LEEFF
44 CD XB(43)
45 CD LEEE
46 CD XB(45)
47 CD IFLAG
48 CD XB(46)
49 CD MOSF
50 CD XB(47)
51 CD LITDSP
52 CD XB(50)
53 CD LITM
54 CD XB(51)
55 CD ITIM
56 CD XB(52)
57 CD LITPR
58 CD XB(53)
59 CD LITSI
60 CD XB(54)
61 CD LITX
62 CD XB(55)
63 CD LARG
64 CD XB(97)
65 CD NEWTAG(3)
66 CD XB(100)
67 CD WKBUFF
68 CD XB(101)
69 ARGNO - CURRENT ARGUMENT'S NO. (I.E. 1 TO 64)
70 CD ARPTR - INDEX IN WKBUFF TO ARGUMENT PROMPT FOR CURRENT ARGUMENT
71 CD BITINO - BIT NO. IN BIT MASK CURRENTLY BEING PROCESSED
72 CD BITNUM - BIT NO. IN BIT MASK OF NEXT DIFFERING BIT
73 CD COMFLG - FLAG SET TO 1 IF A CONTINUATION PROMPT FOR CURRENT ARGUMENT
74 CD HAS BEEN EXPLICITLY REQUESTED VIA A TRAILING COMMA OR IS
75 CD IMPLICITLY REQUIRED BECAUSE INCOMPLETE ELEMENTS REMAIN BEYOND
76 CD LAST ENTERED ELEMENT OF THIS ARGUMENT
77 CD DEBUG - FLAG SET TO VALUE OF DEBUG FLAG (BITS 13-15 OF XE(4))
78 CD DIRECT - ASCII ARRAY OF VALID INTERFACE TABLE EDIT DIRECTIVES
79 CD DFLAG - FLAG SET TO VALUE OF DATA EDIT FOR THIS ARGUMENT
80 CD IARG - INDEX IN WKBUFF TO CURRENT ARG
81 CD ICCLASS - CLASS OF THIS ARGUMENT
82 CD IOFLAG - FLAG SET TO I/O BITS FOR THIS ARGUMENT
83 CD IRET - GENERAL RETURN FLAG (0=OK, -1=ERROR, 5='EXIT')
84 CD ISIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
85 CD ISIZE - NO. OF WORDS/ELEMENT FOR EACH ALLOWED IVALUE
86 CD ISUB - CURRENT EFFECTIVE SUBSCRIPT FOR THIS ARGUMENT
87 CD ITYPE - TYPE FLAG FOR THIS ARGUMENT
JSUB - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
LENFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
LIST1U - LU TO WHICH PRINT SHOULD GO (USED WHEN
LIST DIRECTIVE CALLS XILSD OR XICHR)
LITSDF - DISP. TO LITERAL DATA FOR THIS ARGUMENT
LISTDL - INDEX IN WKBUFF TO END OF LITERAL DATA
LITLEN - LENGTH OF LITERAL DATA AREA OF WKBUFF
LITPR - INDEX IN WKBUFF TO START OF LITERAL DATA
LISIZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
LISTFLG - FLAG USED TO DETERMINE ORIGIN OF A
CALL TO XILSD OR XICHR.

MODAV - PREVIOUS VALUE OF PRMTMD WHILE PRMTMD = 4 (CONTINUE)
MARG - INDEX IN WKBUFF TO START OF SHORT PROMPTS
MDNBTM - INDEX IN WKBUFF TO BIT MASK WORD(S) FOR THIS ARGUMENT
MENDAB - ASCII NAME OF TABLE BEING GENERATED
MODIM - NO. OF BIT MASKS ASSOCIATED WITH THIS ARGUMENT
NUMARG - NO. OF ARGUMENTS IN THIS INTE. TABLE
NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
PRMLEN - LENGTH (IN WORDS) OF PROMPT BUILT
PRMTMD - CURRENT PROMPTING MODE

* 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
* 3, PROMPT WITH A LIST OF CURRENT VALUES
* 4, CONTINUATION OF SAME ARGUMENT
* 5, PROMPT WITH "/" (CREATE MODE)

PROMPT - ASCII ARRAY FOR PROMPT BUILT
SFLAG - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
VERSON - VERSION NO. OF THIS INTERFACE TABLE
WKBLEN - LENGTH OF WKBUFF
WKBUFF - WORKING BUFFER FOR INTERFACE TABLE BEING EDITED

ORGANIZED AS :

HEADER (7 WORDS)

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

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

1  * FDS ERROR MESSAGES
2  *
3  *
4  * XA  ATTENTION FUNCTION
5  * XA0 MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION
6  * XA00 MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION
7  * XA01 USER INITIATED INTERRUPT ENTER REQUEST - KILL(?) , STATUS(S) , RETURN( )
8  * XA02 USER INITIATED INTERRUPT ENTER REQUEST - KILL(?) , STATUS(S) , RETURN( )
9  * XA03 ERROR, LU IS NOT SIGNED ON TO FDS
10  * XA04 FDS ATTENTION FUNCTION TERMINATING
11  * XA05 FDS MANAGER SIGNALS TO TERMINATE BOTTOM ASSOC TASK
12  * XA06 FDS EXECUTIVE ACTIVE - NO ACTION TAKEN
13  * XA07 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT
14  * XA08 MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN
15  * XB  BATCH JOB CREATION
16  *
17  * XC  CONFIGURATION PROGRAMS
18  * XC01 LU 'NN' SIGNED ON TO FDS
19  * XC02 LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED
20  * XC03 LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED
21  * XC04 'LU' IS AN INVALID LU
22  * XC05 FDS CURRENTLY AT MAX USER'S. CANNOT SIGN ON
23  * XC06 'LU' IS ALREADY SIGNED ON TO FDS
24  * XC07 CANNOT FIND 'NAME' ID - SIGN ON TERMINATED
25  * XC08 ENTER VALID USER ID (A-Z)
26  *
27  * XM  MANAGER
28  * XM01 INVALID REQUEST II FROM 'NAME'
29  * XM02 FDS SIGN OFF FOR LU 'LU'
30  * XM03 INVALID BACK CHAIN FOR CURRENT PROCESSOR
31  * XM04 'H' TRACKS NOT AVAILABLE FOR DWA
32  * XM05 NO DWA SPACE FOR 'NAME'
33  * XM06 NO AWA FOR DWA DIRECTORY
34  * XM07 I/O ERROR FOR DWA, STATUS = 'NNNNNN'
35  * XM08 TOO SEARCH ERROR, PHASE 3 COMPRESS
36  *
37  * XP  PROCESSOR SERVICES
38  * XP01 TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED
39  * XP02 DATA AREA OVERFLOW
40  * XP03 SUBSCRIPT OUT OF RANGE
41  * XP04 INVALID ENTRY AFTER SUBSCRIPT
42  * XP05 INVALID REPEAT ENTRY
43  * XP06 TOO MANY NESTED REPEATS
44  * XP07 INVALID SUBSCRIPT
45  * XP08 RESPONSE IS TOO LONG FOR BUFFER
46  * XP09 INVALID RESPONSE ON OR AFTER COLUMN NN
47  * XP10 PROCESSOR INITIALIZATION ERROR
48  * XP11 AWA ACCESS FAILURE FOR ..........
49  * XP12 PARAMETER I/O INCONSISTENCY
50  * XP13 INVALID PARAMETER REQUEST
51  * XP14 INVALID ORDE FILE ATTRIBUTE(S)
52  * XP15 RETRIEVAL OF TOO MUCH DATA REQUESTED
53  *
54  * XV  SYSTEM SERVICES
55  * XV01 'ID-NAME' NAMED PROGRAM MADE A PAM REQUEST RESERVED FOR MANAGER
56  * XV02 'ID-NAME' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER REPLY
57  * 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 CD1
2 CD1
3 CD1
4 CD1
5 CD2
6 CD2
7 CD2
8 CD2
9 CD2
10 CD2
11 CD2
12 CD2
13 CD2
14 CD2
15 CD2
16 CD2
17 CD3
18 CD3
19 CD3
20 CD3
21 CD3
22 CD3
23 CD3
24 CD3
25 CD3
26 CD3
27 CD3
28 CD3
29 CD3
30 CD3
31 CD3

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 . THE
token from the previous . record continues to appear in the
identification columns of the output
- If 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
•-rrrr

rr ►-rh^I h ^•I F• h
NN N MNNM NMhhhhr
NMMNNM MMM
NN 1..NMrhrrrrhrr
NM NM NM«NMNM NM NM NMrrrrrrhrrr
NM NM NM NM MN MN MNNM NM N►-rrrrhrrrrrrrrrh
N
l•lM
M NM NM MN MN MN MN MN MN MNMNM NM NM NM NM NM NM NM MN MN MN MN MN MN 1.1
^0 d0. 0. dsLdiL0, iL4G^^ d4.LLd0

d d d 4L4 d4d O

Y.^

i

d
W
N
♦0.

• W

Z v'

W
N•• ••OW

a
o •♦

W

M d

WN .•N

♦

^[
W

wY N
WN ••\

•• A
W

i
O

Y
J

O
•
K

^.

M
V.

M
Z
M
A
W
O

1C
JOCID
0
Z
r

=

h^
! r

<
J

W
•J>

401

M.

W/M O

J

M

N
W

i

V pp^^
qq^^"000.O
900,
2«rr
0 0 ►-1•1-•rrrhhrrrl-r
000000000
H F««
F H
VWW••
4 H

<KK<«KKK«KKK< ZZii NN
••••

.0
6ZYYVVVYV^70[KKOCKOK MMM«
Mr^l«MMMNMK` «K<J
MO OOOOOOGdY•OG 4i G
J
89 OW
0
r i Z Z ZZ2Z2 W W W W W WAY O W WW W

W

♦ ♦_ h

h
t O G J

Y« « Z« M M««« N N N N N N /zzli h h hs rZ Z••••i
A
64
M
O
h
O
h
N0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
N1
0
0
0
0
V
Y
••
h
««
«
M
«
M«
M
M«
M
M
M
M
M
O
M
M
M
M
Z
Z
W
W
M r 0 0
«
r
h
r
r
h
r
r
r
r
r
h
h
h
h
h
h
h
h
r
NN
•.
Z ^
YI
VL•YYVV
VV
V
VYYYOYVYY
Y
V
Z
O«
•
•••i
I W ^jON m N NW WN Y:W
W
W
W
W
W
W
W
W
W
W
W
W
i
W
W
W
W
O
NNNNNNNNNNNNWNNNN
♦ \W WN
O
J r w wdig 1 1 1 1 1 1 1 1/ 1 1! 1 1 1 1 1 1 1 1 1
ZO
W W
•+i
Wt ^iWL
•\
it lhri o
OMA N
• N
•• s i M Y
N

«M

I.W.

WO W

%.0.. A

i t Vi
N< t
♦ ••
K
MM
MWN Y.
p tW yy 0••
r
p
i WiWYh64
f « J« hhN W
W
i
Oi
<
«
^
MOVN
O
MO
Mt
OMhh r O
N«N MM 000,00
ri ••
i Wi W
i• VrMY.
W1►
=64 r
OM OO O O Y<hhJT1[fl
N N W r W W O W W W ii
W W W O <MM
O W=

O V
♦ W KW
iu hNi

Oh
W

J
tW
J
J

ZZ
O
M
h
V
W
N
O
O
W
Y
s

M

Y
Z
i

W
J

faM
V
pW
M

^►

i O N O N O O K

V J W
V
WW Fi•Y. psy rNmwifto ft1610
JyI::
V ZO i v
W Y 0-0 OO
N
O
w< wo
wwwwwww^N s Nvv O 40
v«Ji • WO O M
W^ N J W J ' 1L
r
11
OM
WO
1WLiY
i OM
?Z
JO
aY.
W
W
hIKO
W«
WO
MW
Y.i
O ppONr
1
Mh
NOOh^ K! h J^ IrW N{YW V..1
WWWW ONtZNM <Y!•.I$. w•YYe.
WY.
NJM a~
iJ O W^ Wm
o- W^,Jwri
V M r
.N
,/
JJ M 0 i Y
.
OM
WZl►
S
W
dr149MMh
«

iiO
«"a
O
W
m

• N M« r

• i• M r N M« M i•• r« r b M

^NNM1MlNIPI f f f Y1 f M f f f NN •O.O N.O.O.O.O.O.O.O.O.O .O•O W .0.0.0.0 •Y •0 .0 .0 .0 ^O.O ^O.O.O.O.O OA^O p
MCI in M

OO

A
OwomrO/:: ^VOIN NNYf1NNV 1111PR•00.0.0.V0•f0 •N0•%010 .00AAAAAAAAAA gam 400 p

4-3

o


;SEP: SET LEVEL INCREMENT = 1
DECREMENT LEVEL

;END?: CALL FSTWRD 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
116 1 CD1
117 1 CD1
118 1 *********
119 1 CD2
120 1 CD2
121 1 CD2
122 1 CD2
123 1 CD3
124 1 CD3
125 1 CD3
126 1 CD3
127 1 CD3
128 1 *********
129 1 *
130 1 *
131 1 *
132 1 BEGIN FSTWRD
133 2 BLANK OUTPUT WORD
134 2 LOCATE FIRST NON-BLANK CHARACTER
135 2 DO UNTIL SIX CHARACTERS STORED OR END-OF-RECORD
136 3 IF CHARACTER IS NON-BLANK AND NON-:
137 3 THEN
138 4 STORE CHARACTER
139 4 ELSE
140 4 EXIT DO
141 4 ENDOF
142 2 EDDO
143 2 Z RETURN LOCATION
144 1 END FSTWRD
ALGEBRAIC AND/OR ALPHABETIC ARRAY SORT

ENTRY POINT INTO SUBROUTINE SORT2

THIS ROUTINE WAS EXTRACTED FROM THE MDAS SUBMONITOR PROGRAM FOR USE IN
POLIST. DOCUMENTATION MAY BE FOUND IN 'LEVEL II MDAS PROTOTYPE
MONITOR PROGRAM DOCUMENT (PART II),' TRW NOTE NO. 74-FMT-937,
14 JUNE 1974.
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.
**NAME**

**DEFINITION**

**DIRECTIVES**

1. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

2. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE ROUTINES

3. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

4. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

5. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

6. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

7. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

8. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

9. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

10. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

11. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

12. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

13. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

14. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

15. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

16. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

17. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

18. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

19. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

20. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

21. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

22. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

23. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

24. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

25. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

26. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

27. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

28. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

29. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

30. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

31. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

32. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

33. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

34. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

35. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

36. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

37. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

38. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

39. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

40. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

41. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

42. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

43. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

44. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

45. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

46. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

47. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

48. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

49. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

50. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

51. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

52. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

53. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

54. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

55. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

56. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

57. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

58. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

59. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

60. **KDLA**
   INTERFACE ROUTINE FOR DATA BASE DIRECTIVES

**EXECUTIVE PROGRAM MAIN LOGIC**

1. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

2. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

3. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

4. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

5. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

6. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

7. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

8. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

9. **EXECUTE**
   EXECUTION SEGMENT INTERFACE ROUTINE

10. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

11. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

12. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

13. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

14. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

15. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

16. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

17. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

18. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

19. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

20. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

21. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

22. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

23. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

24. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

25. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

26. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

27. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

28. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

29. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

30. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

31. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

32. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

33. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

34. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

35. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

36. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

37. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

38. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

39. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

40. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

41. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

42. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

43. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

44. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

45. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

46. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

47. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

48. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

49. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

50. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

51. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

52. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

53. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

54. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

55. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

56. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

57. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

58. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

59. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE

60. **EXECUTE**
    EXECUTION SEGMENT INTERFACE ROUTINE
<table>
<thead>
<tr>
<th>Page</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>ITE USER RESPONSE PROCESSOR</td>
</tr>
<tr>
<td>62</td>
<td>ITE OVERLAY INTERFACE ROUTINE</td>
</tr>
<tr>
<td>63</td>
<td>ITE PARAMETER DATA PROCESSOR</td>
</tr>
<tr>
<td>64</td>
<td>SITE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>65</td>
<td>SITE PROMPT CONSTRUCTOR</td>
</tr>
<tr>
<td>66</td>
<td>SITE SUBSCRIPT PROCESSOR</td>
</tr>
<tr>
<td>67</td>
<td>EL LIBRARY MAINTENANCE PROGRAM</td>
</tr>
<tr>
<td>68</td>
<td>ELCDB CREATES A HUD/PDB DATA BASE FILE</td>
</tr>
<tr>
<td>69</td>
<td>ELBMD MAIN DATA BASE FILE SEGMENT</td>
</tr>
<tr>
<td>70</td>
<td>ELDDEL DELETES A PROCESSOR FROM LIBRARY DIRECTORY</td>
</tr>
<tr>
<td>71</td>
<td>ELCLFL CREATES A DEFAULT INTERFACE TABLE</td>
</tr>
<tr>
<td>72</td>
<td>ELINS DISPLAYS PARAMETER SPECIFICATIONS INSTRUCTION</td>
</tr>
<tr>
<td>73</td>
<td>ELINT MAIN LOGIC FOR DEFAULT VALUES</td>
</tr>
<tr>
<td>74</td>
<td>ELRAN MAIN MAINTENANCE *PROGRAM</td>
</tr>
<tr>
<td>75</td>
<td>ELMOD MODIFIES A PROCESSOR INTERFACE TABLE</td>
</tr>
<tr>
<td>76</td>
<td>ELMSG MAINTAINS FBS MESSAGE FILE</td>
</tr>
<tr>
<td>77</td>
<td>ELPAN CREATES HUD/PDB LOG FILE</td>
</tr>
<tr>
<td>78</td>
<td>ELPFL CREATES A PROCESSOR PROMPT TABLE</td>
</tr>
<tr>
<td>79</td>
<td>ELPRO MODIFIES HUD/PDB LOG FILE</td>
</tr>
<tr>
<td>80</td>
<td>ELPRN CREATES SYSTEM PROMPT FILES</td>
</tr>
<tr>
<td>81</td>
<td>ELVPS ADDS PROMPT FILES</td>
</tr>
<tr>
<td>82</td>
<td>ELVPSAD ADDS PROMPT FILES</td>
</tr>
<tr>
<td>83</td>
<td>EN MANAGER</td>
</tr>
<tr>
<td>84</td>
<td>ENMAN AWA FREE SPACE</td>
</tr>
<tr>
<td>85</td>
<td>ENMANX AWA MANAGEMENT PROCESSING (XINC EP)</td>
</tr>
<tr>
<td>86</td>
<td>ENEXGX GENERAL PACK INTERFACE (XIPCK EP)</td>
</tr>
<tr>
<td>87</td>
<td>ESHAVA DATA AND CONTROL DATA MAPPING</td>
</tr>
<tr>
<td>88</td>
<td>EKMD1 DMA ALLOCATION (XMDMA EP)</td>
</tr>
<tr>
<td>89</td>
<td>EKMDA DMA DEALLOCATION (XMDMA EP)</td>
</tr>
<tr>
<td>90</td>
<td>EKMDIM DMA INITIALIZATION (XMDMA EP)</td>
</tr>
<tr>
<td>91</td>
<td>EKMDRT DMA RETRIEVE (READ) (XMDMA EP)</td>
</tr>
<tr>
<td>92</td>
<td>EMRT DMA STORE (WRITE) (XMDMA EP)</td>
</tr>
<tr>
<td>93</td>
<td>EMRTA DMA MANAGEMENT</td>
</tr>
<tr>
<td>94</td>
<td>ENSF FBS MANAGER</td>
</tr>
<tr>
<td>95</td>
<td>ENSFPM POST AND WAIT INTERFACE ROUTINE (XINC EP)</td>
</tr>
<tr>
<td>96</td>
<td>ENSFK AWA COMPRESS</td>
</tr>
<tr>
<td>97</td>
<td>ENSFK1 PHASE 1 COMPRESS (PURGE TO DMA) (XIPCK EP)</td>
</tr>
<tr>
<td>98</td>
<td>ENSFK2 PHASE 2 COMPRESS (toc COMPRESS) (XIPCK EP)</td>
</tr>
<tr>
<td>99</td>
<td>ENSFK3 PHASE 3 COMPRESS (SPACE ORDER) (XIPCK EP)</td>
</tr>
<tr>
<td>100</td>
<td>ENSFM TOC LOCK-UP ROUTINE</td>
</tr>
<tr>
<td>101</td>
<td>ENSRT SEQUENCE LOCATION COUNTER RESET</td>
</tr>
<tr>
<td>102</td>
<td>ENSRJX SEQUENCE TABLE EXECUTION</td>
</tr>
<tr>
<td>103</td>
<td>OSP PROCESSOR SERVICES</td>
</tr>
<tr>
<td>104</td>
<td>OSPAT Processor parameter attribute retrieval</td>
</tr>
<tr>
<td>105</td>
<td>OSPEP Processor parameter retrieval</td>
</tr>
<tr>
<td>106</td>
<td>OSPRTI Indexed parameter retrieval</td>
</tr>
<tr>
<td>107</td>
<td>OSPRTM Processor parameter storage (XIPGET EP)</td>
</tr>
<tr>
<td>108</td>
<td>OSPRTTI Indexed parameter storage (XIPGETT EP)</td>
</tr>
<tr>
<td>109</td>
<td>OSPRDM PROMPTS USER, READS RESPONSE, RETURNS ENCODED BUFFER</td>
</tr>
<tr>
<td>110</td>
<td>OSPRDS CALLS XPROM, DECODES BUFFER, RETURNS RESPONSE IN DATA AREA</td>
</tr>
<tr>
<td>111</td>
<td>OSPRSL Processor service for ana access</td>
</tr>
<tr>
<td>112</td>
<td>OSPRZ Processor termination routine</td>
</tr>
<tr>
<td>113</td>
<td>EX EXECUTIVE SERVICES</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>1</td>
</tr>
<tr>
<td>180</td>
<td>1</td>
</tr>
<tr>
<td>181</td>
<td>1</td>
</tr>
<tr>
<td>182</td>
<td>1</td>
</tr>
<tr>
<td>183</td>
<td>1</td>
</tr>
<tr>
<td>184</td>
<td>1</td>
</tr>
<tr>
<td>185</td>
<td>1</td>
</tr>
<tr>
<td>186</td>
<td>1</td>
</tr>
<tr>
<td>187</td>
<td>1</td>
</tr>
<tr>
<td>188</td>
<td>1</td>
</tr>
<tr>
<td>189</td>
<td>1</td>
</tr>
<tr>
<td>190</td>
<td>1</td>
</tr>
<tr>
<td>191</td>
<td>1</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td>193</td>
<td>1</td>
</tr>
<tr>
<td>194</td>
<td>1</td>
</tr>
<tr>
<td>195</td>
<td>1</td>
</tr>
<tr>
<td>196</td>
<td>1</td>
</tr>
<tr>
<td>197</td>
<td>1</td>
</tr>
<tr>
<td>198</td>
<td>1</td>
</tr>
<tr>
<td>199</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>201</td>
<td>1</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
</tr>
<tr>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>205</td>
<td>1</td>
</tr>
<tr>
<td>206</td>
<td>1</td>
</tr>
<tr>
<td>207</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>1</td>
</tr>
<tr>
<td>209</td>
<td>1</td>
</tr>
<tr>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>211</td>
<td>1</td>
</tr>
<tr>
<td>212</td>
<td>1</td>
</tr>
<tr>
<td>213</td>
<td>1</td>
</tr>
<tr>
<td>214</td>
<td>1</td>
</tr>
<tr>
<td>215</td>
<td>1</td>
</tr>
</tbody>
</table>
BEGIN XATTN

FUNCTION

FDS 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
FDS STRUCTURE FOR THIS TERMINAL.

NOTES

THE ID FOR XATTN IS CONNECTED TO THE USER'S
TERMINAL AT FDS SIGN-ON BY USING THE ERT
FOR THE DEVICE. THE FUNCTION IS DISCONNECTED
AT FDS SIGN-OFF.
17 2 SAVE EGT ADDRESS(IN BREG ON ENTRY)    
18 2 CALL ESLU(BREG) GET LU IN ASCII & BINARY 
19 2 STARTSEARCH UNTIL LAST STATUS TABLE ENTRY 
20 2 EXITIF STBLU EN LU 
21 3 SET STB ENTRY ADDRESS 
22 3 ENDLOOP 
23 2 SET STB ENTRY TO ZERO 
24 3 ENDSEARCH 
25 2 IF STB ENTRY FOUND, THEN 
26 3 GET MANAGER'S ID ADDRESS(STMG) 
27 4 IF MANAGER IS DORMANT, THEN 
28 4 WRITE "***XAO- MANAGER HAS TERMINATED;" 
29 4 REPLY TO CONTINUE TERMINATION ; 
30 4 READ(LU) ** WAIT FOR REPLY ** 
31 4 LOCK ON THE FDS TABLE RESOURCE 
32 4 CALL SLBRM DISABLE 
33 4 IF STBEX(EXECUTIVE ADDRESS) .NE. 0, THEN 
34 5 IF STAT(CURRENT) .NE. 0, THEN 
35 6 IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XMGR, THEN 
36 7 FIND BOTTOM AT 
37 7 DO UNTIL NEXT-AT .ER. STMG(MANGER) 
38 8 CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER 
39 8 CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER 
40 8 CLEAR NEXT-AT'S PARN ONE(P1) 
41 8 CALL SLBRM ENABLE 
42 8 CALL MESS 'OFF,BOTTOM' 
43 8 CALL SLBRM DISABLE 
44 8 SET BOTTOM TO NEXT-AT 
45 7 ENDDO 
46 6 ENDF 
47 6 ENDF 
48 6 CALL SLSTL MAKE EXEC DORMANT 
49 6 CLEAR EXEC'S ID & STBR 
50 4 ENDF 
51 4 DECREMENT NUMBER ACTIVE(STBAC) 
52 4 GET EGT ADDRESS 
53 4 RESTORE INTERRUPT HANDLER(FROM STBEX) 
54 4 CLEAR STBR 
55 4 CLEAR MANAGER'S ID, STMG, & STLG 
56 4 ENABLE.......(VIA A JMP TO EEC(DISPATCHER)) 
57 4 RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS 
58 4 CLEAR LOCK ON FDS TABLE 
59 3 ELSE ** MANAGER IS STILL ALIVE ** 
60 4 WRITE "***XAO- USER INITIATED INTERRUPT?" 
61 4 WRITE 'ENTER REQUEST- KILL(3), STATUS(S), OR RETURN(BLANK)' 
62 4 READ (LU) REQUEST 
63 4 IF REQUEST IS KILL OR 3, THEN 
64 5 PERFORM XAKIL 
65 4 ELSE 
66 5 IF REQUEST IS STATUS OR 5, THEN 
67 6 PERFORM XASTAT 
68 6 ENDF 
69 4 ENDF 
70 3 ELSE 
71 2 WRITE "***XAO- ERROR LU IS NOT SIGNED-ON TO FDS" 
72 2 ENDF 
73 2 WRITE "***XAO- FDS ATTENTION FUNCTION TERMINATING" 
74 1 END XATTN
PRODUCE A FDS STATUS REPORT

1: GO TO XSTAT
2: CALL SLIBR, DISABLE
3: GET CURRENT-TIME FROM TIME
4: GET MANAGER'S ADDRESS FROM STMGR
5: MOVE NAME, STATUS, PARTITION, & PRIORITY
6: GET EXECUTIVE'S ADDRESS FROM STBEX
7: MOVE NAME, STATUS, PARTITION, & PRIORITY
8: GET CURRENT AT FROM STBAT
9: MOVE NAME, STATUS, PARTITION, & PRIORITY
10: PERFORM SUBRNT(CURRENT) FIND BOTTOM AT
11: SET BOTTOM TO CURRENT
12: GO WHILE FATHER-ID NE ZERO
13: SET FATHER-ID FROM CURRENT
14: CALCULATE NEXT
15: IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
16: MOVE NAME, STATUS, PARTITION, & PRIORITY
17: ENDIF
18: IF NEXT IS THE MANAGER, THEN
19: SET CURRENT AS TOP
20: ENDIF
21: SET CURRENT TO NEXT
22: ENDDO
23: IF TOP EQ ZERO, THEN
24: SET TOP TO CURRENT
25: USE TOP TO MOVE NAME, PARTITION, & PRIORITY
26: SET STATUS TO 'IN USE' OCTAL 17
27: ENDIF
28: CALL SLIBX, ENABLE
29: WRITE FIRST SET OF HEADERS
30: SET TOP AS REPORT DATA
31: WRITE REPORT LINE
32: SET MANAGER AS REPORT DATA
33: WRITE REPORT LINE
34: SET EXECUTIVE AS REPORT DATA
35: WRITE REPORT LINE
36: SET CURRENT AS REPORT DATA
37: WRITE REPORT LINE
38: WRITE INTERMEDIATE HEADERS
39: DO UNTIL MAX ENTRIES OR NO MORE DATA
40: WRITE REPORT LINE
41: SET NEXT REPORT DATA
42: ENDDO
43: END XSTAT
44: SAMPLE REPORT
45: FDS STATUS FOR LU 10 HH:MM:SS 360
46: NAME PRIOR PART# STATUS
47: TOP AT- PROC 922 9 GENERAL WAIT
48: MANAGER- X498BH 40 3 GENERAL WAIT
49: EXECUTIVE- X32XX 80 3 GENERAL WAIT
50: CURRENT AT- PROC 11311 4 GENERAL WAIT
51: BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
52: PROC 32767 6 DISC ALLOCATE SUSPEND
53: PROC 11311 4 GENERAL WAIT
54: PROC 20603 3 GENERAL WAIT
55: PROC 845 6 GENERAL WAIT
BEGIN XAKILL
140 1 BEGIN XAKILL
141 2 TERIMINATE CURRENT FDS FUNCTION
142 2 IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
143 2 ON NEXT RETURN VIA A PAW.
144 2 IF THE EXEC IS ACTIVE- Do NOTHING
145 2 IF A PROCESSOR IS ACTIVE- USE RTE MESS5 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 "***XAO5 FDS MANAGER SIGNAL3D TO TERMINATE SEQUENCE'
150 2 ELSE
151 3 IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
152 4 WRITE "***XAO6 FDS EXECUTIVE ACT1VE; NO ACTION TAKEN'
153 3 ELSE
154 4 PERFORM XABTM(CURRENT) FIND BOTTOM AT
155 4 IF BOTTOM AT IS D.RTR OR SNP 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 XAKILL
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 ENDF
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 ENDF
174 4 ENDIF
175 3 ENDF
176 2 ENDF
177 1 END XAKILL
1 BEGIN XCONF

2 * FDS CONFIGURATION MANAGER
3 * D1 INITIATES AN FDS SYSTEM FOR A REQUESTED TERMINAL OR
4 * D1 TERMINATES AN FDS SYSTEM FOR A REQUESTED TERMINAL
5 * D1 INITIATED VIA
6 * R, FDS, LU, ID, DMA SIZE, PARM, OPTIONS
7 * D1 INPUTS
8 * D2 LOGICAL UNIT(LU) FOR THE REQUESTED TERMINAL,
9 * D2 A PARM TO DENOTE INITIATION OR TERMINATION
10 * D2 A ONE CHARACTER USERID
11 * D2 A DEBUG OPTION INDICATOR
12 * D2 THE NUMBER OF DMA TRAKCS
13 * D3 OUTPUTS
14 * D3 INITIATION-
15 * D3 A BLANK ID-SEGMENT WILL BE CONSTRUCTED FOR THE FDS MANAGER,
16 * D3 AND FDS EXECUTIVE
17 * D3 THE ENT FOR THE REQUESTED LU WILL BE CONNECTED
18 * D3 TO THE FDS ATTENTION TASK
19 * D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
20 * D3 WILL BE UPDATED TO REFLECT THE INITIATION
21 * D3 TERMINATION-
22 * D3 THE FDS ID SEGMENTS WILL BE RETURNED TO BLANK STATUS
23 * D3 THE ENT FOR THE LU WILL BE REINSTATED
24 * D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
25 * D3 WILL BE UPDATED FOR THE TERMINATION
1 BEGIN XCON
  2 * SIGN ON A USER TO FDS
  3  DO UNTIL VALID USER ID (P3)
  4    IF ID NOT A - Z THEN
  5      WRITE 'XCOG ENTER VALID ID (A - Z)'
  6    ENDIF
  7  ENDU
  8  IF USER ID IS BEING USED, THEN
  9    WRITE "***XCOG LU 'LU' IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED"
 10  EXIT :XCETA
 11  ENDIF
 12  IF FDS RESOURCE NOT DEFINED, THEN
 13    CALL RMAQ (GLOBAL ALLOCATE, LOCAL SET)
 14  ELSE
 15    CALL RMAG (LOCAL SET)
 16  ENDIF
 17  IF NUMBER SIGNED ON(STRAC) .EQ. MAXIMUM USERS(STRMN), THEN
 18    ISSUE MESSAGE "***XCOG FDS CURRENTLY AT MAX USER'S."
 19  ELSE
 20    DO FOR STRMN(NUMBER OF FDS ENTRIES)
 21      IF ENTRY'S LUCSTBLU .EQ. REQUESTING LU(P1) THEN
 22        ISSUE MESSAGE "***XCOG 'LU' IS ALREADY SIGNED ON TO FDS"
 23        EXIT :XCETA
 24      ELSE
 25        IF THIS ENTRY IS AVAILABLE, THEN
 26          SET AS CURRENT-ENTRY-ADDRESS
 27          ENDIF
 28    ENDIF
 29  ENDU
 30  BECOME PRIVILEGED & DISABLED
 31  CALL SLDR
 32  START SEARCH WHILE NUMBER-FOUND <.T. NUMBER-NEEDED
 33  SEARCH ID-SEGMENTS USING KEYS(B(1657)
 34  IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
 35    SET ID ADDRESS OF XEXEC
 36    INCREMENT NUMBER-FOUND
 37  ELSE
 38    IF XEXEC NOT FOUND AND THIS ID .EQ. XUSER, THEN
 39      SET ID ADDRESS OF XUSER
 40      INCREMENT NUMBER-FOUND
 41  ELSE
 42    IF XATM NOT FOUND AND THIS ID .EQ. XATM, THEN
 43      SET ID ADDRESS OF XATM
 44      INCREMENT NUMBER-FOUND
 45  ELSE
 46    IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
 47      SET ID ADDRESS OF FIRST-BLANK
 48      INCREMENT NUMBER-FOUND
 49  ELSE
 50    IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
 51      SET ID ADDRESS OF SECOND-BLANK
 52      INCREMENT COUNT
 53  ENDIF
 54 ENDIF
 55 WRITE ENDIF
 56 ENDIF
 57 WRITE ENDIF
 58 EXIT IF THERE ARE NO MORE IDS
CALL BLIX   ENABLE
ISSUE MESSAGES "**XCO? CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP

BUILD FTRY IN XSTU
SET LU INQ STBLU
SET LU IN ASCII INTO STBLA
SET USER'S ID INTO STBD
SET ADDRESS OF FIRST-BLANK INTO STBM
SET ADDRESS OF SECOND-BLANK INTO STBE
INCREMENT ACTIVE COUNT(STBAC)
BUILD XMNAM & XEXWH
MOVE PRIORITY THRU DISC ADDRESS FROM XMG to FIRST-BLANK
TURN ON TH BIT
SET NAME TO XPNAM
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
TURN ON TH BIT
SET NAME TO XEXWH
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE ERT OVERLAY
SAVE ERT VALUE IN STER
SET ID ADDRESS OF XATHN INTO ERT
ENDIF
SET INPUT PARTS INTO ID OF XMG
SCHEDULE XMG via BLIST
CALL BLIST
CALL BLIX  ENABLE
IF FDS HAS A FATHER, THEN
CALL MESS 'OFF,FATHER'
ENDIF
ISSUE MESSAGE "***XCO1 LU 'NN' SIGNED ON TO FDS"
ENDSEARCH

:XCETA
CALL RMNG (LOCAL CLEAR)
ENDIF
END IF
CALL EXEC TERMINATE
END XCON
Reproducibility of the original page is poor.
**FORTRAN CALLING PROCEDURE**

1. **CALL XELDS (XDCLD)**

**XDCLD** determines which of the following directives was requested and calls the appropriate handler subroutine.

**INPUT**

- **XCOMMON - MASTPA (RITS 10-13 CONTAIN A 0 INDEX INTO A LIST OF DIRECTIVES)**

**INTERNAL VARIABLES**

- **LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES**

**NOTES**

- **USES .ENTR, XDLST, XERMT**

**XDCLD is designed to be the main routine for the overlay segment**

**BEGIN XDCLD**

1. **EXTRACT DIRECTIVE INDEX FROM MASTPA**
2. **CASE (.LIST:) INDEX**
3. **LIST: CALL XDLST**
4. **ENDCASE**
5. **CALL XERMT TO RETURN FROM SEGMENT**

END XDCLD
92 1 *0  FORTRAN CALLING PROCEDURE
93 1 *0 CALL XDCLS (XDCLU)
94 1 *0
95 1 *0 ********
96 1 *0 XDCLU DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
97 1 *0 REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
98 1 *0 STORE
99 1 *0 RESTORE
100 1 *0 UNLOAD
101 1 *0 LOAD
102 1 *0 BATCH
103 1 *0
104 1 *0 ********
105 1 *0 INPUT
106 1 *02 XE COMMON - MASSTA (BITS 10-13 CONTAIN A 9-13 INDEX INTO A LIST OF
107 1 *02 DIRECTIVES)
108 1 *0
109 1 *0 ********
110 1 *04 INTERNAL VARIABLES
111 1 *04 LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
112 1 *04
113 1 *04 ********
114 1 *05 NOTES
115 1 *05 USES .ENTR, XDSTO, XDRES, XDUNL, XDLOA, XDBAT, XERTN
116 1 *05
117 1 *05 XDCLU IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
118 1 *05 CONTAINING THE REFERENCED DIRECTIVES
119 1 *05
120 1 *05 ********
121 1 *
122 1 *
123 1 *
124 1 *
125 1 *
126 1 BEGIN XDCLU
127 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7
128 2 CASE (:STOR:, :REST:, :UNLO:, :LOAD:, :BATCH:) INDEX
129 3 :STOR: CALL XDSTO
130 3 :REST: CALL XDRES
131 3 :UNLO: CALL XDUNL
132 3 :LOAD: CALL XDLOA
133 3 :BATCH: CALL XDBAT
134 2 END CASE
135 2 CALL XERTN TO RETURN FROM SEGMENT
136 1 END XDCLU
FORTRAN CALLING PROCEDURE

CALL XDCLE

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

INPUT

X COMMON - CARTAG, FLAG, LU
MANAGER - AWA HEADER AND TOC (SEE XMANA)

OUTPUT

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

LOCAL VARIABLES

AVA - AWA HEADER AND TOC RECEIVED FROM MANAGER
HD1 - TOC CHAIN 1 HEAD
HD3 - TOC CHAIN 3 HEAD
HD6 - TOC CHAIN 6 HEAD
NEXT - INDEX TO NEXT ALLOCATABLE EIGHT WORD ENTRY IN QUEUE
NMAX - MAXIMUM SIZE OF QUEUE
ORG - BASE REFERENCE ADDRESS FOR AWA ADDRESSES
QUEUE - BUFFER FOR CONSTRUCTING AWA RESTORE REQUESTS FOR NON-DELETED ITEMS
TOCMAX - MAXIMUM SIZE OF TOC ACCOMODABLE BY XDCLE

NOTES

USES EXEC, IAND, PTRN, PURGE, XREQ, XR16, XMOV, XMGS, XRD6, XRFN, XUDRG, XVABN

IN THE EVENT THAT A COMPACTED AWA TOC WILL NOT FIT IN THE ALLOCATED BUFFER SPACE AND ELEMENTS OF CHAIN 1 OR 8 EXTEND BEYOND IT, THE CLEAR FUNCTION WILL NOT BE PERFORMED.

SHOULD THE RESTORATION OF CHAIN 1 OR 8 FAIL AFTER THE CLEAR REQUEST HAS BEEN COMPLETED BY THE MANAGER, FBS WILL BE TERMINATED.

IN ORDER TO PROTECT USER DISK FILES.

C C********
240 1 C**********
241 1 C00  FORTRAN CALLING PROCEDURE
242 1 C00  CALL XDCOP
243 1 C00  C**********
244 1 C01  XDCOP PROCESSES A USER REQUEST TO COPY A SEQUENCE TABLE.
245 1 C01  INTERFACE TABLE, DATA ELEMENT, DDE, UTOD, OR FOR. ONLY
246 1 C01  THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S AWN CAN BE COPIED.
247 1 C01  C**********
248 1 C02  INPUT
249 1 C02  COMMON XE - CARTAG, COMBUF, QAL, REEPTR, TOKENS
250 1 C02  C**********
251 1 C03  OUTPUT
252 1 C03  COMMON XE - COMPR, REBUF
253 1 C03  C**********
254 1 C04  INTERNAL VARIABLES
255 1 C04  COMMON XS - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE)
256 1 C04  (2) DTYP: TYPE CODE FOR DATA BASE FILES
257 1 C04  (3) J: INDEX
258 1 C04  (4) IDC: EXEC BUFFER
259 1 C04  (148) IERR: FILE MANAGER ERROR RETURN
260 1 C04  (149) IMSG: ERROR MESSAGE NUMBERS
261 1 C04  (150) HFNAME: NEW FMGR FILE NAME
262 1 C04  (153) MNPTR: POINTER TO OLD NAME IN COMBUF
263 1 C04  (154) OFNAME: OLD FMGR FILE NAME
264 1 C04  (157) MNPTR: POINTER TO OLD NAME IN COMBUF
265 1 C04  (158) TOSCE: TOC ENTRY FOR DATA BASE RENAME
266 1 C04  (160) EOF: EOF RETURN FROM FMGR
267 1 C04  COMMON XB - (204) IUB: USER BUFFER FOR EXEC I/O (1024)
268 1 C04  (1244) FMTYPE: FMGR FILE TYPE
269 1 C04  (1245) PARM1: FIRST BYTE OF REEPTR
270 1 C04  (1246) PARM2: SECOND BYTE OF REEPTR
271 1 C04  (1247) PREFIX: PREFIX FOR FILE NAME
272 1 C04  (1248) SCRITY: FILE SECURITY CODE (0 OR 88)
273 1 C04  (1249) SIZE: FILE SIZE IN BLOCKS
274 1 C04  (1251) IDC: EXEC BUFFER
275 1 C04  C**********
276 1 C05  ROUTINES USED -
277 1 C05  C05  CLOSE, CREAT, EXEC, OPEN, PURGE, READF, WRITF, XDOBA, XDOBD,
278 1 C05  XREQ, XROV, XRSID, XRSID, XRSFE
279 1 C05  C**********
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
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
EXIT XDCOP

:SYNTAX: CALL XRMSC "SYNTAX ERROR ..." AND EXIT
:CLASER: CALL XRMSC "INVALID CLASS DESIGNATOR ..." AND EXIT
:NAMERR: CALL XRMSC "NEW NAME IS INVALID ..." AND EXIT
:MAXERR: CALL XRMSC "AUTHORIZED LIMIT ..." AND EXIT
:INVALS: CALL XRMSC "NOB CANNOT BE ..." AND EXIT
:FILEERR: CALL XRMSC "FILE ACCESS ERROR ... ON ......" AND EXIT TO :END:
:TOCERR: CALL XRMSC TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
:TPER: CALL XRMSC "INCONSISTENT FILE TYPE ..."

:END:
IF POB HAS BEEN LOGGED IN XPOB, THEN
CALL XPOB TO DELETE POB FROM XPOB
ENDIF
IF A NEW FILE HAS BEEN BUILT, THEN
PURGE NEW FILE
CLOSE OLD FILE
ENDIF
END XDCOP
FORTRAN CALLING PROCEDURES

CALL XDOBB(NAME, ERR, ISIZE)
CALL XDOB(NAME, ERR)
********
XDOBB, XDOBY, AND XDOBO PROVIDE THE ABILITY TO ADD AN MDB OR
PDB TO MDB/PDB DIRECTORY (XPDB), TO VERIFY THAT AN MDB OR PDB IS
LOGGED IN XPDB AND RETRIEVE THE SIZE OF THAT MDB/PDB, AND TO
DELETE AN MDB OR PDB FROM XPDB, RESPECTIVELY
********
INPUT
NAME - 2 WORD INTEGER ARRAY CONTAINING THE UNQUALIFIED ASCII
MDB/PDB NAME
ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
XDOBB ONLY)
********
OUTPUT
IERR - INTEGER WORD CONTAINING RETURN CODE
-1 FOR THE ERRO CODE RETURNED FOR XPDB ACCESS
0 NO ERRORS
1 DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE
2 PDB/PDB LIMIT EXCEEDED
ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
XDOBY ONLY)
********
COMMON
XE - RUAL
CARTG
XS - DCO (WORDS 1-144)
********
NOTES
XRLK, XRLCK, .ENDR, OPEN, CLOSE, READF, AND WRITF ARE USED
WHEN WORKING WITH MDB'S, QUAL SHOULD BE SET TO 77 OCTAL
445 1 BEGIN XD8A
446 2 2 STORE RETURN ADDRESS
447 3 BEGIN XD8A
448 4 CALL :ENTW TO SET UP CALLING ARGUMENTS
449 5 CALL X8L6K FOR EXCLUSIVE USE OF XD8B
450 6 EXIT TO :FILEERR: IF ERROR RETURNED
451 7 COMPUTE RECORD NUMBER FOR USER'S DIRECTORY (QUAL-77B)/2+1
452 8 CALL READS FOR RECORD COMPUTED
453 9 EXIT TO :FILEERR: IF ERROR RETURNED
454 0 DETERMINE PART OF RECORD TO BE USED
455 1 IF REQUEST IS FOR ADD, THEN
456 2 EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
457 3 START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
458 4 EXIT TO :NAMERR: IF ENTRY NAME MATCHES PARAMETER NAME
459 5 ENDLOOP
460 6 ENDSEARCH
461 7 STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
462 8 INCREMENT # OF CURRENT ENTRIES
463 9 ELSE
464 0 START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
465 1 EXIT IF ENTRY NAME MATCHES PARAMETER NAME
466 2 ENDLOOP
467 3 EXIT TO :NAMERR:
468 4 ENDSEARCH
469 5 IF REQUEST IS FOR VERIFY, THEN
470 6 STORE WORD 3 OF ENTRY IN ISIZE
471 7 ELSE
472 8 REPLACE ENTRY WITH LAST ENTRY
473 9 STORE ZEROS IN LAST ENTRY
474 0 DECURRENT # OF CURRENT ENTRIES
475 1 ENDIF
476 2 ENDIF
477 3 CALL WRIT TO WRITE RECORD TO XD8B
478 4 EXIT TO :RETURN:
479 5 2 :NAMERR: SET IERR = 1 AND EXIT TO :RETURN:
480 6 2 :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
481 7 :FILEERR: SET IERR = FNGA ERROR CODE
482 8 :RETURN: CALL CLOSE FOR XD8B
483 9 CALL X8L6K TO RETURN RESOURCE #
484 0 END XD8A
**FORTRAN CALLING PROCEDURE**

**XDELETE**

XDELETE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE AMA.

XDELETE IF THE ELEMENT IS A DATA BASE, THE ASSOCIATED FILE MANAGER FILE IS PURGED AND FOR A PDB THE PDB DIRECTORY IS UPDATED.

**INPUT**

COMMON XE - COMBUF, COMPR, LU, QUAL, TOKENS

**OUTPUT**

COMMON XE - REGBUF

**NOTES**

Routines Used

EXEC

IAND

PURGE

XDBB

XERB

XEXIT

XRG

XRMD

XRMS

XRPK

XRPK

XRUPK

XUDBG

**C**
1 BEGIN XDELE
2 DO WHILE END-OF-STATEMENT NOT REACHED
3 PROCESSING EACH ELEMENT SPECIFIED
4 ERASE IF COMMA IS NOT NEXT LEXICAL 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 (0)
9 ENDIF
10 IF DATA BASE TO BE DELETED, THEN
11 BUILD AND ISSUE AWS 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 APROD TO DELETE THIS PROD FROM XPDB
16 IF ERROR IS RETURNED, THEN
17 CALL XHMSG - "FILE ACCESS ERROR 0... XPDB"
18 SET ERROR FLAG
19 ENDIF
20 BUILD AND ISSUE AWS MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
21 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
22 CALL XHMSG - "XXXXX NOT FOUND"
23 SET ERROR FLAG
24 ENDIF
25 IF CLASS IS DATA BASE (C), OR
26 CLASS IS DDBE (F), THEN
27 IF ERROR FLAG IS NOT SET, THEN
28 CALL IDT-Bold TO CONSTRUCT FILE NAME
29 ISSUE RTE PURGE FOR THE FILE
30 IF RETURN CODE FROM PURGE, THEN
31 CALL XHMSG - "FILER ERROR XXX XXXXXX"
32 ENDIF
33 ENDIF
34 ELSE
35 CALL XHMSG - "... IS A MDO. NOT DELETED."
36 ENDIF
37 ENDDO
38 1 EXIT TO :RETURN:
39 2 :ERROR: CALL XHMSG - "SYNTAX ERROR"
40 :RETURN:
41 1 END XDELE
CALL XNLIST (TOCST)

XNLIST PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECODING EACH ELEMENT AND FLAGGING IT IN TOCST AS AN ELEMENT
TO BE STORED/RESTORED.

INPUTS IN CALLING SEQUENCE:
TOCST - (INTEGER, 1200 WORDS) ADDRESS OF FIRST TOC ENTRY
OF A LIST OF POSSIBLE ELEMENTS' TOC ENTERIES

INPUTS IN COMMON:
XB(151) ABLGL, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:
TOCST - (INTEGER, 1200 WORDS) LIST OF TOC ENTRIES; THOSE
THAT ARE TO BE STORED/RESTORE ARE FLAGGED.

OUTPUTS IN COMMON:
XB(151) ABLGL, XB(157) TOTSTZ, XB(158) TRWD

INTERNAL XB COMMON USED:

XB(151) ABLGL - (INTEGER, 1 WORD) ABORT FLAG
XB(152) ERFLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TOTSTZ - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
BE STORED/RESTORED
XB(158) TRWD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
BE STORED/RESTORED
XB(198) RETC - (INTEGER, 1 WORD) XCOM RETURN CODE
XB(199) BATCLS - (INTEGER, 1 WORD) CLASS OF DATA BEING SEARCHED
FOR IN LIST OF TOC ENTRIES
XB(200) NMPTER - (INTEGER, 1 WORD) POINTER TO NAME IN COMBUF
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCST

COMMON USED:

EQUIVALENCE
+ (REX(5), RASITA), (REX(8), EOS )
+ (REX(95), NAME ), (REX(96), XTPN )
+ (REX(113), COMMA ), (REX(144), COMPR )
+ (REX(145), COMBUF)
647 1 BEGIN XDLIS
648 2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
649 3 TURN ERFLG OFF
650 3 :RTM1:
651 3 DO UNTIL E01 IS SENSED IN COMM
652 4 ERREXIT IF TOKEN IS NOT "NAME" TO :ERR1:
653 4 SAVE INDEX TO NAME FIELD
654 4 INCREMENT TO NEXT TOKEN
655 4 IF TOKEN IS A RHYPHEN THEN
656 5 ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:
657 5 INCREMENT TO NEXT TOKEN
658 5 DECODE CLASS NAME (I, S, D, F)
659 5 ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:
660 5 SET CLASS TO CLASS SPECIFIED
661 4 ELSE
662 5 SET CLASS TO DATA ELEMENT
663 4 ENDF
664 4 IF XDLIS CALLED FROM STORE THEN
665 5 ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:
666 4 ENDF
667 4 ERREXIT IF NAME/CLASS ENTRY NOT FOUND IN TOC TO :ERR2:
668 4 CALL ARSET TO TURN STORE/RESTORE BIT ON
669 4 INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
670 3 ENDDO
671 3 :RTN2:
672 3 IF ERROR FLAG IS ON THEN
673 4 CALL XCOM TO REPROMPT USER TO CONTINUE
674 4 ERREXIT IF RESPONSE IS X TO :ERR3:
675 4 ENDF
676 2 ENDDO
677 1 EXIT XDLIS
678 2 :ERR1:
679 2 SET ERROR FLAG ON
680 2 CALL XMSSG TO DISPLAY SYNTAX ERROR
681 2 GO TO :RTM2:
682 2 :ERR2:
683 2 IF ERROR FLAG IS OFF THEN
684 3 TURN ERROR FLAG ON
685 3 CALL XMSSG TO DISPLAY NOT STORED/RESTORED MESSAGE
686 2 ENDF
687 2 CALL EXEC TO DISPLAY ELEMENT NAME
688 2 GO TO :RTM1:
689 2 :ERR3:
690 2 SET ABFLG TO ABORT STORE/RESTORE OPERATION
691 1 END XDLIS
BEGIN XDLST

IF DEVICE ID FIELD SPECIFIED, THEN

SET LU FOR LISTING AS INDICATED ON INCREMENTAL I/O
ELSE

SET LU FOR LISTING TO BE TERMINAL LU
ENDIF

DO UNTIL END-OF-STATEMENT IS REACHED

ERRER IF COMMA IS NOT SPECIFIED :ERR09:
ERRER IF A NAME DOES NOT FOLLOW THE COMMA :ERR09:
ERRER IF A CLASS DESIGNATOR IS SPECIFIED, THEN
SET CLASS (1, S, OR D) FOR XEREN CALL
ELSE
USE DATA (O) CLASS IN XEREN CALL
ENDIF

CALL XEREN TO RETRIEVE THIS TABLE OR DATA ELEMENT
ERRER IF NOT FOUND :ERR01:
ERRER IF AWA SPACE NOT AVAILABLE FOR TABLE IN DATA :ERR11:
CALL EXEC TO PERFORM CLASS READ OF DATA OR TABLE INTO
BOTTOM OF WORKING BUFFER
IF INTERFACE TABLE TO BE LISTED, THEN
CALL XMOV TO MOVE CHARACTERISTICS TO TOP OF WORKING BUFFER
READ SHORT PROMPTS FOR THIS PROCESSOR INTO WORKING BUFFER
CALL XEXIT TO INITIALIZE LITERAL ENTRIES
INITIALIZE INTERFACE TABLE EDITOR COMMON TO USE ITS LIST RTN.
INITIALIZE 'LSTFLG' TO INDICATE ENTIRE TABLE TO BE LISTED
CALL XILST TO LIST THE INTERFACE TABLE
ELSE
IF SEQUENCE TABLE TO BE LISTED, THEN
CALL XMOV TO MOVE TABLE TO TOP OF WORKING BUFFER
CALL XILST TO LIST SEQUENCE TABLE
ELSE
INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE OF ITS LIST RTN.
SET 'LSTFLG' TO INDICATE ONLY 1 DATA ELEMENT BEING LISTED
INITIALIZE PRINT BUFFER WITH NAME OF ELEMENT
CALL XILSTD TO LIST THE DATA
ENDIF
ENDIF
INCREMENT TO NEXT TOKEN IN OPERAND LIST OF THE DIRECTIVE IMAGE

:ERR01: CALL XERNSG -- '... NOT FOUND'
:ERR11: CALL XERNSG -- '... CANNOT BE MOVE FROM DATA TO AWA'
ENDIF
ENDDO
EXIT TO :RETURN:
:ERR09: CALL XERNSG -- 'SYNTAX ERROR'
:RETURN:
END XDLST
783 1 CD**********
784 1 CD0     FORTRAN CALLING PROCEEDURE
785 1 CD0     CALL XDOFF
786 1 CD0     ********
787 1 CD1     XDOFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
788 1 CD1     DELETES ALL DRE AND UTDB FILES LOGGED IN THE ANA,
789 1 CD1     PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
790 1 CD1     RETURNS NORMAL PARM TO THE FDS MANAGER AND TERMINATES
791 1 CD1     NORMALY VIA RTE.
792 1 CD1     ********
793 1 CD2     INPUT
794 1 CD2     ********
795 1 CD2     COMMON XE = LU, FLAGS, QUAL, RERBUF
796 1 CD2     ********
797 1 CD2     COMMON XB = ORG = ORIGIN ADDRESS OF ANA, USED TO CALCULATE
798 1 CD2     INDICES INTO 'ANA' FROM ADDRESS POINTERS
799 1 CD2     OF TOC ENTRIES
800 1 CD2     ********
801 1 CD2     ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND
802 1 CD2     TOC RETRIEVED VIA XREQ
803 1 CD2     ********
804 1 CD2     INTERNAL VARIABLES
805 1 CD2     ********
806 1 CD2     COMMON XS = POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC
807 1 CD2     ENTRY, MOST SIGNIFICANT (BIT 15) BIT
808 1 CD2     SET TO INDICATE END-OF-CHAIN
809 1 CD2     ********
810 1 CD2     INDEX = VALUE COMPUTED FROM POINTER TO BE
811 1 CD2     FORTRAN INDEX INTO 'ANA' FOR NEXT TOC
812 1 CD2     ENTRY
813 1 CD2     ********
814 1 CD2     NOTES
815 1 CD2     ROUTINE USED = EXEC, PURGE, XDSTA, XPXIT, XREQ, XREX, XRI6,
816 1 CD2     XRMV, XRMSG, XRRFM, XRRCX, XRSET, XRUPK, XRCOM,
817 1 CD2     XUD6, XVABM
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL XFRN TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR ORDE FILES IS NOT NEGATIVE THEN
6 DO UNTIL ORDE CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
8 OUTPUT 'X113 TOC TOO LARGE, PURGE INCOMPLETE'
9 EXIT DO
10 ENDF
11 CALL XFRN TO CREATE FILE NAME 'XXX'
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 'X113 TOC TOO LARGE, PURGE INCOMPLETE'
20 EXIT DO
21 ENDF
22 IF TYPE OF DATA BASE IS UDB, THEN
23 CALL XFRN TO CREATE FILE NAME 'XXX'
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 XSDTA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XDBG
32 ENDF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVBN - 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 XDRDE (DATBUF, DBDCB)**

### XDRDE HANDLES THE RESTORING OF DATA ELEMENTS, SEQUENCE TABLES AND INTERFACE TABLES

### INPUTS FROM CALLING SEQUENCE:

- **DATBUF** - (INTEGER, 1480 WORDS) BUFFER USED TO READ IN RECORDS OF DATA FROM DATA BASE FILE
- **DBDCB** - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DCB

### INPUTS FROM XB COMMON:

- **XB(154)**: ADFLE, **XB(155)**: EBRLG, **XB(157)**: TOTSIZ
- **XB(201)**: NOTOC

### OUTPUTS IN CALLING SEQUENCE:

- **DATBUF, DBDCB**

### OUTPUTS IN XB COMMON:

- **XB(151)**: ADFLE, **XB(152)**: EBRLG, **XB(153)**: TOTSIZ,
- **XB(155)**: TOTSIZ, **XB(170)**: EMBLK, **XB(171)**: TOCNX,
- **XB(201)**: NOTOC

### INTERNAL XB COMMON USED:

- **XB(151)**: ADFLE - ABORT FLAG
- **XB(152)**: EBRLG - ERROR MESSAGE FLAG
- **XB(153)**: ADFLE - RESTORE ALL UTD FILE
- **XB(154)**: DEBUG - DEBUG FLAG
- **XB(155)**: FNAME - DB FILE NAME
- **XB(156)**: DATBLK- BLOCK # WHERE DATA ITEM BEGINS
- **XB(157)**: DATMXX - WORD INDEX INTO DATBUF WHERE DATA BEGINS
- **XB(158)**: EERR - ERROR FLAG FOR FMGR CALLS
- **XB(159)**: TOCNX- # WORDS OF TIC IN TCBUF (NOTOC *8) + 8
- **XB(160)**: NOBLS= - # BLOCKS TO READ
- **XB(161)**: LH - # WORDS OF DATA TO MOVE
- **XB(162)**: DATEND- BLOCK # WHERE DATA ITEM ENDS
- **XB(163)**: TOPLBK- BLOCK # OF FIRST BLOCK IN DATBUF
- **XB(164)**: EMBLK- BLOCK # OF LAST BLOCK READ
- **XB(165)**: TOCNXX- INDEX INTO TOC OF CURRENT DATA ITEM
- **XB(166)**: IL - # WORDS OF DATA TO READ
- **XB(167)**: TOCENT- DATA BASE TOC ENTRY
- **XB(201)**: NOTOC - # TOC ENTRIES IN TCBUF
- **XB(201)**: TCBUF- 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-DRDE 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 XRMOV 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 ENDF
18 BUILD AWA REQUEST TO ALLOCATE AND STORE DATA
19 CALL EXEC TO WRITE DATA TO SAN
20 ERREXIT IF ERROR FROM EXEC TO AWA 2:
21 IF AWA REQUEST BUFFER IS FULL THEN
22 CALL XRDEQ TO ISSUE REQUEST
23 EXIT XDRDE IF ERROR IN XRDEQ
24 ENDF
25 ENDO
26 EXIT XDRDE
27 :ERR1:
28 CALL XRSC TO CONVERT ERROR CODE TO ASCII
29 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
30 GO TO :ERR3:
31 :ERR2:
32 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
33 :ERR3:
34 SET ABFLG TO SAY ABORT RESTORE
35 END XDRDE
FORTRAN CALLING SEQUENCE:

CALL XRDF (DATBUF, DBDCB)

XRDF HANDLES THE RESTORING OF ORDE FILES FROM A MDB/UTDB FILE

INPUTS IN CALLING SEQUENCE:

DATBUF - (INTEGER, 1400 WORDS) BUFFER USED TO READ IN
RECORDS OF ORDE FILES FROM DATA BASE FILE

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

INPUTS IN XR COMMON:

XB(152) ERFILE, XB(154) ALLFLG, XB(165) TOCEND,
XB(169) TOPLFLG, XB(170) ENDORL, XB(171) TOCNOX,
XB(182) TOTSIZ

OUTPUTS IN CALLING SEQUENCE:

DBDCB

OUTPUTS IN XR COMMON:

XB(151) ABFLG

INTERNAL XR COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) ERFLG - ERROR MESSAGE FLAG
XB(154) ALLFLG - RESTORE ALL UTDB FLAG
XB(155) DEBUG - DEBUG FLAG
XB(159) FLDNM - DB FILE NAME
XB(162) DATBLK - BLOCK # WHERE ORDE BEGINS
XB(163) DATNOX - WORD INDEX INTO DATBUF WHERE ORDE BEGINS
XB(164) ERR - ERROR FLAG FOR FNGR CALLS
XB(165) TOCEND - # WORDS OF TOC IN TOCBUF (MOTOC #8) + 8
XB(166) NOBLSK - # BLOCKS OF UTDB/ORDE TO READ/WRITE
XB(167) LEN - # WORDS OF DATA TO MOVE
XB(169) TBPBLK - BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBLK - BLOCK # OF LAST BLOCK READ
XB(171) TOCNOX - INDEX TO CURRENT TOC ENTRY
XB(172) DRDFIL - DRDE FILE NAME
XB(175) FLDL - # WORDS OF DATA TO READ/WRITE
XB(201) TOCBUF - TOC BUFFER

COMMON USED:
BEGIN XDRDF
  DO WHILE THERE ARE T Obl ENTRIES TO PROCESS
    IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
      CALL XDRFN TO CREATE THE DRDE FILE NAME
      IF THERE WAS A CREATE ERROR THEN
        IF ERROR FLAG IS OFF THEN
          SET ERROR FLAG ON
          CALL XMSG TO DISPLAY MAIN MESSAGE
        ELSE
          CALL XDRF TO DISPLAY ERROR MESSAGE AND ERROR CODE
        ENDIF
      ELSE
        CALL CLOSE TO CLOSE FILE
        ERREXIT IF CLOSE ERROR TO :ERR1:
        CALL OPEN TO OPEN DRDE FILE AS TYPE 1
        ERREXIT IF OPEN ERROR TO :ERR1:
        DO UNTIL ALL BLOCKS OF DRDE HAVE BEEN PROCESSED
          IF DATA FOR DRDE IS IN DATBUFF THEN
            CALL WRITF DATA TO DRDE
            ERREXIT IF WRITF ERROR TO :ERR1:
          ELSE
            CALL READF TO READ NEXT BUFFER OF DATA
            ERREXIT IF READF ERROR TO :ERR2:
          ENDIF
          SET INDICES INDICATING DATA IN BUFFER
        ENDIF
      ENDIF
      CALL CLOSE TO CLOSE DRDE FILE
      ERREXIT IF CLOSE FAILED TO :ERR1:
      BUILD DMA REQUEST TO ALLOCATE DRDE IN DMA
      IF DMA REQUEST BUFFER IS FULL THEN
        CALL XDRF TO MAKE REQUEST
      EXIT XDRDF IF XDRF ERROR
      ENDIF
    ELSE
    ENDIF
  ENDDO
  EXIT XDRDF
  :ERR1:
  CALL PURGE TO PURGE DRDE FILE
  :ERR2:
  CALL XMSG TO DISPLAY ERROR MESSAGE WITH FILE NAME
  SET ABLFLG TO Say ABORT RESTORE
  END XDRDF
**FORTRAN CALLING PROCEDURE**

**CALL XREC**

**XREC PROCESSES THE RECALL DIRECTIVE. A UTOB IS CREATED AND THE CONTENTS OF THE SPECIFIED PPG ARE COPIED TO IT.**

**INPUT**

**COMMON XE - CARTAG, COMBUF, COMPTR, FLAGS, LV, TOKENS**

**FILES**

**- XXXX (PPG FILE SPECIFIED)**

**OUTPUT**

**COMMON XE - REGBUF, REOPTR**

**FILES**

**- XXXX (UTOB FILE SPECIFIED)**

**INTERNAL VARIABLES**

**DCBPDD - DCB FOR THE PPG FILE; ALLOCATED IN XB COMMON;**

**CONTAINS 1152 WORD BUFFER USED TO READ THE PPG AND TO WRITE THE UTOB**

**DCBUTO - DCB FOR THE UTOB FILE; ALLOCATED IS XS COMMON**
BEGIN XDEC

SET STATUS FLAG INDICATING PRO & UDB FILES NOT OPEN

ENDIF

Go to (0:

END OF XDEC
1200 2 :ERROR7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERROR9: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERROR16: ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERROR18: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ...
1204 2 :ERROR19: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERROR21: ISSUE MESSAGE - "AWA OVERFLOW - XXXX NOT LOGGED"
1206 2 :ERROR22: ISSUE MESSAGE - "XXXX ALREADY EXISTS"
1207 2 :ERROR44: ISSUE MESSAGE - "FILE ACCESS ERROR #-- XPDDB"
1208 2 :RETURN:
1209 2 IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
1210 3 PURGE UTDB
1211 2 ENDIF
1212 2 IF STATUS FLAG INDICATES UTDB IS LOGGED IN AW/A, THEN
1213 3 CALL XER2 TO DELETE UTDB FROM AW/A
1214 2 ENDIF
1215 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1216 3 CALL CLOSE FOR PDB FILE
1217 2 ENDIF
1218 1 "ND XDREC"
**FORTRAN CALLING PROCEDURE**

CALL XDREN

XDREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE, INTERFACE TABLE, DATA ELEMENT, OR DE, UUID, OR PDB. ONLY THOSE TABLES OR ELEMENTS LOGGED IN THE USERS' AWX ARE RENAMED.

**COMMON XE - CARTRG, COMBUF, COMTRA, QUAL, RERPTR, TOKENS**

**COMMON XS - (2) DTYPE: TYPE CODE FOR DATA BASE FILES**

INDEX

FILE MANAGER ERROR RETURN

ERROR MESSAGE NUMBERS

NEW FGKR FILE NAME

NEW NAME IN COMBUF

POINT TO NEW NAME IN COMBUF

FILE NAME

TO OLD NAME IN COMBUF

TOC ENTRY FOR DATA BASE RENAME

DATA CLASS CODE (STORED IN LEFT BYTE)

**COMMON XB - (201) IOC8: EXEC BUFFER**

**EXTERNAL REFERENCES**

**Routines Used**

**EXEC**

**IAND**

**KVVT**

**MAAP**

**XDBBX**

**XDOBD**

**XREG**

**XRMOV**

**XRMSC**

**XRSTR**
1 BEGIN XDREN
2  EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
3  SAVE POINTER TO CURRENT NAME
4  INCREMENT TO NEXT TOKEN
5  IF TOKEN IS A HYPHEN, THEN
6  EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
7  DECODE CLASS NAME
8  EXIT TO :CLASS: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,B)
9  ELSE
10  SET CLASS TO BE
11  ENDIF
12
13  EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
14  SAVE POINTER TO NEW NAME
15  IF CLASS IS DATA BASE OR ORDE, THEN
16  EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS
17  ENDIF
18  IF CLASS IS DATA BASE, THEN
19  EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
20  ENDIF
21  IF CLASS IS INTERFACE TABLE, THEN
22  CALL EXEC TO WRITE/READ NEW NAME
23  BUILD AHA MANAGER REQUEST TO CHANGE NAME IN TOC
24  BUILD AHA MANAGER REQUEST TO STORE NEW NAME IN TABLE
25  CALL XREN TO ISSUE REQUESTS
26  EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR
27  ELSE
28  IF CLASS IS DATA BASE, THEN
29  BUILD AND ISSUE AHA MANAGER REQUEST FOR TOC ENTRY
30  EXIT TO :NAMERR: IF RETURN INDICATES ERROR
31  EXIT TO :INVALID: IF DATA BASE IS AN NDB
32  ENDIF
33  BUILD AND ISSUE AHA MANAGER REQUEST FOR NAME CHANGE
34  EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR
35  IF CLASS IS DATA BASE OR ORDE, THEN
36  CALL XRDFN TO FORMAT FILE NAME
37  CALL FILE MANAGER TO CHANGE DISC FILE NAME
38  EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR
39  IF FILE IS A PDB, THEN
40  CALL XDDDD TO DELETE OLD PDB FROM XPD
41  EXIT TO :NAMAGN: IF ERROR RETURNED
42  CALL XDDDA TO ADD NEW PDB NAME TO XPD
43  EXIT TO :TELUSER: IF ERROR RETURNED
44  ENDIF
45
46  ENDIF
47
48  ENDIF
49
50  EXIT XDREN
51  EXIT XDREN
52
53  :SYNTAX: CALL XRMSG TO DISPLAY SYNTAX ERROR AND EXIT
54  :TOOLNG: CALL XRMSG ("NEW NAME IS TOO LONG") AND EXIT
55  :CALGET:
56  CALL EXEC TO GET BUFFER CONTAINING NEW TABLE NAME
57  :NAMERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT
58  :CLASS: CALL XRMSG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT
59  :INVALID: CALL XRMSG ("AN NDB CANNOT BE RENAMED") AND EXIT
**FORTRAN CALLING PROCEDURE**

**CALL XDREQ**

**C**

**INPUT**

**COMMON XE - LU, REGPTR, REGBUF**

**COMMON XB - DEBUG, ERFGL**

**COMMON XS - DDRDCB**

**OUTPUT**

**COMMON XB - ABFLG**

**NOTES**

**EXTERNAL REFERENCES**

**EXEC**

**IAND**

**PURGE**

**XREQ**

**XREXX**

**XRIG**

**XRMOV**

**XRMSG**
1 BEGIN XREQ
2 CALL XREQ TO PROCESS AWA REQUEST(S)
3 IF AN ERROR RETURNED BY AWA MANAGER, THEN
4 IF ERROR FLAG (ERFLG) IS ZERO, THEN
5 TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED
6 CALL XRMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
7 ENDIF
8 CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
9 IF CLASS OF ELEMENT IS DDE, THEN
10 CALL PURGE TO DELETE THE FILE
11 ELSE, ELEMENT RESIDES IN AWA
12 ENDIF
13 IF AWA REQUESTS EXIST IN REQBUF BEYOND FAILING REQUEST, THEN
14 MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
15 ENDIF
16 ELSE
17 SET REQPTR TO 1 INDICATING NO REQUESTS PRESENT
18 ENDIF
19 EXIT XREQ
20 END XREQ
1408 1 CDD
1409 1 CDD  FORTRAN CALLING PROCEDURE
1410 1 CDD
1411 1 CDD  CALL XDBE (DATBUF)
1412 1 CDD
1413 1 CDD
1414 1 CDD  ********
1415 1 CDD
1416 1 CDD  XDBE PROCESSES THE RESTORE DIRECTIVE. THE SPECIFIED UTDB OR HDB
1417 1 CDD  FILE IS OPENED AND ITS TOC IS READ. XDLIS IS CALLED TO MARK
1418 1 CDD  TOC ENTRIES FOR RESTORE. XDBE IS CALLED TO RESTORE DATA
1419 1 CDD  ELEMENTS AND TABLES. XBRDF IS CALL TO RESTORE DBE'S.
1420 1 CDD
1421 1 CDD  ********
1422 1 CDD
1423 1 CDD  INPUT
1424 1 CDD  COMMON XE - LU,  FLAGS, BPATH, BANBUF, TOKENS,
1425 1 CDD  CARTAG, COMPTA, CONBUF
1426 1 CDD
1427 1 CDD  CALLING SEQUENCE
1428 1 CDD
1429 1 CDD  DATBUF - 1408 WORD BUFFER USED TO READ UTDB FILE
1430 1 CDD
1431 1 CDD  ********
1432 1 CDD
1433 1 CDD  OUTPUT
1434 1 CDD
1435 1 CDD  COMMON XE - BANBUF
1436 1 CDD
1437 1 CDD  COMMON XD - ABFLG = ABORT FLAG, NON-ZERO VALUE FROM XDRD,
1438 1 CDD  XDBE, XDLIS OR XDBE INDICATES ABORT
1439 1 CDD  OF RESTORE
1440 1 CDD
1441 1 CDD  ALLFLG = SET NON-ZERO IF LIST OF ELEMENTS SPECIFIED
1442 1 CDD
1443 1 CDD  DEBUG = DEBUG BIT OF 'FLAGS' IN XE COMMON
1444 1 CDD
1445 1 CDD  ENDBLK = BLOCK # OF LAST BLOCK READ
1446 1 CDD
1447 1 CDD  ERFLG = SET NON-ZERO IF MESSAGE #54 ISSUED SO
1448 1 CDD
1449 1 CDD  THAT IT IS ISSUED ONLY ONCE
1450 1 CDD
1451 1 CDD  FILNAM = UTDB/HDB FILE NAME
1452 1 CDD
1453 1 CDD  TOCBUF = UTDB TOC ENTRIES, 8 WORDS EACH, MAX 1200
1454 1 CDD
1455 1 CDD  TOCWDS = MAX DBE TOCENTRIES, MAXIDER TO ENTRY
1456 1 CDD
1457 1 CDD  TOCTOT = TOTAL DBE TOC ENTRY
1458 1 CDD
1459 1 CDD  NOTES
1460 1 CDD
1461 1 CDD  Routines Called
1462 1 CDD
1463 1 CDD
1464 1 CDD
1465 1 CDD
1466 1 CDD
1476 1 BEGIN XDRES
1477 2  SET ABFLG TO ZERO
1478 3  ERREXIT IF NEXT TOKEN IS NOT A COMM :ERRD9:
1479 4  ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERRD9:
1480 5  RETAIN THIS NAME AS DATA BASE TO BE RESTORED
1481 6  INCREMENT TO NEXT TOKEN
1482 7  IF TOKEN IS NOT A COMM, THEN
1483 8  ERREXIT IF TOKEN IS NOT EOS :ERR04:
1484 9  ENDEF
1485 10  BUILD AWA REQUEST FOR TOC ENTRY RETRIEVE
1486 11  CALL AWA TO PROCESS AWA REQUEST
1487 12  ERREXIT IF AWA REQUEST FAILED :ERR10:
1488 13  ERREXIT IF DATA BASE FOUND IS A PDB :ERR33:
1489 14  IF DATA BASE IS A UTBD, THEN
1490 15  CALL XRMFM TO CONSTRUCT QUALIFIED FILE NAME
1491 16  ENDF
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 INTO TOCBUF
1495 20  ERREXIT IF READ FAILED :ERR08:
1496 21  INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1497 22  IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1498 23  CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1499 24  ERREXIT IF READ FAILED :ERR08:
1500 25  ENDF
1501 26  UPDATE TOTSIZ TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 27  CLEAR ERROR MESSAGE FLAG (ERRFLG)
1503 28  IF TOKEN IS EOS (I.E., NO LIST OF ELEMENTS), THEN
1504 29  SET ALLFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 30  ELSE
1506 31  SET ALLFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 32  CALL XDLIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 33  EXIT XDRES IF ABFLG SET BY XDLIS
1509 34  ENDF
1510 35  CALL XDREO TO RESTORE AWA RESIDENT ELEMENTS
1511 36  1 EXIT XDRES IF ABFLG SET BY XDREO
1512 37  2 CALL ZDRDF TO RESTORE DRDE'S
1513 38  EXIT XDRES IF ABFLG SET BY ZDRDF
1514 39  2 CALL CLOSE TO CLOSE DATA BASE FILE
1515 40  ERREXIT IF CLOSE FAILED :ERR08:
1516 41  DO WHILE AWA REQUESTS REMAIN IN ERRBUF
1517 42  CALL XDREO TO PROCESS AWA REQUESTS
1518 43  EXIT XDRES IF ABFLG SET BY XDREO
1519 44  ENDOO
1520 45  1 EXIT XDRES
1521 46  :ERR04: CALL XRMFG - 'SYNTAX ERROR. EXTRMEOS DATA'
1522 47  :ERR08: CALL XRMFG - 'FILE MANAGER ERROR .... .......
1523 48  :ERR09: CALL XRMFG - 'SYNTAX ERROR. MISSING OR ILLEGAL FIELD'
1524 49  :ERR10: CALL XRMFG - '........ NOT FOUND'
1525 50  :ERR33: CALL XRMFG - 'CAn NOT RESTORE A PDB'
1526 51  DO UNTIL ALL AWA REQUESTS IN ERRBUF HAVE BEEN PROCESSED
1527 52  IF REQUEST IS TO STORE DATA, THEN
1528 53  CALL EXEC TO FREE THE SPECIFIED CLASS NO. AND SAM BUFFER
1529 54  ENDF
<table>
<thead>
<tr>
<th>FORTRAN CALLING PROCEDURE</th>
<th>CALL</th>
<th>XERAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL XERAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XERAY PROCEDURE THE SAVE DIRECTIVE A PDB IS CREATED AND THE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTENTS OF THE SPECIFIED UDB ARE COPIED TO IT.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMON XE - CARTAG, COMBOF, COMPTA, FLACS, LV, TOKENS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILES - XEER (UDB FILE SPECIFIED)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMON XE - REGBUF, REPTPR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILES - XEERQ (PDB FILE SPECIFIED)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERNAL VARIABLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBIPR - DDB FOR THE UDB FILE ALLOCATED IN THE COMMON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTAINS 1152 WORD BUFFER USED TO READ THE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTOB AND TO WRITE THE PDB.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5-56
1 BEGIN YDSAV
2  SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
3  IF TIT "*" IS NOT NEXT TOKEN :ERROR:
4  INCREMENT TO NEXT TOKEN
5  ERREXIT IF TIT IS NOT "NAME" :ERROR:
6  ERREXIT IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR:
7  ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
8  RETAIN THIS NAME AS UTD
9  INCREMENT TO NEXT TOKEN
10 ERREXIT IF "*" IS NOT NEXT TOKEN :ERROR:
11  INCREMENT TO NEXT TOKEN
12 ERREXIT IF TOKEN IS NOT "NAME" :ERROR:
13 ERREXIT IF TOKEN IS MORE THAN 4 CHARACTERS :ERROR:
14 ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
15 RETAIN THIS NAME AS PDB
16  INCREMENT TO NEXT TOKEN
17 ERREXIT IF TOKEN IS NOT EOS :ERROR:
18 BUILD AWA REQUEST TO RETRIEVE UTD'S TOC ENTRY
19 CALL XKEN TO PROCESS AWA REQUESTS
20 ERREXIT IF TOC RETRIEVE FAILED :ERROR:
21 ERREXIT IF TOC ENTRY DOES NOT INDICATE UTD :ERROR:
22 CALL XODDA TO ADD PDB TO UTD
23 ERREXIT IF FMER ERROR :ERROR:
24 ERREXIT IF PDB LIMIT EXCEEDED :ERROR:
25 ERREXIT IF PDB LIMIT EXCEEDED :ERROR:
26 CALL XDKEN TO BUILD UTD FILE NAME
27 CALL OPEN TO OPEN UTD FILE (SPECIFYING TYPE 1)
28 ERREXIT IF OPEN FAILED :ERROR:
29 SET STATUS FLAG INDICATING PDB LOGGED
30 CALL XDKEN TO BUILD UTD FILE NAME
31 CALL OPEN TO OPEN UTD FILE -- USE SIZE OF UTD:
32 CALL XKEN TO PROCESS AWA REQUEST
33 ERREXIT IF PDB IS DUPLICATE :ERROR:
34 ERREXIT IF AWA OVERFLOW :ERROR:
35 SET STATUS FLAG INDICATING PDB LOGGED IN AWA
36 CALL XKEN TO BUILD PDB FILE NAME
37 CALL CRKT TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE
38 OF UTD FILE ELSE TOC ENTRY
39 ERREXIT IF CREAT FAILED :ERROR:
40 SET STATUS FLAG INDICATING PDB FILE OPEN
41 DO FOR ALL DATA IN UTD FILE
42 CALL ... ADV TO READ 1 BUFFER OF UTD DATA
43 ERREXIT IF READ FAILED :ERROR:
44 CALL WRITI TO WRITE 1 BUFFER TO PDB FILE
45 ERREXIT IF WRITE FAILED :ERROR:
46 ENDDO
47 CLOSE UTD FILE
48 CLOSE PDB FILE
49 EXIT YDSAV
50 :ERROR: ISSUE MESSAGE "SYNTAX ERROR -- ILLEGAL OR MISSING FIELD"
51 :ERROR: ISSUE MESSAGE "INVALID UTD FILE NAME ...."
52 :ERROR: ISSUE MESSAGE "INVALID PDB FILE NAME ...."
53 :ERROR: ISSUE MESSAGE "SYNTAX ERROR, EXTRANEOUS DATA"
54 :ERROR: ISSUE MESSAGE "PDB .... ALREADY EXISTS"
1625 2 ERR10: ISSUE MESSAGE "PRO FILE ACCESS ERROR ..."
1626 2 ERR00: ISSUE MESSAGE "FILE MANAGER ERROR ....."
1627 2 ERR20: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PRO'S 
1628 2 ALREADY REACHED"
1629 2 ERR21: ISSUE MESSAGE "ANA OVERFLOW. NOT LOGGED"
1630 2 RETURN:
1631 2 IF STATUS FLAG INDICATES PRO FILE IS OPEN, THEN
1632 2 PURGE PRO FILE
1633 2 ENDIF
1634 2 IF FLAG INDICATES PRO IS IN AMA, THEN
1635 2 CALL REN TO DELETE PRO FROM AMA
1636 2 ENDIF
1637 2 IF FLAG INDICATES UTOB IS OPEN, THEN
1638 2 CLOSE UTOB
1639 2 ENDIF
1640 2 IF FLAG INDICATES PRO IS IN XPRO, THEN
1641 3 CALL XPROD TO DELETE PRO FROM XPRO
1642 2 ENDIF
1643 1 END XBSAV
FORTran CALLING PROCEDURE

1 CALL XOSTA (LU)

**********

**1** OUTPUT AMA AND DMA USAGE DATA AND STATISTICS ON AMA COMPACTION

**1** PHASES

**1**

**1**

**1** INPUT

**1**

**1** LU — LOGICAL UNIT OF OUTPUT DEVICE

**1**

**1**

**1** XB COMMON

**1**

**1** ASZ — TOTAL SIZE OF AMA

**1**

**1** FFE — AMOUNT OF FREE SPACE IN AMA

**1**

**1** DSS — TOTAL SIZE OF DMA (IN 64 WORD SECTORS)

**1**

**1** DFR — AMOUNT OF FREE SECTORS IN DMA

**1**

**1** HP — ARRAY OF THREE COMPACTION PHASE COUNTS

**1**

**1**

**1**

**1**

**1** OUTPUT

**1**

**1** USAGE AND STATISTICS TO LU

**1**

**1** NOTES

**1**

**1** USES EXEC, XRIG

**1**

**1**

**1**

**1**

**1**

**1** BEGINS XOSTA

**1**

**1** FORMAT AND PRINTAMA USAGE DATA

**1**

**1** FORMAT AND PRINTDMA USAGE DATA

**1**

**1** FORMAT AND PRINT COMPACTION STATISTICS

**1**

**1** END XOSTA
1737 1 BEGIN XSTO,
1738 2: ;: ABFLG TO ZERO (ABORT FLAG)
1739 3: ERREXIT IF UTDB NAME IS NOT VALID TO :ERR2:
1740 4: BUILD REQUEST FOR AWA TOC
1741 5: CALL XREQ TO MAKE MANAGER REQUEST
1742 6: CALL EXEC TO GET AWA TOC
1743 7: ERREXIT IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1744 8: ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1745 9: ENDOD
1746 10: SET ERFLG OFF (ERROR MESSAGE FLAG)
1747 11: SET TOTSIZ = 0 (UTDB TOTAL SIZE)
1748 12: IF WHOLE AWA IS TO BE STORED THEN
1749 13: DO FOR IT, ST, DE, ORDE AWA TOC ENTRIES
1750 14: DO FOR EACH ENTRY IN THIS CHAIN
1751 15: IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1752 16: PREFIX IS NOT AN AMPERSAND THEN
1753 17: SET STORE/RESTORE BIT ON IN TOC ENTRY
1754 18: INCREMENT TOTSIZ BY SIZE OF THIS ELEMENT
1755 19: ENDIF
1756 20: ENDDO
1757 21: ELSE
1758 22: CALL XOLIS TO PROCESS LIST TO BE STORED
1759 23: ERREXIT IF ABFLG IS NOT ZERO TO :ERR2:
1760 24: ENDIF
1761 25: SET NOTOC = 0 (NUMBER OF UTDB TOC ENTRIES)
1762 26: DO FOR IT, ST, DE, ORDE CHAINS
1763 27: DO FOR EACH ENTRY IN THIS CHAIN
1764 28: IF STORE/RESTORE BIT IS ON THEN
1765 29: TURN STORE/RESTORE BIT OFF
1766 30: BUILD UTDB TOC ENTRY
1767 31: INCREMENT NOTOC BY 1
1768 32: ENDIF
1769 33: ENDDO
1770 34: ENDDO
1771 35: ERREXIT IF THERE ARE NO UTDB TOC ENTRIES (NOTOC=0) TO :ERR2:
1772 36: COMPARE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1773 37: CALL XP” ’ TO CREATE FILE NAME
1774 38: CALL TO CREATE UTDB FOR TOTSIZ
1775 39: ERREXIT IF ERROR IN CREATE TO :ERR3:
1776 40: CALL XWRITE TO WRITE UTDB FILE
1777 41: ERREXIT IF ABFLG IS 4 (DRDE LARGER THAN SPECIFIED) TO :ERR1:
1778 42: ERREXIT IF ABFLG IS 3 (DRDE FILE ERROR) TO :ERR4:
1779 43: ERREXIT IF ABFLG IS 2 (UTDB FILE ERROR) TO :ERR3:
1780 44: CALL XWRITE TO WRITE TOC RECORDS AT RECORD 1
1781 45: ERREXIT IF ERROR IN WRITE TO :ERR3:
1782 46: CALL CLOSE TO CLOSE UTDB FILE
1783 47: ERREXIT IF ERROR IN CLOSE TO :ERR3:
1784 48: BUILD REQUEST TO ALLOCATE UTDB IN AWA
1785 49: CALL XREQ TO MAKE REQUEST
1786 50: ERREXIT IF AWA OVERFLOW TO :ERR1:
1787 51: EXIT XSTO
1788 52: :ERR1:
1789 53: CALL XMSG TO DISPLAY MSGNO
1790 54: GO TO :ERR4:
1791 55: :ERR2:
1805 1 CD0 FORTRAN CALLING PROCEDURE
1806 1 CD0 CALL XDTOC
1807 1 CD0
1808 1 CD0
1809 1 CD0
1810 1 CD0
1811 1 CD0 FDS AWA/DWA TABLE OF CONTENTS DIRECTIVE HANDLER. XDTOC INTER-
1812 1 CD0 PRETS THE TOC DIRECTIVE, RETRIEVES THE INDICATED TOC (AWA OR
1813 1 CD0 DATA BASE FILE) AND FORMATS AND OUTPUTS THE REQUESTED ENTRIES.
1814 1 CD0
1815 1 CD0
1816 1 CD0 INPUT
1817 1 CD0 X COMMON - COMBUF, COMSTR, FLAGS, LU
1818 1 CD0 MANAGER - AWA HEADER AND TOC (SEE MODULE XMWA)
1819 1 CD0 FILES - DATA BASE FILES AS APPROPRIATE
1820 1 CD0
1821 1 CD0
1822 1 CD0 OUTPUT
1823 1 CD0 X COMMON - COMBUF, RERBUF, REPSTR
1824 1 CD0 PRINTED OUTPUT - AWA OR UTDB TOC DISPLAY
1825 1 CD0
1826 1 CD0
1827 1 CD0 LOCAL VARIABLES
1828 1 CD0 ASIZ - TOTAL ALLOCATABLE SIZE OF AWA (SEE MODULE XMWA)
1829 1 CD0 CLASS - FIRST PORTION OF AWA (HEADER AND TOC)
1830 1 CD0 CLASS - ARRAY OF ALLOCABLE CLASS DESIGNATORS FOR TOC DIRECTIVE
1831 1 CD0 CLS - CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE
1832 1 CD0 CODE - CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'
1833 1 CD0 EC - END OF TOC CHAIN DESIGNATOR (-32768)
1834 1 CD0 FRE - CURRENT AMOUNT OF FREE SPACE IN AWA (SEE MODULE XMWA)
1835 1 CD0 HC - ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)
1836 1 CD0 IDIM - TOC ENTRY DIM FIELD (WORD 8)
1837 1 CD0 LINE - BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE
1838 1 CD0 MULT - ARRAY OF MULTIPLIERS USED TO DETERMINE NUMBER OF WORDS
1839 1 CD0 IN NUMBERS OF TWO DIMENSIONAL DATA ELEMENTS (FUNCTION OF
1840 1 CD0 DATA TYPE)
1841 1 CD0 OPTION - ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY
1842 1 CD0 OORG - ORIGIN ADDRESS OF AWA (SEE MODULE XMWA)
1843 1 CD0 POS - VALUE OF 2 OR 22 INDICATING FIRST OR SECOND POSITION IN
1844 1 CD0 TOC DISPLAY LINE
1845 1 CD0 SIZE - TOC ENTRY SIZE FIELD (WORD 7)
1846 1 CD0 TCMAX - MAXIMUM SIZE OF TOC WHICH CAN BE TOTALLY ACCOMMODATED BY
1847 1 CD0 INTERNAL BUFFER
1848 1 CD0 TYPE - TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)
1849 1 CD0 TYPEID - ARRAY OF TYPE ID CODES FOR DE AND ORDE DISPLAYS
1850 1 CD0 UNIT - LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY
1851 1 CD0
1852 1 CD0
1853 1 CD0 NOTES
1854 1 CD0 USES CLOSE, EXEC, IAND, IXOR, KCVT, OPEN, READF, XDTA, XREG,
1855 1 CD0 XRIG, XRMOV, XRSG, XRSG, XRSHF, XRUSG
1856 1 CD0
1857 1 CD0 IN THE EVENT THE COMPACTED AWA TOC AND HEADER DATA WILL NOT FIT
1858 1 CD0 IN THE ALLOCATED BUFFER SPACE, AS MUCH OF EACH CHAIN AS POSSIBLE
1859 1 CD0 WILL BE DISPLAYED UNTIL THE LINK FIELDS LEAD BEYOND THE BUFFER.
1860 1 CD0
1861 1 CD0
1862 1 CD0 IT IS ASSUMED THAT XDTOC AND XDST USE THE SAME SIZE BUFFER FOR
1863 1 CD0 TOC MANIPULATION; THUS, A DATA BASE FILE MAY NOT HAVE A TOC TOO
1864 1 CD0 LARGE FOR THE XDTOC INTERNAL BUFFER.
1864 1 BECM XDOTC
1865 2 (3) FOR OUTPUT OF ALL USER CLASSES FROM AWA TO TERMINAL
1866 3 IF NEXT TOKEN IS NOT EOS
1867 4 IF TOKEN IS A HYPHEN
1868 5 THEN
1869 6 INCERTEN TO NEXT TOKEN
1870 7 IF TOKEN IS NOT A CHARACTER T
1871 8 THEN
1872 9 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
1873 10 THEN
1874 11 SET OUTPUT UNIT FOR LINE PRINTER
1875 12 ENDIF
1876 13 INCREMENT TO NEXT TOKEN
1877 14 ENDIF
1878 15 IF TOKEN IS NOT EOS
1879 16 THEN
1880 17 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1881 18 INCREMENT TO NEXT TOKEN
1882 19 IF TOKEN IS NOT A COMMA
1883 20 THEN
1884 21 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (0, F, I, S OR B)
1885 22 SET OPTION FOR INDICATED CLASS
1886 23 INCREMENT TO NEXT TOKEN
1887 24 ENDIF
1888 25 IF TOKEN IS NOT EOS
1889 26 THEN
1890 27 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1891 28 INCREMENT TO NEXT TOKEN
1892 29 EXIT TO :ERROR: IF TOKEN IS NOT A FOUR CHARACTER NAME
1893 30 INCREMENT TO NEXT TOKEN
1894 31 EXIT TO :ERROR: IF TOKEN IS NOT EOS
1895 32 CALL XREQ TO RETRIEVE DATA BASE FILE TOC ENTRY
1896 33 EXIT TO :ERROR: IF NOT LOGGED IN TOC
1897 34 SET TYPE AND SET NAME IN HEADER
1898 35 CONSTRUCT FILE NAME
1899 36 READ FIRST DATA BASE FILE TOC RECORD
1900 37 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AWA TOC
1901 38 READ SUBSEQUENT TOC RECORDS
1902 39 EXIT TO :ERROR: IF FILE ACCESS FAILS
1903 40 DO FOR EACH NON-EMPTY TOC CHAIN
1904 41 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
1905 42 IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NULL CHAIN EXISTED
1906 43 THEN
1907 44 MARK PREVIOUS ENTRY AS AN END OF CHAIN
1908 45 ENDIF
1909 46 EMDO
1910 47 DO FOR EACH ENTRY IN TOC
1911 48 IF NOT MARKED AS AN END OF CHAIN
1912 49 THEN
1913 50 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
1914 51 ENDIF
1915 52 EMDO
1916 53 INCLUDE CHAIN 8 (DATA BASE FILES) FROM DISPLAY
1917 54 ENDIF
1918 55 EMDO
1919 56 IF REFERENCING AWA TOC
1920 57 THEN
1921 58 CALL XREQ TO RETRIEVE AWA TOC
IF DEBUG AND/OR TRACE FLAGS ARE SET
THEN
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
THEN
FORMAT I-DIM FIELD IN OCTAL
ENDIF
FORMAT TYPE FIELD IN INTEGER
IF CLASS 2 OR 3
THEN
FORMAT TYPE FIELD USING DATA TYPE TABLE
IF CLASS 2
THEN
FORMAT I-DIM & J-DIM FIELDS
ENDIF
ELSE
IF CLASS 8
THEN
FORMAT TYPE FIELD USING FILE TYPE TABLE
ENDIF
ENDIF
ELSE
PRINT 'DATA LOST' MESSAGE
EXIT PROCESSING FOR THIS CHAIN
ENDIF
PRINT ENTRY
ENDO
IF PROCESSING AWA TOC
THEN
CALL XSTA TO DISPLAY AWA USAGE STATISTICS
ENDIF
EXIT XDTOC
:ERRORS: EXIT WITH INVALID OUTPUT DEVICE ID
:ERROR4: EXIT WITH SYNTAX ERROR
:ERRORS: EXIT WITH INVALID CLASS DESIGNATOR
:ERROR6: EXIT WITH INVALID UTDB FILE NAME
:ERROR7: EXIT WITH UTDB FILE ACCESS ERROR
1 END XDTOC
BEGIN XDRT
SET WRDNO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
SET MORE = 0 (NUMBER OF AVA REQUESTS IN RENBUF)
DO FOR ALL UTDOR TOC ENTRIES UNTIL CLASS IS DRDE
BUILD REQUEST FOR DATA FROM AVA
INCREMENT MORE BY 1
IF REQUEST BUFFER IS FULL (MORE=8) THEN
PERFORM REQDAT TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
ENDIF
ENDDO
IF THERE ARE REMAINING REQUESTS (MORE>0) THEN
SET NEXT REQUEST TO BE END OF REQUEST LIST
PERFORM REQDAT TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
IF THERE IS A PARTIAL DATA RECORD LEFT (WRDNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDOR FILE
ENDIF
ENDIF
DO FOR EACH DRDE UTDOR TOC ENTRY
STORE DATREC IN UTDOR TOC ENTRY
CALL DRNFM TO CREATE FILE NAME
IF DRDE FILE IS TYPE 3 THEN
CALL OPEN TO OPEN FILE AS CORRECT TYPE
ERREXIT IF OPEN ERROR TO :ERR1:
DO UNTIL EOF IS READ
CALL READ TO READ 1 RECORD
ERREXIT IF READ ERROR TO :ERR1:
STORE RECORD LENGTH AT FRONT AND REAR OF DATA
INCREMENT WRDNO BY LENGTH + 2
IF THERE IS ENOUGH DATA TO WRITE (WRDNO>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDOR FILE
ENDIF
ENDIF
IF THERE IS REMAINING DATA (WRDNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDOR FILE
ENDIF
ENDIF
SET DATREC TO NEXT AVAILABLE RECORD FOR DATA ELSE
CALL OPEN TO OPEN FILE AS TYPE 1
ERREXIT IF OPEN ERROR TO :ERR1:
COMPUTE TOTAL SIZE OF FILE IN WORDS
DO UNTIL ALL DATA IS COPIED TO UTDOR (SIZE=0)
IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
SET LENGTH = SIZE
ENDIF
CALL READ TO READ LENGTH DATA
ERREXIT IF READ ERROR TO :ERR1:
CALL WRITF TO WRITE LENGTH DATA
ERREXIT IF WRITE ERROR TO :ERR3:
INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
DECOLUMN SIZE BY LENGTH IN WORDS WRITTEN
ENDIF
ENDIF
CALL CLOSE TO CLOSE DRDE FILE
ERREXIT IF CLOSE ERROR TO :ERR1:
ENDIF
EXIT XDRT
1 C00 FORTRAN CALLING PROCEDURE
2 C00 CALL XELDS (THREE WORD ARRAY CONTAINING 'XECAL')
3 C00
4 C00
5 C00 FDS EXECUTIVE INITIALIZATION SEGMENT MAIN ROUTINE
6 C00
7 C00
8 C00 INPUT
9 C00 COMMON XE - SUBSTA
10 C00
11 C00
12 C00 COMMON XB - INITIALIZED ACCORDING TO THE VALUE OF SUBSTA
13 C00 (SEE APPROPRIATE INITIALIZATION SUBROUTINE)
14 C00
15 C00 NOTE?
16 C00 USES XEIN, XEINI, XEINM, XEINX, XETH
17 C00
18 C00
19 C00
20 C00 BEG XECAL
21 C00 CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SREDT:, :INTED:) SUBSTA (-1 TO 3)
22 C00
23 C00 :
24 C00
25 C00
26 C00
27 C00
28 C00
29 C00
30 C00 2:CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SREDT:, :INTED:) SUBSTA (-1 TO 3)
31 C00 3 :GLOBAL: CALL XEIN TO INITIALIZ GLOBL COMMON
32 C00 3 :DIRECT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES
33 C00 3 :EXECUT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL
34 C00 3 :SREDT: CALL XEIN TO INITIALIZ DYNAMIC COMMON FOR SEQUENCE EDITING
35 C00 3 :INTED: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING
36 C00 2:ENDCASE
37 C00 1:END XECAL
39  1 CDO          FORTRAN CALLING PROCEDURE
40  1 CDO
41  1 CDO  CALL XEIND
42  1 CDO
43  1 C********
44  1 CDO
45  1 CDO   INITIALIZE XB COMMON FOR DIRECTIVE LEVEL
46  1 CDO
47  1 C********
48  1 CDO
49  1 CDO   COMMON XE - CARTRG, FLAGS
50  1 CDO
51  1 CDO
52  1 C********
53  1 CDO
54  1 CDO   COMMON XB - DIRECT, NUMDIR
55  1 CDO
56  1 C********
57  1 CDO
58  1 CDO   USES CLOSE, EXEC, OPEN, PRM, READ, XREXT, XRIG, XRMSG, XVABN
59  1 CDO
60  1 C********
61  1 *
62  1 *
63  1 *
64  1 *
65  1 BEGIN XEIND
66  2 READ FDS DIRECTIVE PROMPT FILE
67  2 STORE DIRECTIVES IN DYNAMIC COMMON
68  1 END XEIND
108 1 BEGIN XEINE
109 2 INITIALIZE COMMON TO ZEROS
110 3 SET FILE (ART=.DCE NUMBER
111 4 SET STATES TO DIRECTIVE LEVEL
112 5 SET TKNLNG
113 6 INITIALIZE TOKENS
114 7 READ LIBRARY DIRECTORY FIRST RECORD
115 8 EXIT TO :ERROR: IF READ FAILED
116 9 STORE NUMBER OF PROCESSORS IN NPROC
117 10 READ PROCESSOR NAME RECORD
118 11 EXIT TO :ERROR: IF READ FAILED
119 12 CALL XERG TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AMA
120 13 EXIT TO :ERROR: IF REQUEST FAILED
121 14 DO FOR EACH ID SEGMENT IN SYSTEM
122 15 DO FOR EACH PROCESSOR IN LIBRARY DIRECTORY
123 16 IF NAMES ARE THE SAME
124 17 THEN
125 18 NOTE EXISTANCE OF ID
126 19 ENDIF
127 20 ENDDO
128 21 IF ANY PROCESSORS NOT MARKED
129 22 THEN
130 23 LIST PROCESSORS MISSING ID SEGMENTS
131 24 EXIT XEINE WITH INITIALIZATION FAILURE
132 25 ENDF
133 26 CALL OPEN TO OPEN PDB/MDB DIRECTORY ( XPDB)
134 27 EXIT TO :ERROR: IF OPEN FAILED
135 28 CALL READF TO READ 1ST RECORD OF XPDB (LIST OF MDB'S)
136 29 EXIT TO :ERROR: IF READ FAILED
137 30 PERFORM DLOG TO LOG MDB'S IN AMA
138 31 CALL READF TO READ QMB'S FOR THIS QUALIFIER
139 32 CALL READF TO READ THAT RECORD OF XPDB
140 33 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)
141 34 CALL CLOSE TO CLOSE XPDB
142 35 PERFORM DLOG TO LOG QMB'S IN AMA
143 36 IF THERE ARE REQUESTS IN THE AMA REQUEST BUFFER, THEN
144 37 CALL XERG TO PROCESS THE AMA REQUESTS
145 38 ENDF
146 39 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
147 40 EXIT XEINE
148 41 BEGIN DLOG
149 42 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
150 43 BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
151 44 IF 8 AMA REQUESTS HAVE BEEN BUILT, THEN
152 45 CALL XERG TO PROCESS AMA REQUESTS
153 46 ENDF
154 47 EXIT TO :ERROR: IF A REQUEST FAILED
155 48 ENDDO
156 49 END DLOG
157 50 :ERROR: LIBRARY INITIALIZATION ERROR TERMINATION
158 51 END XEINE
FORTRAN CALLING PROCEDURE

CALL XEINI

*********

INITIALIZE XE AND XD COMMON FOR INTERFACE TABLE EDITING

*********

INPUT

COMMON XE -

MASTA = MASTER STATE

TOREMS = LEXICAL TOKEN VALUES

NPROC = NUMBER OF ENTRIES IN LIBRARY DIRECTORY

COMPTN = INDEX OF NEXT TOKEN IN COMBUF

COMBUF = COMMUNICATIONS BUFFER

PRCNAM = PROCESSOR NAME FROM EXECUTION CONTROLLER

LIBRARY -

INLIB = OLD INTERFACE TABLE TO BE EDITED

<XXXXX = DEFAULT INTERFACE TABLE

>XXXXX = PROMPT TABLE

OUTPUT

*********

COMMON XE -

SUBSTA = SUBSTATE -- SET TO 1 IF INIT. ERROR

RESBUF = REQUEST BUFFER FOR XA Inputs

PRCNAM = PROCESSOR NAME ON INTE DIRECTIVE

COMMON XD -

WKRLNG = LENGTH OF WKBUF (CONSTANT)

WKBUF = WORKING BUFFER CONTAINING IN/E. TABLE WITH

SHORT PROMPTS

DIRECT = SUPPORTED INTE DIRECTIVES

NUMDIR = NO. OF ENTRIES IN DIRECT

MINTAB = MINT. TABLE NAME

NUMARG = NO. OF ARGUMENTS

WARG = NO. OF WORDS IN SP.C. AND HEADER OF WKBUF

LITLEN = NO. OF WORDS IN LITERAL AREA

LITPTR = INDEX TO START OF LITERAL AREA

ISIZES - ARRAY MAPPING ARG. TYPE TO EFFECTIVE LENGTH

*********

NOTES

USES FILES -

<XXXXX = DEFAULT INTERFACE TABLE FOR THIS PROCESSOR

>XXXXX = PROMPT TABLE FOR THIS PROCESSOR

USES ROUTINES

XRMOV = EXEC

XRMSG = CLOSE

XREXT = OPEN

XREG = READF

XRPK = XRNAM

XRCPH

BEGIN XEINI

IF CALLED AS A RESULT INTE DIRECTIVE, THEN
POSITION TO 1ST TOKEN AFTER 'INTE'
ERROR IF TOKEN IS NOT ':', :ERROR1:
INCREMENT TO NEXT TOKEN
ERROR IF Token IS NOT A NAME :ERROR2:
SET PRNAM TO THIS NAME
INCREMENT TO NEXT TOKEN
SET INTNAM TO 0
SET NEWTAB TO 'INTAB'
IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN
ERROR IF TOKEN IS NOT A COMMA, :ERROR2:
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
ERROR IF TOKEN IS NOT A COMMA :ERROR2:
INCREMENT TO NEXT TOKEN
IF TOKEN IS A NAME, THEN
SET NEWTAB TO THE NAME
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
ERROR IF TOKEN IS NOT EOS :ERROR2:
ENDIF
ELSE
INTNAM AND PRNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER
NEWTAB IS SET TO 'INTAB'
ISSUE MESSAGE THAT INTERFACE TABLE EDITOR BEEN INVOKED
ENDIF
MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE
ERROR IF REQUEST IS UNSUCCESSFUL :ERROR11:
START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR
EXIT IF PROCESSOR PRNAM IS FOUND
ELSE
INCREMENT TO NEXT ENTRY
ENDLOOP
ERROR :ERROR8:
ENDSEARCH
SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND
IF INTNAM = 0, THEN
CONSTRUCT THE NAME OF THE DEFAULT INTERFACE TABLE AS "PRCNAM"
OPEN THE DEFAULT INTERFACE TABLE FILE
ERROR IF OPEN FAILED :ERROR2:
READ THE DEFAULT INTERFACE TABLE
ERROR IF THE READ FAILED :ERROR16:
READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM
OF WKBUF
ERROR IF READ FAILED :ERROR14:
ELSE
MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE
ERROR IF ERROR OR COULD NOT FIND :ERROR:
ENDIF
ERROR IF VERS .WE. VERSION NO. OF TABLE TO BE EDITED :ERROR10:
CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"
OPEN THE PROMPT TABLE
ERROR IF OPEN FAILED :ERROR15:
READ 2ND RECORD OF THIS PROMPT TABLE
ERREXIT IF READ FAILED: ERR17:
MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUF TO AREA FOLLOWING
THE PROMPT TABLE
THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
LITERAL DISPL. THAT MATCH EACH LITERAL DATA ENTRY
THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
EXIT XEINI

:ERR2: "SYNTAX ERROR ..."
:ERR4: "INTNAM" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNAM DOES MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
FORTRAN CALLING PROCEDURE

CALL XEINS

INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING

INPUT

COMMON XE - CARTRG, COMBUF, C OMPTR, LU, NPROC, TOKENS

OUTPUT

COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,
OLDTAB, PRMTAB, WKBLNG, WKBUI, XLIBD

NOTES

USES FILES - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE

USES ROUTINES

EXEC

IAND

OPEN

READ

XCPTR

XEQ

XEXIT

XRMOV

XRMSG

XUDBG

**********
350 1 BEGIN XEIMS
351 2 BUILD AXA REQUEST TO RETRIEVE "XLIBD"
352 3 ERREXIT IF TOKEN IS NOT COMMA :ERR02:
353 4 INCREMENT TO NEXT TOKEN
354 5 IF TOKEN IS NAME, THEN
355 6 BUILD AXA REQUEST TO RETRIEVE OLDTAB
356 7 RETAIN THIS NAME AS OLDTAB
357 8 INCREMENT TO NEXT TOKEN
358 9 ELSE
359 10 SET OLDTAB TO ZERO
360 11 ERREXIT IF TOKEN IS NOT COMMA :ERR02:
361 12 INCREMENT TO NEXT TOKEN
362 13 ERREXIT IF TOKEN IS NOT NAME :ERR02:
363 14 CALL XREQ TO PROCESS AXA REQUEST(S)
364 15 IF OLDTAB NOT EQUAL TO OLDTAB, THEN
365 16 BUILD AXA REQUEST TO VERIFY EXISTENCE OF NEWTAB
366 17 ERREXIT IF NO ERROR FROM XREQ :ERR12:
367 18 ENDIF
368 19 IF OLDTAB NOT ZERO, THEN
370 20 SET PROMPT MODE AS UPDATE
371 21 SET NO. ENTRIES AS OLDTAB SIZE / 7
372 22 READ OLDTAB INTO WORKING BUFFER
373 23 ERREXIT IF FIRST REQUEST (RETRIEVE "XLIBD") FAILED :ERR01:
374 24 ELSE
375 25 SET PROMPT MODE AS CREATE
376 26 SET NO. ENTRIES TO ZERO
377 27 ERREXIT
378 28 OPEN, READ AND CLOSE FILE XSPRM
379 29 SET COUNT AND SEREDIT DIRECTIVES INTO XSP COMMON
380 30 EXIT XEIMS
381 31 :ERR01: CALL XREX - "INITIALIZATION ERROR ...."
382 32 :ERR02: CALL XREX - "SYNTAX ERROR"
383 33 :ERR04: DEFAULT MESSAGE TO '....NOT FOUND'
384 34 IF ERROR WAS NO AXA SPACE THEN
385 35 SET MSG TO '....NO AXA SPACE'
386 36 CALL XREX - ".... ALREADY EXISTS"
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON X8 - LITPTR, NUMARS, WKBLNG, WKBUF

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

OUTPUT

COMMON X8 - LITPTR, LITDN, NARC, WDSUF

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

USES Routines

XIEXT
XRMOV
XRNKB
XRNKG
XRSET
BEGIN EXIT
DO UNTIL ALL LITERAL ENTRIES HAVE BEEN PROCESSED
START SEARCH UNTIL ALL ARGUMENTS SEARCHED
EXIT IF 'DISP' FIELD FOR ARGUMENT = 'DISP.' OF LITERAL ENTRY, AND
I-BIT IS ON
BUILD 3 WORD LITERAL ENTRY (3, I-SUB, J-SUB)
EXIT IF 'DISP' FIELD FOR ARGUMENT = 'DISP.' OF LITERAL ENTRY, AND
D-BIT IS ON
IF ARGUMENT IS COMPLETE (C-BIT ON), THEN
CREATE BIT MASK WORD(S) IN NEW LITERAL ENTRY AREA
ELSE
MOVE BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA
ENDIF
DO UNTIL ALL BITS OF BIT MASK PROCESSED
DETERMINE NUMBER OF CONTIGUOUS BITS ON (OR OFF)
AND MOVE CORRESPONDING NUMBER OF DATA WORDS
(C-OR O'S) INTO NEW LITERAL ENTRY AREA
ENDDO
ORELSE
INCREMENT TO NEXT ARGUMENT
ENDLOOP
EXIT IF NOT CALLED BY LIBRARY MAINTENANCE :ERR5:
SKIP TO NEXT LITERAL ENTRY
ENDSEARCH
EXIT TO :RETURN:
:ERR5: CALL XRM6G - 'SYSTEM INITIALIZATION ERR 5'
:RETURN:
END EXIT
FORTRAN CALLING PROCEDURE

CALL XEIMX

INITIALIZE XE AND XB COMMON FOR EXECUTION CONTROLLER

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

SEQUENCE TABLE, LIBRARY DIRECTORY NAME TABLE

COMMON XB - LIBBUFF, NOPROC2, SELNG, WKBUFF

NOTES

USES EXEC, PRM, XREQ, XREXT, XRI6, XRMOV, XRMSG, XUDEG, XVABN
BEGIN XEINX
1 IF INITIALIZATION FROM DIRECTIVE
2 IF DIRECTIVE IS NAME
3 THEN
4 EXIT TO :ERROR: IF NEXT TOKEN IS NOT EOS
5 ELSE
6 IF DIRECTIVE IS AUTO
7 THEN
8 IF TOKEN IS A HYPHEN
9 THEN
10 INCREMENT TO NEXT TOKEN
11 EXIT TO :ERROR3: IF TOKEN IS NOT THE NAME 'T'
12 CHANGE EXECUTION CONTROL MODE TO AUTO-T
13 INCREMENT TO NEXT TOKEN
14 ENDIF
15 ENSIF
16 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A COMMA
17 INCREMENT TO NEXT TOKEN
18 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A NAME
19 STORE NAME IN SEGNAM
20 CALL XREG TO RETRIEVE SEQUENCE TABLE
21 EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
22 INCREMENT TO NEXT TOKEN
23 SET SEGSTA TO FIRST SEQUENCE NUMBER
24 SET SEGEND TO LAST SEQUENCE NUMBER
25 IF TOKEN NOT EOS
26 THEN
27 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
28 INCREMENT TO NEXT TOKEN
29 IF TOKEN IS AN INTEGER
30 THEN
31 STORE STARTING RANGE NUMBER
32 SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
33 EXIT TO :ERROR3: IF NOT FOUND
34 INCREMENT TO NEXT TOKEN
35 ENDIF
36 IF TOKEN NOT EOS
37 THEN
38 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
39 INCREMENT TO NEXT TOKEN
40 EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
41 ERREXIT TO :ERROR5: IF 'OLDING SEG # < BEGINNING SEG #
42 SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
43 EXIT TO :ERROR3: IF NOT FOUND
44 INCREMENT TO NEXT TOKEN
45 EXIT TO :ERROR2: IF TOKEN NO. :OS
46 ENDIF
47 ENSIF
48 SET SEAPTR TO SEGA
49 ENDIF
50 ENSIF
51 INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
52 EXIT TO :ERROR1: IF INITIALIZATION FAILS
53 1 EXIT KEINF
54 2 :ERROR1: INITIALIZATION FAILURE TERMINATION
55 2 :ERROR2: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
551  2  :ERROR3: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552  2  :ERROR4: SET SUBSTA TO DIRECTIVE LEVEL
553  2  IF ERROR WAS NO AWA SPACE THEN
554  2  EXIT WITH NO AWA SPACE ERROR
555  2  ELSE
556  2  EXIT WITH SEQUENCE TABLE NOT FOUND ERROR
557  2  ENDIF
558  2  :ERROR5: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
559  2  :ERROR13: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH NUMBER NOT FOUND
560  1  END XEINX
FORTRAN CALLING PROCEDURE:

CALL XELDS (SEGNM) TO CALL SEGMENT FROM MAIN
CALL XERTM TO RETURN TO MAIN PROGRAM

YELDS ALLOWS A MAIN PROGRAM TO "CALL" A SEGMENT AND
THE ENTRY POINT XERTM 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) XELDS CONTAINS 2 ENTRY POINTS: XELDS AND XERTM
2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
3) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XERTM"
   FOLLOWED BY A CALL TO MAIN WHICH IS NOT EXECUTED
4) WHEN LOADER IS RUN, THE FOURTH PARAMETER MUST BE 1
   INDICATING TO LOAD MAIN PLUS SEGMENTS

BEGIN XELDS

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
END XELDS
608 1  CD0      COMMON CALLING PROCEDURE
609 1  CD0      CALL XESCM
610 1  CD0      
611 1  CD0      C******
612 1  CD0      CDT
613 1  CD0      XESCM IS CALLED BY XEXEC AFTER COMPLETION OF ALL AUTOMATIC AND
614 1  CD0      SEMI-AUTOMATIC EXECUTIONS TO ASSURE THAT ALL DO AND SCAN PROCESSOR
615 1  CD0      CONTROL DATA IS PURGED FROM THE ANA AND ALL INCOMPLETE DATA BOX
616 1  CD0      FILES ARE PURGED FROM THE SYSTEM.
617 1  CD0      
618 1  CD0      
619 1  CD0      C******
620 1  CD0      INPUT
621 1  CD0      
622 1  CD0      XE COMMON - CARTNG
623 1  CD0      ANA - EDOSTK, BSCNTB (SEE DO AND SCAN PROCESSORS
624 1  CD0      OUTPUT
625 1  CD0      C******
626 1  CD0      
627 1  CD0      XE COMMON - REBUF, REPTR
628 1  CD0      XB COMMON - SCRATCH
629 1  CD0      
630 1  CD0      
631 1  CD0      EXTERNAL ROUTINES
632 1  CD0      CLOSE, EXEC, PURGE, XREQ, XRI6, XRMV, XRUNG
633 1  CD0      
634 1  CD0      
635 1  CD0      
636 1  CD0      
637 1  CD0      EEGIN XESCM
638 1  CD0      
639 2  CD0      BUILD REQUESTS TO DELETE EDOSTK AND RETRIEVE BSCNTB THEN DELETE IT
640 2  CD0      CALL XREQ TO ATTEMPT REQUESTS
641 2  CD0      ISSUE MESSAGE XE18 FOR SUCCESSFUL DELETES
642 2  CD0      IF XREQ COMPLETED REQUESTS, I.E., BSCNTB EXISTED, THEN
643 3  CD0      DO FOR EACH SCAN CONTROL ENTRY IN BSCNTB
644 4  CD0      CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN
645 4  CD0      CALL PURGE TO PURGE THE FILE
646 4  CD0      IF PURGE RETURNED AN ERROR, THEN
647 5  CD0      CALL XRMG TO DISPLAY WARNING MESSAGE
648 4  CD0      ENDF
649 4  CD0      BUILD REQUEST TO DELETE DATA BOX FROM TOC
650 3  CD0      ENDDO
651 2  CD0      CALL XREQ TO DELETE DATA BOXES FROM ANA TOC
652 2  CD0      ENDF
653 1  END XESCM
655 1 CD9 FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
656 1 CD1 ONE PROGRAM PER SIGNED-ON USER
657 1 CD1
658 1 CD1
659 1 C******
660 1 CD2 INPUT
661 1 CD2 SCHEDULING PARAMETERS - LU, CLASNO, QUAL, FLAGS (SEE XE COMMON)
662 1 CD2
663 1 C******
664 1 CD3 OUTPUT
665 1 CD5 COMMON XE - CONBUF, COMPTR, MASSTA, SUBSTA, PLUS XEINE
666 1 CD5 INITIALIZATION
667 1 CD5 COMMON XB - INITIALIZATIONS FROM XEIND, XEINI, XEINS, XEINX
668 1 CD3
669 1 C******
670 1 CD5 NOTES
671 1 CD5 USES BMPAR, XDCLD, XDCLF, XELDS, XINTE, XRCPR, XRMSG, XSERE,
672 1 CD5 XTCOM, XTCMT
673 1 CD5
674 1 CD5 THE CALLS TO XELDS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
675 1 CD5 XECAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
676 1 CD5
677 1 CD5 THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CONTROLS OCCURS
678 1 CD5 BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XTCMT FROM
679 1 CD5 CALLING XSERE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
680 1 CD5 MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
681 1 CD5 WHEN MASSTA IS SET TO THE DIRECTIVE LEVEL.
682 1 CD5
683 1 C******
BEGIN EXEC
RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASMO, QUAL & FLAGS
CALL XEINP TO INITIALIZE GLOBAL COMMON
DO FOREVER -- TERMINATES INSIDE HANDLER FOR ZOFF
CALL STMCM FOR INPUT OF DIRECTIVE
IF ERROR OR NOT A VALID DIRECTIVE NAME THEN ISSUE MESSAGE EDG ELSE IF NAME IS INT8 THEN SET STATES TO INT8 LEVEL CALL XEINP TO INITIALIZE DYNAMIC COMMON EXIT TO :RESET: IF ERROR CALL XINTE TO EDIT TABLE ELSE IF NAME IS SER8 THEN SET STATES TO SER8 LEVEL CALL XEINP TO INITIALIZE DYNAMIC COMMON EXIT TO :RESET: IF ERROR CALL XSER8 TO EDIT TABLE ELSE IF NAME IS FOR SOME EXECUTION CONTROL OPTION THEN SET STATES TO APPROPRIATE EXECUTION CONTROL MODE DO UNTIL MASTP IS AT DIRECTIVE LEVEL CALL XEINP TO INITIALIZE DYNAMIC COMMON EXIT TO :RESET: IF ERROR CALL XEXEC TO PERFORM EXECUTIONS IF SUBSTA IS SET TO SER8 LEVEL THEN CALL XEINP TO REINITIALIZE DYNAMIC COMMON EXIT TO :RESET: IF ERROR CALL XSER8 TO SUPPORT EXECUTION CONTROL ENDIF IF SUBSTA IS SET TO INT8 LEVEL THEN CALL XEINP TO REINITIALIZE DYNAMIC COMMON EXIT TO :RESET: IF ERROR CALL XINTE TO SUPPORT EXECUTION CONTROL ENDIF IF EXECUTION MODE WAS SEMI OR AUTO THEN CALL XESCEN TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES ENDIF ELSE CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL? ENDIF ENDIF :RESET: IF SUBSTA IS NOT DIRECTIVE LEVEL THEN CALL XEINP TO REINITIALIZE DYNAMIC COMMON ENDIF
1  CALL XINTE

2  CALL XINTE

3  CALL XINTE

4  CALL XINTE

5  CALL XINTE

6  CALL XINTE

7  CALL XINTE

8  CALL XINTE

9  CALL XINTE

10  CALL XINTE

11  CALL XINTE

12  CALL XINTE

13  CALL XINTE

14  CALL XINTE

15  CALL XINTE

16  CALL XINTE

17  CALL XINTE

18  CALL XINTE

19  CALL XINTE

20  CALL XINTE

21  CALL XINTE

22  CALL XINTE

23  CALL XINTE

24  CALL XINTE

25  CALL XINTE

26  CALL XINTE

27  CALL XINTE

28  CALL XINTE

29  CALL XINTE

30  CALL XINTE

31  CALL XINTE

32  CALL XINTE

33  CALL XINTE

34  CALL XINTE

35  CALL XINTE

36  CALL XINTE

37  CALL XINTE

38  CALL XINTE
FORTRAN CALLING PROCEDURE

CALL XIMIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

COMMON XE -
COMBUF = TERMINAL COMMUNICATIONS OUTPUT BUFFER
COMPT = 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
MARG = INDEX TO START OF SHORT PROMPT AREA OF MBUF
NEWTAB = NAME OF INTERFACE TABLE TO BE CREATED BY THIS EDIT. INPUT = 0 IF XINTER IS CALLED BY FDS LIBRARY MAINTENANCE PROGRAM.
NUMARG = NUMBER OF ARGUMENTS IN THIS INTERFACE TABLE
WKBUF = WORKING BUFFER CONTAINING INTERFACE TABLE AND SHORT PROMPTS WITH LITERAL IN EXPANDED FORM.

OUTPUT

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

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

USES ROUTINES
EXEC XREQ XILIT XIPRM XINPT
1 CD$  XRMOV
2 CD$  XRMSE
3 CD$  XTCOM
4 E--------
5 # * INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'INT?' DIRECTIVE
6 # OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
7 # PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
8 # TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
9 # OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
10 # NOTE: ALL INITIALIZATION, INCLUDING WKBUF (OLD INTERFACE TABLE),
11 # HAS BEEN PERFORMED BY XEINI.
12 #
13 1 BEGIN XEINI
14 2 IF OP (NO. OF PARAMETERS) > 0, THEN
15 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
16 3 * PRMTND = 1 => CREATE R MODE
17 3 * PRMTND = 3 => CREATE A MODE
18 3 * PRMTND = 4 => CREATE CONTINUE MODE
19 3 * PRMTND = 5 => MODIFY MODE
20 3 DO UNTIL 'EXIT' OR '..' IS ENTERED
21 4 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTND, SIZE, TYPE, AND STATUS
22 4 OF NEXT ARGUMENT
23 4 CALL XTCOM TO PROMPT USER AND RETURN PARSED INPUT
24 4 IF '..' WAS NOT ENTERED, THEN
25 5 IF \"\" WAS ENTERED, THEN
26 6 SET PRMTND TO 5
27 7 ELSE
28 8 IF NOTHING WAS ENTERED (I.E. TOKEN IS CD$), THEN
29 9 INCREMENT TO NEXT ARGUMENT
30 10 ELSE
31 11 CALL XIPMT TO PROCESS THE USER'S INPUT
32 12 ENDIF
33 13 ENDIF
34 14 ENDDO
35 15 IF A '..' WAS ENTERED, THEN
36 16 SET RETURN CODE INDICATING \" (I.E. MASSTA = 0)
37 17 ELSE
38 18 COMpress THE LITERAL LIST AREA
39 19 ENDIF
40 20 ENDDO
41 21 STORE INTERFACE TABLE AS NEWNAME
42 22 IF STORE INTO AWS FAILED, THEN
43 23 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
44 24 ELSE
45 25 SET GOOD RETURN CODE
46 26 ENDF
47 27 ENDF
48 28 ENDF
49 1 ENDF XEINI
152 1 CDO       FORTRAN CALLING PROCEDURE
153 1 CDO
154 1 CDO       CALL XIPRM
155 1 CDO
156 1 C*******
157 1 CDO1      PROMPT CONSTRUCTOR FOR INTERFACE TABLE EDITOR
158 1 CDO1
159 1 CDO1
160 1 C*******
161 1 CDO2      INPUT
162 1 CDO2
163 1 CDO2
164 1 CDO2      COMMON XB -
165 1 CDO2      ARGNO, CFLAG, DFLAG, IARG, IFLAG,
166 1 CDO2      ISIZE, ITYPE, LHEFF, LITSIZ, MARG,
167 1 CDO2      MODBIT, NUMARG, PRMTMD, WKBUF
168 1 CDO2
169 1 C*******
170 1 CDO3      OUTPUT
171 1 CDO3
172 1 CDO3
173 1 CDO3      COMMON XB -
174 1 CDO3      ARGNO, ISUB, NOSAV, PRMLEN,
175 1 CDO3      PRMTMD, PROMPT
176 1 CDO3
177 1 C*******
178 1 CDS5      NOTES
179 1 CDS5
180 1 CDS5
181 1 CDS5      USES ROUTINES
182 1 CDS5      CMXMD
183 1 CDS5      XXEXT
184 1 CDS5      XILSN
185 1 CDS5      XRMOV
186 1 CDS5      XRMVB
187 1 CDS5      XRPCK
188 1 CDS5      XRUPK
189 1 CDS5
190 1 C*******
1 * CONSTRUCT PROMPT TO BE ISSUED
2 BEGIN XIPMN
3 DO UNTIL A PROMPT IS CONSTRUCTED
4 IF PRTMD = 5, THEN
5 CONSTRUCT A ":" PROMPT
6 ELSE
7 IF PRTMD = 4 (CONTINUE MODE), OR
8 PRTMD = 6 (CONTINUE HERE MODE), THEN
9 IF ARGNO IS A SCALAR, THEN
10 CONSTRUCT PROMPT AS \"\arg\:=\" OR \"\arg\:=\";
11 ELSE
12 IF PRTMD NOT = 6, THEN
13 COMPUTE ISUB AS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LASTE)
14 ELSE
15 COMPUTE CURRENT ELEMENT NO. (SUBSCRIPT) FROM CURRENT INDEX
16 ENDIF
17 IF THERE ARE NO EMPTY SLOTS BEYOND LASTE, THEN
18 SET PRTMD TO 4
19 EXIT THE CONTINUE MODE
20 ELSE
21 IF ARGNO IS DOUBLE SUBSCRIPTED PARAMETER, THEN
22 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
23 CONSTRUCT PROMPT AS \"\arg\:=\{i,j\}\:";
24 ELSE
25 CONSTRUCT PROMPT AS \"\arg\:=\{\arg\}\:";
26 ENDIF
27 ENDIF
28 ELSE
29 IF ARGNO = BP, THEN
30 SET PRTMD TO 5
31 ELSE
32 INCREMENT ARGNO TO NEXT PARAMETER
33 SET IFLAG TO 1, 2, OR 3 INDICATING I, O, OR IO
34 SET LAST ENTERED INDICATOR (LASTE) TO O
35 IF PRTMD = 3, THEN
36 IF SOME DATA VALUE(S) OR PARM NAME EXISTS FOR ARGNO, THEN
37 CALL ILSD TO LIST DATA FOR THIS ARGUMENT
38 ENDIF
39 CONSTRUCT PROMPT AS \"\arg\:=\:" OR \"\arg\:=\:";
40 ELSE
41 IF ARGNO IS MARKED INCOMPLETE, THEN
42 IF A PARTIAL LITERAL LIST EXISTS, OR
43 THIS ARGUMENT IS A SCALAR, THEN
44 COMPUTE ISUB AS FIRST EMPTY ELEMENT
45 IF DOUBLE SUBSCRIPTED PARAMETER, THEN
46 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
47 CONSTRUCT PROMPT \"\arg\:=\{i,j\}\:";
48 ELSE
49 CONSTRUCT PROMPT \"\arg\:=\{\arg\}\:";
50 ENDIF
51 ELSE
52 CONSTRUCT PROMPT AS \"\arg\:=\:" OR \"\arg\:=\:";
53 ENDIF
54 ENDIF
55 ENDIF
56 ENDIF
57 END
FOR calling procedure

CALL XPMT

** PROMPT DIRECTIVE PROCESSOR **

CALL

** INPUT **

COMMON XE - COMBUF, COMPTA, TOKENS

** OUTPUT **

COMMON XB - ARGNO, PRMTND

** **

* XPMT PROCEDURE IS THE PROMPT DIRECTIVE *

BEGIN XPMT

IF TOKEN IS NOT COMMA :ERROR2:

POSITION TO NEXT TOKEN

ERRENT IF TOKEN IS NOT NAME :ERROR2:

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

IF NAME IS 'A', THEN

SET PRMTND TO 1

ELSE IF NAME IS 'B', THEN

SET PRMTND TO 3

ELSE

ERRENT :ERROR2:

ENDIF

END IF

SET ARGNO TO 0

EXIT TO :RETURN:

:ERROR2: CALL XMSG FOR 'INVALID SYNTAX'

:RETURN:

END XPMT
FORTRAN CALLING PROCEEDURE
CALL XILS

LIST DIRECTIVE PROCESSOR (WITHIN INTERFACE TABLE EDITOR)

INPUT
COMMON IE - COMBUF, COMPTR, TOKENS
COMMON IE - CFLAG, LITPTR, MARG
COMMON IE - MARG, VERSON, WBUF

OUTPUT
COMMON IE - ARGNO

NOTES

USES ROUTINES
EXEC
XICH
XICT
XILS
XICPR
XRLG
XRMD
XRMSG
XRSET
337 1 * XILST PROCESSES THE LIST DIRECTIVE
338 2 BEGIN XILST
339 3 IF TOKEN IS "=" THEN
340 4 POSITION TO NEXT TOKEN
341 5 ERREXIT IF TOKEN IS NOT NAME ; ERR02:
342 6 ERREXIT IF NAME IS NOT 'C', 'Y', OR 'A' ; ERR02:
343 7 SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)
344 8 POSITION TO NEXT TOKEN
345 9 ELSE
346 10 SET MODEFG TO 2
347 11 ENDF
348 12 IF TOKEN IS EOS, THEN
349 13 WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
350 14 AND STATUS
351 15 DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
352 16 IF MODEFG = 1 OR MODEFG = 3, THEN
353 17 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
354 18 ENDF
355 19 IF MODEFG = 2 OR MODEFG = 3, THEN
356 20 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
357 21 ENDF
358 22 ENDDO
359 23 ELSE
360 24 DO UNTIL EOS IS REACHED
361 25 ERREXIT IF TOKEN IS NOT COMMA ; ERR02:
362 26 ERREXIT IF NEXT TOKEN IS NOT NAME : ERR02:
363 27 SET ARGNO TO 1
364 28 START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
365 29 EXIT IF NAME = ARGNO'S NAME IN PROPTT TABLE
366 30 IF MODEFG = 1 OR MODEFG = 3, THEN
367 31 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
368 32 ENDF
369 33 IF MODEFG = 2 OR MODEFG = 3, THEN
370 34 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
371 35 ENDF
372 36 ENDOLOOP
373 37 PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
374 38 ENDOSEARCH
375 39 INCREMENT TO NEXT TOKEN
376 40 ENDDO
377 41 ENDF
378 42 EXIT TO :RETURN:
379 43 :ERR02: CALL XREMSG TO WRITE 'INVALID SYNTAX'
380 44 :RETURN:
381 45 END XILST
1 CD0   FORTRAN CALLING PROCEDURE
2 CD0
3 CD0   CALL XISUB
4 CD0
5 CD0   ********
6 CD0
7 CD0   EFFECTIVE SUBSCRIPT CALCULATION ROUTINE
8 CD0
9 CD0   ********
10 CD0   INPUT
11 CD0
12 CD0   COMMON XE - COMBUF, COMPTR, TOKENS
13 CD0
14 CD0   COMMON XE - IDIM, ISIZE, LENEFF
15 CD0
16 CD0   ********
17 CD0   OUTPUT
18 CD0
19 CD0   COMMON XE - IRET, ISUB
20 CD0
21 CD0   ********
22 CD0   USES ROUTINES
23 CD0
24 CD0   CD0   XRMSG
25 CD0
26 CD0   ********
27 CD0   * XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
28 CD0   * THE INPUT SUBSCRIPT
29 CD0   * BEGIN XISUB
30 CD0
31 CD0   INCREMENT TO NEXT TOKEN
32 CD0
33 CD0   ERREK IF TOKEN IS NOT INTEGER VALUE :ERR14:
34 CD0
35 CD0   IF IDIM FOR THIS ARGUMENT > 0, THEN
36 CD0
37 CD0   ERREK IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
38 CD0   INCREMENT TO NEXT TOKEN
39 CD0   ERREK IF TOKEN IS NOT COMMA :ERR15:
40 CD0   INCREMENT TO NEXT TOKEN
41 CD0   ERREK IF TOKEN IS NOT INTEGER VALUE :ERR14:
42 CD0   CALCULATE ISUB AS (J-I)*IDIM+I
43 CD0   ELSE
44 CD0
45 CD0   SET ISUB TO INTEGER VALUE
46 CD0   ENDIF
47 CD0
48 CD0   ERREK IF ISUB > A**E :ERR16:
49 CD0
50 CD0   INCREMENT TO NEXT TOKEN
51 CD0   ERREK IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
52 CD0   1 EXIT TO :RETURN:
53 CD0
54 CD0   :ERR14: CALL XRMSG - 'INVALID SUBSCRIPT SYNTAX'
55 CD0
56 CD0   :ERR15: CALL XRMSG - 'DOUBLE SUBSCRIPTED - MUST SPECIFY BOTH'
57 CD0
58 CD0   :ERR16: CALL XRMSG - 'INVALID SUBSCRIPT VALUE'
59 CD0
60 CD0   2 :RETURN:
61 CD0   1 END XISUB
FORTRAN CALLING PROCEDURE

CALL XIDAT

LITERAL DATA PROCESSOR

INPUT

COMMON XE - COMBUF, COMPTN, TOKENS

COMMON XB - CFLAG, COMPLE, DFLAG, IARG, IARG4,
   ISIZE, ISIZE4, ITYPE, LNEFF, LIDSP,
   LITNUM, LITNUM4, MODIM, MODIM4, MODIM5,
   PNTHD, SFLAG, WKBLG, WKBUF

OUTPUT

COMMON XB - IRETC, ISUB, LIDSP, LITNUM, MODIM5,
   PNTHD, WKBUF

INTERNAL VARIABLES

COMMON XS -

IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS

LITERAL DATUM

NUMCEL = NUMBER OF CONSECUTIVE ELEMENTS TO BE

MARKED COMPLETED AS A RESULT OF THIS

LITERAL DATUM

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

DESCRIBING NESTED REPEAT GROUPS.

FOR EACH ENTRY IS 3 WORDS:

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

WORD 2 = AFTER REPEAT SYMBOL

WORD 3 = FLAG INDICATING WHETHER

REPEAT GROUP IS PARETENTICALLY

GROUPED

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

NOTES

USES ROUTINES

XISUB

XRART

XRMOV

XRMSG

REPRODUCED BY OF THE

ORLCA ON 10-25-67
1 CD5 XRNDB
2 CD5 XRNSET
3 CD5 XRNEND
4 CD5 XRS
5 CD5 XRN

**C********
1 * XKEEP PROCESSES THE INPUT LITERAL LIST
2 XKEEP DO UNTIL EOS IS REACHED, OR
3 XKEEP UNTIL AN ERROR OCCURS
4 XKEEP IF THIS IS A DATA ELEMENT, THEN
5 XKEEP (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
6 XKEEP IF ARG. TYPE (ITY) IS INTEGER, REAL, OR DOUBLE, THEN
7 XKEEP ERREXIT IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
8 XKEEP SET LENGTH TO BE MOVED (LENUM) TO EFFECTIVE LENGTH FOR DATA
9 XKEEP OF ARG'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
10 XKEEP ELSE
11 XKEEP IF ARG. TYPE IS FREE, THEN
12 XKEEP SET LENGTH TO BE MOVED (LENUM) TO EFFECTIVE LENGTH
13 XKEEP FOR DATA INPUT
14 XKEEP ELSE THIS MUST BE CHARACTER DATA BEING INPUT
15 XKEEP ERREXIT IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ERR10:
16 XKEEP INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
17 XKEEP COMPUTE NO. WORDS IN INPUT CHARACTER STRING
18 XKEEP ERREXIT IF NO. WORDS (LENUM) > EFFECTIVE LENGTH OF
19 XKEEP THIS ARGUMENT'S DATA (LENEFF) :ERR10:
20 XKEEP ENDIF
21 XKEEP ENDIF
22 XKEEP INCREMENT TOKEN POINTER TO THE DATA INPUT
23 XKEEP VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
24 XKEEP ARGUMENT FOR DATA INPUT (ISUB <= ISISIZE - LENUM + 1)
25 XKEEP ERREXIT IF INSUFFICIENT SPACE :ERR11:
26 XKEEP IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
27 XKEEP ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
28 XKEEP ENDIF
29 XKEEP MOVE DATA FROM INPUT COMMUNICATIONS BUFFER TO LITERAL AREA
30 XKEEP SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
31 XKEEP ARGUMENT, TO LENUM
32 XKEEP IF LENUM < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
33 XKEEP MOVE LENUM-LENEFF BLANKS INTO LITERAL AREA AS A FILL
34 XKEEP ENDIF
35 XKEEP INCREMENT TO NEXT TOKEN
36 XKEEP INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENUM
37 XKEEP SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
38 XKEEP ELSE, TOKEN INDICATES NON-DATA ELEMENT
39 XKEEP IF THIS IS A "(" , THEN
40 XKEEP CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
41 XKEEP BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITY) AND
42 XKEEP SECONDARY DIMENSION (IDIM)
43 XKEEP EXIT XKEEP IF ERROR (IRTEC < 0)
44 XKEEP ELSE
45 X6 XKEEP IF THIS IS AN "[", THEN
46 X9 XKEEP IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
48 X9 XKEEP CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
49 X9 XKEEP ELSE
50 X9 XKEEP MAKE ONE ELEMENT OF THIS ARGUMENT AT ISUB AS INCOMPLETE
51 X9 XKEEP INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BT EFFECTIVE
52 X9 XKEEP LENGTH OF ONE ELEMENT (LENEFF)
53 X9 XKEEP ENDIF
54 X9 XKEEP TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
55 X9 XKEEP INCREMENT TO NEXT TOKEN
ELSE
  IF TOKEN IS A REPEAT COUNT, THEN
    IF NEXT TOKEN IS '(', THEN
      SET PARENFLAG TO 1
      INCREMENT TO NEXT TOKEN
    ELSE
      SET PARENFLAG TO 0
    ENDIF
    SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
  ELSE
    ERREXIT (INVALID FIELD) :ERR07:
  ENDIF
  INCREMENT TO NEXT TOKEN
ENDIF
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
        ERREXIT IF TOKEN IS NOT A COMMA :ERR02:
      ENDIF
      INCREMENT TO NEXT TOKEN
    ENDIF
  ENDIF
ENDIF
ENDO
615 1 EXIT TO :RETURN:
616 2 :ERROR2: CALL XRMSG "INVALID SYNTAX"
617 2 :ERROR7: CALL XRMSG "ONLY DATA VALID TO RIGHT OF =
618 2 :ERROR10: CALL XRMSG "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
619 2 :RETURN:
620 1 END XIDAT
* XINPT PROCESSES THE USER'S INPUT TEXT

   1 BEGIXINPT
   2   IF PRTM = 5, THEN
   3   SET IFLAG OFF (=0)
   4   IF TOKEN IS NOT A NAME :ERROR:
   5   SAVE NAME AND POSITION TO NEXT TOKEN
   6   IF TOKEN IS '"', THEN
   7       POSITION TO NEXT TOKEN
   8       IF TOKEN IS '"', THEN
   9       SET IFLAG TO 10
  10       POSITION TO NEXT TOKEN
  11 ELSE
  12     SET IFLAG TO 1
  13 ENDIF
  14 ELSE
  15     IF TOKEN IS '"', THEN
  16     SET IFLAG TO 0
  17 ENDIF
  18 ENDIF
  19 IF IFLAG NOT SET, THEN
  20   CASE NAME (:EXIT; :PROMPT; :LIST;)
  21     ERREXIT IF ANOTHER TOKEN FALLS :ERROR:
  22     EXIT: SET IRET C SO THAT PROMPTING LOOP TERMINATES
  23     PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE
  24     LIST: CALL XLIST TO PROCESS LIST DIRECTIVE
  25   END CASE
  26 ENDIF
  27 START SEARCH UNTIL IF ENTRIES
  28 Exit IF NAME FOUND IN PROMPT TABLE
  29 SET ARGNO TO ENTRY NO.
  30 SET ISUB TO 1
  31 OR ELSE
  32 INCREMENT TO NEXT PASSWORD TABLE ENTRIES
  33 END LCMP
  34 ERRERE: ERT10:
  35 END SEARCH
  36 ERRERE: IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR:
  37 ENDIF
  38 IF NEXT TOKEN IS A NAME, THEN
  39 CALL XIPAR TO PROCESS A PARAMETER FIELD
  40 ELSE
  41 ERRERE: IFLAG IS NOT I ("=") :ERROR:
  42 CALL XIDAT TO PROCESS DATA LIST
  43 ENDIF
  44 EXIT XINPT
  45 EXIT TO :RETURN:
  46 :ERROR2: CALL XMRS "INVALID SYNTAX"
  47 :ERROR3: CALL XMRS "MUST USE PARAMETER NAME TO RIGHT OF & OR =""
1   C00  FORTRAN CALLING SUBROUTINE
2   C00  CALL XIPAR
3   C00  
4   C00  ************
5   C01  PROCESS AN INPUT PARAMETER NAME AND ANY ASSOCIATED SUBSCRIPT
6   C01  FIELD(S)
7   C01  
8   C01  ************
9   C02  INPUT
10  C02  COMMON XE - XBUF, COMPTR, TOKENS
11  C02  COMMON XE - DFLAG, IARG, IAREA, ICLASS,
12  C02  LITDSP, SFLAG, WKBLEN, WKBUF
13  C02  
14  C02  ************
15  C03  OUTPUT
16  C03  COMMON XE - XRET, LITDWN, WKBUF
17  C03  
18  C03  ************
19  C05  NOTES
20  C05 USES ROUTINES
21  C05  XRMCT
22  C05  XRMSET
23  C05  XRMSEG
24  C05  
25  C05  ************
1 * XIPAR PROCFSSES A USER SPECIFIED PARAMETER FIELD
2
3 BEGIN XIPAR
4 1 IF A NAME IS SPECIFIED, THEN
5 INCPEMENT TO NEXT TOKEN
6 1 IF TOKEN IS '(' THEN
7 PROCESS I AND J SUBSCRIPTS
8 ERREXIT IF INVALDP SUBSCRIPTING :ERR14:
9 1 IF DOUBLY SUBSCRIPTED, THEN
10 5 SET S-FLAG IN ARGNO'S SPECS. FIELD
11 5 SET LITOSP IN ARGNO'S SPECS. TO NEXT LITERAL AREA SPACE (LITDOWN)
12 5 PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT
13 ELSE
14 5 SET LITOSP IN ARGNO'S SPECS TO ISUB
15 ENDIF
16 ELSE
17 5 SET LITOSP IN ARGNO'S SPECS TO 0
18 ENDIF
19 5 ERREXIT IF ORDE HAS MORE THAN A CHARACTERS :ERR18:
20 ERREXIT IF EXTRAMOUS FIELD INPUT :ERR02:
21 5 TURN OF D-FLAG (SUME LITERAL DATA) IN ARGNO'S SPECS.
22 5 SET PARAMETER NAME INTO ARGUMENT'S CHARACTERISTICS
23 5 SET COMPLETE (AND S) FLAG IN ARGUMENT'S CHARACTERISTICS
24 5 IF ALL ARGUMENTS ARE COMPLETE, THEN
25 5 SET INTERFACE TABLE COMPLETE FLAG
26 ENDIF
27 2 ELSE "MUST BE A & INPUT "
28 ERREXIT IF NOT AN AMPERSAND (&) INPUT :ERR02:
29 CLEAR PARAMETER NAME IN ARGMENT'S CHARACTERISTICS
30 2 SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
31 ENDIF
32 1 EXIT TO :RETURN:
33 2 :ERR02: CALL XRESD - "INVALID SYNTAX"
34 2 :ERR14: CALL XRESD - "INVALID SUBSCRIPT SYNTAX"
35 2 :ERR18: CALL XRESD - "INVALID ORDE NAME"
36 2 :RETURN:
37 1 END XIPAR
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<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</td>
</tr>
<tr>
<td>791</td>
<td>1 CD0</td>
</tr>
<tr>
<td>792</td>
<td>1 CD0</td>
</tr>
<tr>
<td>793</td>
<td>1 CD0</td>
</tr>
<tr>
<td>794</td>
<td>1 CD0</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</td>
</tr>
<tr>
<td>799</td>
<td>1 CD0</td>
</tr>
<tr>
<td>800</td>
<td>1 CD0</td>
</tr>
<tr>
<td>801</td>
<td>1 CD0</td>
</tr>
<tr>
<td>802</td>
<td>1 CD0</td>
</tr>
<tr>
<td>803</td>
<td>1 CD0</td>
</tr>
<tr>
<td>804</td>
<td>1 CD0</td>
</tr>
<tr>
<td>805</td>
<td>1 CD0</td>
</tr>
<tr>
<td>806</td>
<td>1 CD0</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</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</td>
</tr>
<tr>
<td>814</td>
<td>1 CD0</td>
</tr>
<tr>
<td>815</td>
<td>1 CD0</td>
</tr>
<tr>
<td>816</td>
<td>1 CD0</td>
</tr>
<tr>
<td>817</td>
<td>1 CD5</td>
</tr>
<tr>
<td>818</td>
<td>1 CD5</td>
</tr>
<tr>
<td>819</td>
<td>1 CD5</td>
</tr>
<tr>
<td>820</td>
<td>1 CD5</td>
</tr>
<tr>
<td>821</td>
<td>1 CD5</td>
</tr>
<tr>
<td>822</td>
<td>1 CD5</td>
</tr>
<tr>
<td>823</td>
<td>1 CD5</td>
</tr>
<tr>
<td>824</td>
<td>1 CD5</td>
</tr>
<tr>
<td>825</td>
<td>1 CD5</td>
</tr>
<tr>
<td>826</td>
<td>1 CD5</td>
</tr>
<tr>
<td>827</td>
<td>1 CD5</td>
</tr>
<tr>
<td>828</td>
<td>1 CD5</td>
</tr>
<tr>
<td>829</td>
<td>1 CD5</td>
</tr>
<tr>
<td>830</td>
<td>1 CD5</td>
</tr>
<tr>
<td>831</td>
<td>1 CD5</td>
</tr>
</tbody>
</table>

FORTRAN CALLING PROCEDURE

CALL XILSD

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

LIST DATA OR PARAMETER VALUES FOR ONE ARGUMENT

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

INPUT

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

COMMON XE - .J

COMMON XE - W.A., Oflag, Iarg, Idir,

AVIS, IDflag, Isub, INTP,E,

Lennf, LITOSP, LITSSF, MARD,

M3X8TH, M08TH, SFLAG, WBUF

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

INTERNAL VARIABLES

COMMON XS - BUFFER - LINE TO BE OUTPUT

BUFFV = INDEX INTO BUFFER FOR NEXT ASCII DATA

KQETRM - RETURN INDICATOR FOR INTERNAL ROUTINES

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

NOTES

USES ROUTINES

EXEC

XRETA

XRI6

XRMOW

XRMDB

XRR06

XRPCK

XRRM6

XRRMP

************
9 + XILSD WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
10. IS RETURNED AS A PROMPT.
20. BEGIN XILSD
30. SET ARGUMENT NAME INTO BUFFER
31. USE T0-F I G L A G TO DETERMINE WHICH OF "3", "n", OR "n3"
32. WILL GO INTO THE PRINT BUFFER
33. IF D-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
34. IF A PARAMETER NAME IS SPECIFIED, THEN
35. PUT PARAMETER NAME INTO BUFFER
36. IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
37. COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
38. PUT SUBSCRIPT INTO BUFFER
39. ELSE
40. IF LITDSP OF ARGUMENT IS > 0, THEN
41. COMPUTE AND CONVERT THIS SUBSCRIPT
42. PUT SUBSCRIPT INTO BUFFER
43. ENDIF
44. ENDIF
45. WRITE OUT THE PRINT BUFFER BUILT
46. ENDIF
47. ELSE
48. LOCATE LITERAL LIST AND MASK
49. IF SYMBOLIC STRING, THEN
50. CALL FILES TO PRINT SYMBOLIC STRING
51. ELSE
52. DO UNTIL ALL ELEMENTS PROCESSED
53. DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
54. UNL ALL ELEMENTS PROCESSED
55. COMPUTE AND CONVERT THE SUBSCRIPT
56. IF MASK FOR ELEMENT INDICATES NO DATA, THEN
57. PUT "3" INTO BUFFER
58. ELSE
59. CONVERT THE DATA USING XR06, XR014, OR XR16
60. PUT DATA AND "3" INTO BUFFER
61. ENDIF
62. IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
63. REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
64. ENDIF
65. WRITE OUT THE PRINT BUFFER BUILT
66. ENDIF
67. ENDIF
68. ENDIF
69. END XILSD
FORTRAN CALLING PROCEDURE

CALL XILSS

XILSS IS CALLED BY XILSC TO LIST SYMBOLIC STRING DATA

INPUT

COMMON XE - LU
COMMON XD - DEBUG, LISTLY, WKRUF
COMMON XS - BUFFER = PRINT LINE BUFFER ALREADY INITIALIZED WITH NAME =
BUFPT = INDEX INTO BUFFER OF NEXT POSITIVE;
DATPT = INDEX INTO WKBUF OF SYMBOLIC STRING DATA

OUTPUT

COMMON XS - BUFFER, BUFPT, DATPT

C********

C TAPL OF THE
<table>
<thead>
<tr>
<th>ENTRY</th>
<th>TOKEN</th>
<th>WORD 1 (SIZE)</th>
<th>WORD 2 (FIELD)</th>
<th>WORD 3 (TOKSZ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>904</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>907</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>909</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>910</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>911</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>912</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>913</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>914</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>915</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>916</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>917</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>918</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>919</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>920</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>921</td>
<td>CD4</td>
<td>INTEGER</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>922</td>
<td>CD4</td>
<td>REAL</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>923</td>
<td>CD4</td>
<td>DOUBLE</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>924</td>
<td>CD4</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>925</td>
<td>CD4</td>
<td>NAME</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>926</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>927</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>928</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>929</td>
<td>CD4</td>
<td>CHAR. STR.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>930</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>931</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>932</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>933</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>934</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>935</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>936</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>937</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>938</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>939</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>940</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>941</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>942</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>943</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>944</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>945</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>946</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>947</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>948</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>949</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>951</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>952</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>953</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>954</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>956</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>957</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INTERNAL VARIABLES**

- **CTRL** = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING
- **ENTRY** IS 3 WORDS:
  - WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER
  - WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER
  - WORD 3 (TOKSZ) = NO. OF WORDS IN SYMBOLIC STRING

- 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
    - Word 3 (tksiz) = no. of words in symbolic string
BEGIN XILSS  
MOVE A 'CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER  
DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED  
EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32  
USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSZ)  
EXIT TO ERROR 2 IF FIELD = 0  
IF SIZE < 0, THEN  
SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMBOLIC STRING  
ENDIF  
IF TOKSZ < 0, THEN  
SET TOKSZ TO SIZE + 2  
ENDIF  
IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN  
CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE  
CLEAR PRINT BUFFER TO BLANKS  
ENDIF  
IF FIELD > 0, THEN  
MOVE FIELD INTO CURRENT PRINT BUFFER POSITION  
ELSE  
CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD  
:ONE: CALL XR16 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING AND PUT RESULTS INTO PRINT BUFFER  
:TW0: CALL XR14 WITH VALUE IN NEXT 2 WORDS OF SYMBOLIC STRING AND PUT RESULTS INTO PRINT BUFFER  
:THREE: CALL XR08 WITH VALUE IN NEXT 3 WORDS OF SYMBOLIC STRING AND PUT RESULTS INTO PRINT BUFFER  
:FOUR: MOVE THE NEXT 3 WORDS OF SYMBOLIC STRING INTO PRINT BUFFER  
:FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMBOLIC STRING INTO PRINT BUFFER  
:SIX: CALL XR16 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING AND PUT RESULTS INTO PRINT BUFFER FOLLOWED BY AN "R"  
:EXIT: PUT A 'CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER INDEX BY 1  
:INDEX BY 1  
:CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE  
:EXIT XILSS  
ENDCASE  
INCREMENT PRINT BUFFER INDEX BY SIZE
1015 CDR FORTRAN CALLING PROCEDURE
1016 CDR
1017 CDR CALL XCHR
1018 CDR
1019 CDR******
1020 CD1
1021 CD1 PRINT THE CHARACTERISTICS OF AN ARGUMENT
1022 CD1
1023 CD1******
1024 CD2
1025 CD2 INPUT
1026 CD2
1027 CD2 COMMON XE - LU
1028 CD2
1029 CD2 COMMON XD - ARGNO, IOFLAG, ISIIZE, ITYPE,
1030 CD2 LENEFF, NARG
1031 CD2
1032 CD2
1033 CD2******
1034 CD3
1035 CD3 NOTES
1036 CD3
1037 CD3 USES Routines
1038 CD3
1039 CD3
1040 CD3 EXEC
1041 CD3
1042 CD3 XRNQ
1043 CD3 XRPCK
1044 CD3
1045 CD3******
1046 CD4 * WRITE ARGUMENT CHARACTERISTICS
1047 CD4 BEGIN XCHR
1048 CD4 2 BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPT:, I/O TYPE AND
1049 CD4 2 DATA TYPE
1050 CD4 2 WRITE OUT THE PRINT BUFFER
1051 CD4 END XCHR
1053 1 CD0  FORTRAN CALLING PROCEDURE
1054 1 CD0
1055 1 CD0
1056 1 CD0  CALL XIEXT
1057 1 CD0
1058 1 CD0
1059 1 CD0  ********
1060 1 CD1  EXTRACT VARIOUS FIELDS OF A ARGUMENT'S CHARACTERISTICS
1061 1 CD1  AND PUT VALUES INTO COMMON
1062 1 CD1
1063 1 CD1  ********
1064 1 CD2  INPUT
1065 1 CD2
1066 1 CD2
1067 1 CD2  COMMON XB - ARGNO, ISIZE, WBUF
1068 1 CD2
1069 1 CD2
1070 1 CD2  ********
1071 1 CD3
1072 1 CD3  OUTPUT
1073 1 CD3
1074 1 CD3  COMMON XB - CFLAG, DFLAG, IARG, IARGA,
1075 1 CD3  ICLASS, IDIM, IFLAG, ISIZE,
1076 1 CD3  ISUB, ITYPE, LENEFF, LTIDSP,
1077 1 CD3  LITSZ, MDIXTM, MOBITM, SFLAG
1078 1 CD3
1079 1 CD3  ********
1080 1 CD5
1081 1 CD5  NOTES
1082 1 CD5
1083 1 CD5  USES ROUTINES
1084 1 CD5
1085 1 CD5  XAM
1086 1 CD5  XREXT
1087 1 CD5
1088 1 CD5  ********
1089 1 CD5  * EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS
1090 1 CD5  * ARGUMENT
1091 1 CD5  BEGIN XIEXT
1092 1 CD5  USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
1093 1 CD5  IN THE WORKING BUFFER
1094 1 CD5  EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
1095 1 CD5  END XIEXT
1145 1 PACK LITERAL AREA INTO FORMAT FOR STORAGE OF INTERFACE TABLE
1146 2 BEGIN XILIT
1147 3 DO UNTIL ALL LITERAL AREAS PROCESSED
1148 4 DO UNTIL ALL ARGUMENTS SEARCHED
1149 5 IF THIS LITERAL ENTRY BELONGS TO THIS ARGUMENT, THEN
1150 6 IF ALL ELEMENTS OF THIS ARGUMENT ARE COMPLETE, THEN
1151 7 MOVE ALL DATA FOR LITERAL ENTRY UP IN WORKING BUFFER
1152 8 ELSE
1153 9 DO UNTIL ALL BITS OF BIT MASK PROCESSED
1154 10 IF THE BIT IS ON, THEN
1155 11 MOVE CORRECT NUMBER OF WORDS (LENEFF) OF LITERAL
1156 12 UP IN THE WORKING BUFFER
1157 13 ENDEF
1158 14 ENDDO
1159 15 ENDF
1160 16 CALCULATE NEW DISPLACEMENT AND SET IN LITDSP
1161 17 ENDF
1162 18 ENDDO
1163 19 ENDDO
1164 20 END XILIT
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR02 : 292</td>
</tr>
<tr>
<td>ERR02 : 616</td>
</tr>
<tr>
<td>ERR02 : 782</td>
</tr>
<tr>
<td>ERR02 : 379</td>
</tr>
<tr>
<td>ERR02 : 709</td>
</tr>
<tr>
<td>ERR02 : 617</td>
</tr>
<tr>
<td>ERR02 : 710</td>
</tr>
<tr>
<td>ERR10 : 618</td>
</tr>
<tr>
<td>ERR10 : 783</td>
</tr>
<tr>
<td>ERR10 : 432</td>
</tr>
<tr>
<td>ERR15 : 433</td>
</tr>
<tr>
<td>ERR15 : 434</td>
</tr>
<tr>
<td>ERR15 : 784</td>
</tr>
<tr>
<td>EXIT : 1004</td>
</tr>
<tr>
<td>EXIT : 685</td>
</tr>
<tr>
<td>FIVE : 1001</td>
</tr>
<tr>
<td>FOUR : 1000</td>
</tr>
<tr>
<td>LIST : 687</td>
</tr>
<tr>
<td>OZONE : 994</td>
</tr>
<tr>
<td>PROMPT : 686</td>
</tr>
<tr>
<td>RETURN : 619</td>
</tr>
<tr>
<td>RETURN : 435</td>
</tr>
<tr>
<td>RETURN : 380</td>
</tr>
<tr>
<td>RETURN : 785</td>
</tr>
<tr>
<td>RETURN : 785</td>
</tr>
<tr>
<td>RETURN : 711</td>
</tr>
<tr>
<td>SIX : 1002</td>
</tr>
<tr>
<td>THREE : 998</td>
</tr>
<tr>
<td>TOLO : 996</td>
</tr>
<tr>
<td>XCHHR : 1047</td>
</tr>
<tr>
<td>XDAT : 502</td>
</tr>
<tr>
<td>XEXIT : 1091</td>
</tr>
<tr>
<td>XLIT : 1146</td>
</tr>
<tr>
<td>XLSO : 835</td>
</tr>
<tr>
<td>XLSST : 973</td>
</tr>
<tr>
<td>XLSST : 338</td>
</tr>
<tr>
<td>XIX : 113</td>
</tr>
<tr>
<td>XIMP : 664</td>
</tr>
<tr>
<td>XINTE : 35</td>
</tr>
<tr>
<td>XIPAR : 752</td>
</tr>
<tr>
<td>XIPMT : 276</td>
</tr>
<tr>
<td>XIPRM : 193</td>
</tr>
<tr>
<td>XISUB : 415</td>
</tr>
</tbody>
</table>

**NOTE:** The content is not legible due to the height of the characters.
**FILE: Routines and Subroutines Used:**

- EXEC, RMPAR
- FDS, FDF, XLDL, XLIM, XLMP, XLMSG, XLPMB, XLPFK, XTOMB, XRM, XRMG

**Variables:**
- PROM - (INTEGER, 31 WORDS) Prompt for Option
- TEMP - (INTEGER, 1 WORD) Temporary used for response
- TOKEZ - (INTEGER, 29 WORDS) The 29 tokens to be initialized into COMMON

**Inputs from Run Sequence:**
- LU - Logical unit number where user desires
- HIS inputs/outputs
- ISECU - Security code of FDS library files
- FLAGS - Debug flag: 0 - OFF, 4 - ON

**Offline Maintenance Program, XLMAN Creates, Deletes, and Modifies FDS Files:**

**RTE Run Procedure for Library Maintenance:**

1. CD**************
2. CD
3. CD0
4. CD0
5. CD0
6. CD0
7. CD0
8. CD0
9. CD0
10. CD1
11. CD1
12. CD1
13. CD1
14. CD1
15. CD1
16. CD2
17. CD2
18. CD2
19. CD2
20. CD2
21. CD2
22. CD2
23. CD2
24. CD4
25. CD4
26. CD4
27. CD4
28. CD4
29. CD4
30. CD4
31. CD4
32. CD4
33. CD4
34. CD5
35. CD5
36. CD5
37. CD5
38. CD5
39. CD5
40. CD5
41. CD5
42. CD5
43. CD5
44. CD5
45. CD5
46. CD5
47. CD5
48. CD5
49. CD5
50. CD5
51. CD5
52. CD5
53. CD**************
55 1 BEGIN XLMAN
56 2 CALL RMpar TO GET INPUT PARAMETERS
57 3 INITIALIZE COMMON TO ZERO
58 2 SET CLASS NUMBER TO ZERO
59 2 CALL EXEC TO GET A CLASS NUMBER
60 1 EXIT XLMAN IF SECURITY CODE IS NOT VALID
61 2 CALL XRMov TO INITIALIZE TOKENS IN COMMON
62 2 DO FOREVER
63 3 :PROMPT:
64 3 INITIALIZE MASTER AND SUBSTAKE FLAGS
65 3 CALL XTCM TO PROMPT FOR OPTION
66 3 IF XTCM RETURN CODE IS NOT ZERO OR
67 4 FIRST TOKEN IS NOT AN INTEGER OR
68 4 INTEGER > 7 THEN
69 4 CALL XRMsg TO WRITE INVALID RESPONSE
70 4 GO TO :PROMPT:
71 3 ENDF:
72 3 CASE INTEGER (:XLPRM::XLPRM::XLPRM::XLPRO::XLDEL::XLMOD::
73 4 * :XLMSG::XLDBF::XLDBF::XLDBF::EXIT::
74 4 :XLPRM:
75 4 SET NUMOR TO INTEGER
76 4 CALL XELs TO LOAD XLPRM TO CREATE SYSTEM PROMPT FILE
77 4 :XLPRO:
78 4 SET VALFLG TO SAY ORIGINAL XLPRO REQUEST
79 4 CALL XELs TO LOAD XLPRO TO ADD A PROCESSOR
80 4 DO UNTIL VALFLG SAYS EXIT (X)
81 5 CALL XELs TO LOAD XLINT TO ENTER DEFAULT VALUES
82 5 CALL XELs TO LOAD XLPRO TO COMPLETE PROCESSING
83 4 ENDDO
84 4 :XLDEL:
85 4 CALL XELs TO LOAD XLDEL TO DELETE A PROCESSOR
86 4 :XLMOD:
87 4 SET VALFLG TO SAY ORIGINAL XLMOD REQUEST
88 4 CALL XELs TO LOAD XLMOD TO MODIFY A PROCESSOR
89 4 DO UNTIL VALFLG SAYS EXIT (X)
90 5 CALL XELs TO LOAD XLINT TO ENTER DEFAULT VALUES
91 5 CALL XELs TO LOAD XLMOD TO COMPLETE PROCESSING
92 4 ENDDO
93 4 :XLMSG:
94 4 CALL XELs TO LOAD XLMSG TO ADD A MESSAGE
95 4 :XLDBF:
96 4 CALL XELs TO LOAD XLDBF TO HANDLE DATA BASE FILES
97 3 ENDCASE
98 2 ENDDO
99 2 :EXIT:
100 2 CALL EXEC TO RELEASE CLASS NUMBER
101 1 END XLMAN
FORTRAN CALLING PROCEDURE:

CALL XEIDS ('XLPRM')

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 - >XMLPRM
2 - >XMLPRM
3 - >XMLPRM

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
WITH WHICH TO CALL XMSG
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
PROM2, PROM3, PROM4 ARE ALL USER PROMPT ARRAYS

RTE FUNCTIONS AND SUBROUTINES USED:

KCVT, CLOSE, CREATE, PURGE, WAITF

FDS FUNCTIONS AND ROUTINES USED:

XMOV, XMSG, XCTOM

COMMON USED:

ERUVALENCE

= (X(3), ISECU, (X(7), NUMBR),

= (X(4), TCR, (X(145), COMBUF),

= (X(145), NOTOK, (X(145), COMBUF),

= (X(145), COM4, (X(14), IBUF),

= (X(128), MODIR)
160 1 BEGIN XLPRM
161 2 :PRM1:
162 2 CALL XICOM TO PROMPT USER FOR # DIRECTIVES
163 2 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
164 2 CALL CREAT TO CREATE PROMPT FILE
165 2 IF FILE ALREADY EXISTS THEN
166 2 CALL XICOM TO PROMPT USER FOR SCRATCH/CANCEL
167 2 EXIT XLPRM IF RESPONSE IS CANCEL
168 2 CALL PURGE TO PURGE PROMPT FILE
169 2 ERREXIT IF PURGE ERROR TO :FILEERR:
170 2 GO TO :PRM1:
171 2 ELSE (CREATE NEW FILE)
172 2 ERREXIT IF CREAT ERROR TO :FILEERR:
173 3 CALL XICOM TO PROMPT USER FOR LIST OF DIRECTIVES
174 3 ERREXIT IF LIST IS INCONSISTENT WITH # 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 ENDDO
179 3 5 CALL WRTIF TO WRITE LIST OF DIRECTIVES TO FILE
180 3 5 ERREXIT IF WRTIF ERROR TO :FILEERR:
181 3 5 DO FOR EACH DIRECTIVE
182 4 4 CALL XICOM TO PROMPT USER FOR DEFINITION
183 4 4 DO UNTIL EOS IS REACHED IN RESPONSE
184 5 5 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185 5 5 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186 5 5 MOVE RESPONSE INTO BUFFER
187 5 5 SET CONTROL CHARACTERS IN BUFFER
188 5 5 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
189 5 5 ENDDO
190 4 5 SET REMAINDER OF BUFFER TO NULL
191 4 4 CALL WRTIF TO WRITE DEFINITION
192 4 4 ERREXIT IF WRTIF ERROR TO :FILEERR:
193 4 4 ENDIF
194 3 3 CALL CLOSE TO CLOSE FILE
195 3 3 ERREXIT IF CLOSE ERROR TO :FILEERR:
196 3 3 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
197 2 2 ENDF
198 1 1 EXIT XLPRM
199 2 :ERR1: (ERROR IN RESPONSE)
200 2 CALL XMSG TO DISPLAY ERROR IN RESPONSE
201 2 RETURN TO RPROMPT USER FOR ANOTHER RESPONSE
202 2 :FILEERR: (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 XLPRM
CD**********

FORTRAN CALLING SEQUENCE:

CALL XELDS ('XLPRO')

**********

CD1

XLPRO WILL ADD A PROCESSOR TO THE LIBRARY DIRECTORy. IF

THE PROCESSOR HAS A DEFAULT INTERFACE TABLE, IT WILL ALSO

CREATE A PROMPT FILE AND INTERFACE TABLE FILE.

**********

CD2

INPUT FROM COMMON:

VALFLG - PROCESS CONTROL

D - ORIGINAL REQUEST TO ADD

1 - COMPLETE PROCESSING

**********

CD3

INTERNAL VARIABLES:

ABSTR - (INTEGER, 128 WORDS) CONTAINS THE ABSTRACT OF THE

PROCESSOR IN LATER BUILDS. PRESENTLY IS A NULL RECORD.

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

TO INTERFACE TABLE FILE

DIRECT - (INTEGER, 6 WORDS) 3 NAME ARRAY CONTAINING VALID

DIRECTIVES FOR THE INTERFACE TABLE EDITOR

NAME - (INTEGER, 3 WORDS) COMBINATION OF PROCESSOR

NAME AND VERSION NUMBER

PRMAM - (INTEGER, 3 WORDS) PROCESSOR NAME

PROMT - (INTEGERS) PROMPTS FOR USER TO BE PROMPTED WITH

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

CALCULATING SIZE

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

**********

CD5

RTE AND FNGR ROUTINES USED:

EXEC, IAND, KEYT, CREAT, OPEN,

READ, WRITE, CLOSE, PURGE

**********

CD6

FDS ROUTINES USED:

XLFL, XLM, XLFL, XLSB

XRCEP, XRMOV, JRMAP, XRCP, XRSET, XRUPK, XTCOM

**********

CD7

COMMON USED:

ERICA

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

**

BEGIN XLPRO

1. IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
2. INITIALIZE MASTER AND SUB STATES
3. PERFORM LIBD TO UPDATE LIBRARY DIRECTORY
4. IF PROCESSOR HAS AN INTERFACE TABLE THEN
5. SET RETN = 2

6. CALL XCTOM TO PROMPT FOR # PARAMETERS
7. ERREXIT IF XCTOM RETURN CODE IS NOT ZERO OR
8. ERREXIT IF # PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
9. CREATE HEXDEP ENTRY WITH #PARAMETERS AND PROCESSOR NAME
10. CALL XLSPS TO WRITE INSTRUCTIONS FOR ENTERING SPECS
11. DO FOR # PARAMETERS
12. CALL XLSPS TO CREATE ONE PARAMETER ENTRY
13. ENDIF
14. SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
15. CALL XLPLF TO CREATE >PROMPT FILE
16. SET RETN = 3

17. IF RETURN CODE IS NOT ZERO OR
18. ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
19. SET FLAG TO CALL INTERFACE TABLE EDITOR
20. EXIT XLPRO

21. ELSE
22. SET CODES ARRAY TO ADD ONLY ABSTRACT
23. CALL XLPLF TO CREATE PROMPT FILE
24. PERFORM XLPRO - NO RETURN EXPECTED
25. ENDF

26. CALL XLPLF TO CREATE THE DEFAULT INTERFACE TABLE/FILE
27. SET VALFLAG TO SET ORIGINAL REQUEST TO ADD A PROCESSOR
28. PERFORM XLPRO - NO RETURN EXPECTED

29. IF RETURN CODE IS NOT ZERO OR
30. ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
31. CALL XRMSG TO DISPLAY ERROR MESSAGE
32. GO TO (:PRMPT1:, :PRMPT2:, :PRMPT3:), RETN
33. END XLPRO
1 BEGIN LIBD
2 CALL OPEN TO OPEN LIBRARY DIRECTORY
3 IF RETURN CODE SAYS FILE NOT FOUND THEN
4 SET RECORD 1 TO ALL ZEROS
5 ELSE
6 ERREXIT IF FILE ERROR TO :FILERR:
7 CALL READF AND CLOSE TO READ IN LIBRARY DIRECTORY
8 ERREXIT IF FILE ERROR TO :FILERR:
9 IF # PROCESSORS + 1 > 50 THEN
10 CALL XRMAT TO WRITE ERROR: TOO MANY PROCESSORS
11 EXIT XLIBD
12 ENDIF
13 SET RETN = 1
14 :PRMPT:
15 CALL JXTOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
16 EXIT XLIBD IF RETURN CODE SAYS T ENTERED
17 ERREXIT IF RETURN CODE > ZERO OR
18 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
19 CALL XRMAT TO MOVE PROCESSOR NAME INTO ENTRY
20 ERREXIT IF VERSION IS NOT INTEGER VALUE 0-127 TO :PRMERR:
21 CALL XRSET TO SET VERSION IN ENTRY
22 ERREXIT IF INTERFACE TABLE OPTION IS NOT YE OR NO TO :PRMERR:
23 SET IT BIT = 0
24 IF RESPONSE IS YES THEN
25 SET IT BIT = 1
26 ENDIF
27 CALL XRSET TO SET BIT ON/OFF
28 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
29 INCREMENT # PROCESSORS BY 1
30 CALL XRMAT TO MOVE NEW ENTRY INTO XLIBD
31 IF # PROCESSORS > 1 THEN
32 CALL PURGE TO PURGE OLD FILE
33 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
34 ENDIF
35 CALL CREATE, WRITE AND CLOSE TO CREATE NEW LIBRARY DIRECTORY
36 ERREXIT IF FILE ERROR TO :FILERR:
37 ENDIF
38 1 END LIBD
1 CD************
2 CD
3 CD0
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0
9 CD0
10 CD0
11 CD0
12 CD0
13 CD0
14 CD0
15 CD0
16 CD0
17 CD0
18 CD0
19 CD0
20 CD0
21 CD0
22 CD0
23 CD0
24 CD0
25 CD0
26 CD0
27 CD0
28 CD0
29 CD0
30 CD0
31 CD0
32 CD0
33 CD0
34 CD0
35 CD0
36 CD0
37 CD0
38 CD0
39 CD0
40 CD0
41 CD0
42 CD0
43 CD0
44 CD0
45 CD0
46 CD0
47 CD0
48 CD0
49 CD0
50 CD0
51 CD0
52 CD0
53 CD0
54 CD0
55 CD0
56 CD0
57 CD0
58 CD0
59 CD0
60 CD0
61 CD0
62 CD0
63 CD0
64 CD0
65 CD0
66 CD0
67 CD0
68 CD0
69 CD0
70 CD0
71 CD0
72 CD0
73 CD0
74 CD0
75 CD0
76 CD0
77 CD0
78 CD0
79 CD0
80 CD0
81 CD0
82 CD0
83 CD0
84 CD0
85 CD0
86 CD0
87 CD0
88 CD0
89 CD0
90 CD0
91 CD0
92 CD0
93 CD0
94 CD0
95 CD0
96 CD0
97 CD0
98 CD0
99 CD0
100 CD0
101 CD0
102 CD0
103 CD0
104 CD0
105 CD0
106 CD0
107 CD0
108 CD0

FORTRAN CALLING SEQUENCE:

CALL XLCDB

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

INPUTS IN COMMON:

   XE(3) QUAH, XE(7) NMBH, XE(8) SECU
   XE(142) ICR, XE(5) OLDNL, XE(6) NEWNL
   XE(9) TOTNL

INTERNAL VARIABLES:

   XB(17) FREC - FIRST RECORD NUMBER IN DATBUF
   XB(18) LREC - LAST RECORD NUMBER IN DATBUF
   XB(40) IDC - DCB FOR OLDNL
   XB(56) IDC - DCB FOR NEWNL
   XB(20) TOCBL - BUFFER FOR COMPLETE DATA BASE TOC

RTE ROUTINES USED:

   CLOSE, CREAT, KCVT, OEP,
   PRED, READF, WRITF

FDS ROUTINES USED:

   XDDDB, XREX, XRMG

COMMON USED:

   EQUIVALENCE (XE(3), QUAL),
   +XE(7), NMBH, XE(3), SECU,
   +XE(142), ICR, XE(5), OLDNL,
   +XE(6), NEWNL, XE(8), TOTNL,
   +XE(10), FICHR, XE(5), QUALI,
   +XE(11), MOTOC, XE(14), SIZE,
   +XE(5), MCG, XE(16), TICPR,
   +XE(17), FREC, XE(5), LREC,
   +XE(39), ERR, XE(40), IDC,
   +XE(20), IDC, XE(6), NEWNL,
1 BEGIN XLCDB
2 CALL OPEN TO OPEN OLDFIL
3 ERREXIT IF OPEN ERROR TO :ERR3:
4 CALL READF TO READ IN HEADER RECORD
5 ERREXIT IF READF ERROR TO :ERR2:
6 COMPUTE NUMBER OF TOC RECORDS (NOTOC) AS (# ENTRIES * 16) / 16
7 IF NOTOC > 1 THEN
8 CALL READF TO READ IN REMAINING TOC RECORDS
9 ERREXIT IF READF ERROR TO :ERR2:
10 ENDIF
11 SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
12 CALL CREAT TO CREATE NEWFIL
13 ERREXIT IF CREAT ERROR TO :ERR2:
14 DO FOR EACH TOC ENTRY
15 IF REQUEST WAS FOR NEWFIL THEN
16 ERREXIT IF NAME > 4 CHAR OR
17 ERREXIT IF CLASS IS DNO AND NAME > 2 CHAR TO :ERR1:
18 APPEND NEWFIL CONVENTION TO FRONT OF NAME
19 ELSE
20 REMOVE NEWFIL CONVENTION FROM NAME
21 ENDIF
22 ENDDO
23 CALL WRITE TO WRITE NEW TOC RECORDS TO NEWFIL
24 ERREXIT IF WRITE ERROR TO :ERR1:
25 SET TOCPTR TO FIRST TOC ENTRY
26 SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
27 CONVERT SIZE FROM BLOCKS TO WORDS
28 DO UNTIL ALL RECORDS ARE CLEARED (SIZE = 0)
29 SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
30 IF SIZE < LENGTH THEN
31 SET LENGTH TO SIZE
32 ENDIF
33 ERREXIT IF READF ERROR TO :ERR1:
34 DECREMENT SIZE BY LENGTH READ
35 UPDATE FIRST AND LAST RECORD NUMBERS
36 START SEARCH FOR TOCPtr = TOCPtr TO LAST TOC ENTRY
37 EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
38 IF CLASS IS AN INTERFACE TABLE THEN
39 COMPUTE INDEX INTO DATBUF FROM FIRST RECORD #, RECORD # IN TOC ENTRY
40 AMOUNT INDEX IN TOC ENTRY
41 SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
42 ENDIF
43 ENDSEARCH
44 CALL WRITF TO WRITE LENGTH WORDS OF DATA TO NEWFIL
45 ERREXIT IF WRITF ERROR TO :ERR1:
46 ENDDO
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>458</td>
<td>2</td>
</tr>
<tr>
<td>459</td>
<td>2</td>
</tr>
<tr>
<td>460</td>
<td>2</td>
</tr>
<tr>
<td>461</td>
<td>2</td>
</tr>
<tr>
<td>462</td>
<td>1</td>
</tr>
<tr>
<td>463</td>
<td>2</td>
</tr>
<tr>
<td>464</td>
<td>2</td>
</tr>
<tr>
<td>465</td>
<td>2</td>
</tr>
<tr>
<td>466</td>
<td>2</td>
</tr>
<tr>
<td>467</td>
<td>2</td>
</tr>
<tr>
<td>468</td>
<td>2</td>
</tr>
<tr>
<td>469</td>
<td>2</td>
</tr>
<tr>
<td>470</td>
<td>3</td>
</tr>
<tr>
<td>471</td>
<td>3</td>
</tr>
<tr>
<td>472</td>
<td>3</td>
</tr>
<tr>
<td>473</td>
<td>2</td>
</tr>
<tr>
<td>474</td>
<td>2</td>
</tr>
<tr>
<td>475</td>
<td>2</td>
</tr>
<tr>
<td>476</td>
<td>2</td>
</tr>
<tr>
<td>477</td>
<td>2</td>
</tr>
<tr>
<td>478</td>
<td>2</td>
</tr>
<tr>
<td>479</td>
<td>2</td>
</tr>
<tr>
<td>480</td>
<td>1</td>
</tr>
</tbody>
</table>
FORTRAN CALLING SEQUENCE:

CALL XLDOS ('XLDOS')

XLDOS IS THE DATA BASE FILES MAIN ROUTINE. IT DETERMINES WHICH
DATA BASE ROUTINE IS NEEDED AND GATHERS INPUTS NEEDED BY THAT
ROUTINE. HANDLES ALL PRD 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 XRQFM
XE(6) SECU - TEMPORARY SLOT FOR ISECU DURING THIS OVERLAY
X(5) OLDNAME - OLD NAME
X(6) NEWNAME - NEW NAME
X(9) TOTSIZ - SIZE OF OLD/NEW NAME IN BLOCKS
X(10) FILECHR - 4 BASE CHARACTERS OF FILE NAME
X(12) QNAM - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPENW

FDS ROUTINES USED:

XDDBA, XDDBY, XERW, XLDDB, XLPWR

COMMON USER:

ENQUIVALENCE (XE(3), ISECU),
(XE(7), NUMBR),
(XE(85), TOKENS),
(XE(142), ICR),
(XE(145), CONTDF),
(X(6), NEWNAME),
(X(9), TOTSIZ),
(X(10), FILECHR),
(X(12), QNAM),
(X(100), LDGB)
537 1 BEGIN XLDBF;
538 2 IF REQUEST IS TO CREATE/modify LOG FILE THEN
539 3 CALL OPEN TO OPEN MDB/PDB LOG FILE
540 3 IF OPEN ERROR SAYS FILE NOT FOUND THEN
541 4 CALL XLPCR TO CREATE MDB/PDB LOG FILE
542 3 ELSE
543 4 ERREXIT IF OPEN ERROR TO :FILERR;
544 4 CALL XLPMOD TO MODIFY MDB/PDB LOG FILE
545 3 ENDIF
546 ELSE
547 DO UNTIL USER REQUESTS EXIT (X)
548 4 CALL XCOM TO PROMPT USER FOR NAME AND USER ID
549 4 IF RESPONSE IS NOT EXIT (PERCENT) THEN
550 5 ERREXIT IF RESPONSE IS INVALID TO :ERR1:
551 5 SAVE A CHARACTER NAME AND ID IN COMMON
552 5 IF REQUEST WAS PDB TO MDB THEN
553 6 SET QUALIFIER TO SEARCH FOR PDB NAME
554 5 ELSE (REQUEST WAS FOR MDB TO PDB)
555 6 SET QUALIFIER TO SEARCH FOR MDB NAME
556 5 ENDIF
557 5 CALL XDBBV TO VERIFY EXISTENCE OF MDB/PDB DEPENDING ON QUALIFIER
558 5 ERREXIT IF NAME WAS NOT FOUND TO :ERR1:
559 5 ERREXIT IF FILE MANAGER ERROR TO :FILERR:
560 5 IF REQUEST WAS PDB TO MDB THEN
561 6 SET QUALIFIER TO ADD PDB TO LOG FILE
562 5 ELSE (REQUEST WAS MDB TO PDB)
563 6 SET QUALIFIER TO ADD PDB TO LOG FILE
564 5 ENDIF
565 5 CALL XDBDA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
566 6 ERREXIT IF DUPLICATE NAME OF
567 6 ERREXIT IF MAXIMUM NUMBER OF ENTRIES EXIST TO :ERR1:
568 6 ERREXIT IF FILE MANAGER ERROR TO :FILERR:
569 6 IF REQUEST WAS PDB TO MDB THEN
570 6 CALL XRDFM TO SET OLDFIG TO PDB NAME
571 6 SET NEWFILE TO MDB NAME
572 6 ELSE (REQUEST WAS MDB TO PDB)
573 6 SET OLDFIG TO MDB NAME
574 6 CALL XRDFM TO SET NEWFILE TO PDB NAME
575 6 ENDIF
576 5 CALL XLDBS TO COPY OLDFIG TO NEWFILE
577 3 ENDIF
578 3 ENDDO
579 2 ENDF
580 1 EXIT XLDBF

581 2 :ERR1:
582 2 CALL XRMSG TO DISPLAY ERROR
583 2 RETURN TO PROMPT FOR ANOTHER 4 CHARACTERS AND USER ID
584 2 :FILERR:
585 2 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
586 1 END XLDBF
588  1 CD************
589  1 CD0
590  1 CD0  FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
591  1 CD0
592  1 CD0  CALL XEDS ('XLDEL')
593  1 CD0
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  COMMAR - (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, KCVT, CREAT, OPEN, READF, WRITF, CLOSE, PURGE
613  1 CD5
614  1 CD5  FDS ROUTINES USED:
615  1 CD5  XRCP, XREX, XRM, XREN, XRSCP, XRSE, XRSET, XRPK, XTCOM
616  1 CD5
617  1 CD5  XE AND XB COMMON USED
618  1 CD5  + (XE(3), ISECU),
619  1 CD5  + (XE(142), ICR), (XE(145), COMBUF),
620  1 CD5  + (XB(48), LIBO1), (XB(51), LIBO2)
621  1 CD5
622  1 CD5
623  1 CD5
624  1 CD************
1 BEGIN XDEL

2 :PROMPT:
3 CALL XICOM TO PROMPT FOR PROCESSOR NAME
4 IF XICOM RETURN CODE IS NOT ZERO OR
5 RESPONSE IS NOT A VALID PROCESSOR NAME THEN
6 CALL XRMSG TO WRITE ERROR MESSAGE
7 GO TO :PROMPT:
8 EXIT XDEL IF RETURN CODE SAYS % ENTERED
9 ENDF:
10 CALL OPEN, READ AND CLOSE TO READ IN LIBRARY DIRECTORY
11 ERREXIT IF FILE ERROR TO :FILERR:
12 IF PROCESSOR IS NOT IN LIBRARY DIRECTORY THEN
13 CALL XRMSG TO DISPLAY ERROR
14 GO TO :PROMPT:
15 ENDF:
16 RECENNT # PROCESSORS BY 1
17 CALL PURGE TO PURGE OLD LIBRARY DIRECTORY
18 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
19 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 XRMSG TO SAY FILE PURGED SUCCESSFULLY
26 IF PROCESSOR HAD AN INTERFACE TABLE THEN
27 CREATE DEFAULT IT NAME
28 CALL PURGE TO PURGE DEFAULT IT
29 ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:
30 CALL XRMSG TO DISPLAY 'FILE NOT AGED' MESSAGE
31 ENDF:
32 GO TO :PROMPT:
33 :FILERR: CALL XRMSG TO WRITE FILE ACCESS ERROR
34 1 END XDEL.
661 1 CD***********
662 1 CD
663 1 CD FORTRAN CALLING SEQUENCE:
664 1 CD
call-xlifl
665 1 CD
666 1 CD
667 1 CD***********
668 1 CD
669 1 CD XLIFL CREATES THE DEFAULT INTERFACE TABLE FILE
670 1 CD
671 1 CD***********
672 1 CD
673 1 CD INTERNAL VARIABLES
674 1 CD
675 1 CD BLOKS - (INTEGER, 1 WORD) # BLOCKS TO BE ALLOCATED TO
676 1 CD THE FILE
677 1 CD
678 1 CD***********
679 1 CD
680 1 CD FDS ROUTINES USED:
681 1 CD
682 1 CD XREXT, XRMSG
683 1 CD
684 1 CD RTE ROUTINES USED:
685 1 CD
686 1 CD CLOSE, CREAT, WRITF
687 1 CD
688 1 CD COMMON USED:
689 1 CD
690 1 CD EQUIVALENCE (X(2), IEXEC),
691 1 CD + (X(142), ICFR), (X(90), LITP1R),
692 1 CD + (X(91), LITLE),
693 1 CD + (X(96), NPARM), (X(101), DEDP),
694 1 CD + (X(105), PARMS), (X(21), IERR),
695 1 CD + (X(2), IDCBO)
1 BEGIN XLIFL
2 EXTRACT LITERAL LENGTH FROM HEADER
3 COMPUTE # BLOCKS FOR THIS FILE
4 CALL CREATE TO CREATE DEFAULT INT TABLE FILE
5 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
6 CALL WRITE TO WRITE HEADER AND SPECS
7 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
8 IF THERE IS A LITERAL RECORD THEN
9 CALL WRITE TO WRITE LITERALS
10 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
11 ENDEF
12 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
13 CALL XFILES TO DISPLAY FILE (NAME) CREATED MESSAGE
14 1 EXIT XLIFL
15 2 :FILERR:
16 2 CALL XFILES TO WRITE FILE ACCESS ERROR
17 2 CALL XEXIT TO RETURN TO MAIN PROGRAM
18 1 END XLIFL
1 CD*********
2 DO
3 DO
4 CALL XLINS
5 DO
6 CALL XLINS
7 XLINS DISPLAYS INSTRUCTIONS FOR ENTERING PARAMETER SPECS
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 1 BEGIN XLINS
52 2 CALL EXEC TO DISPLAY ALL 7 LINES
53 1 END XLINS
DECLARE **COD**:

FORTRAN CALLING PROCEDURE:

CALL XELDS (XLINT)

XLINT SEGMENT SETS UP COMMON TO CALL THE INTERFACE TABLE TO

ACCEPT DEFAULT VALUES FOR THE INTERFACE TABLE

FDS FUNCTIONS AND SUBROUTINES USED:

XEINT, XERTW, XIMIX, XKMOV

COMMON USED:

**EQUIVALENCE** (**XECS**, **MASSTA**),

**(XEC(6), **SUBSTA) (**XEB(1), **NUMDR)**,

**(XEC(2), **DIREC) (**XEB(125), **LSTFLG)**,

**(XEC(37), **ARCHD) (**XEB(61), **PRETHD)**,

**(XEC(73), **SIZE) (**XEB(89), **LITDHN)**,

**(XEC(90), **LITFTP) (**XEB(91), **LITLEN)**,

**(XEB(92), **MAPE)**,

**(XEB(96), **NUMARG)** (**XEB(97), **HEMTAB)**,

**(XEB(100), **WKBLOG)** (**XEB(101), **WKBUF)**,

**(XEB(1400), **EMB)**
BEGIN XLINT

1. SET MASTER STATE AS EXEC
2. SET SUBSTATE AS INTERFACE TABLE EDITOR
3. SET UP LIST OF VALID DIRECTIVES
4. SET UP LIST FLAG TO GET ENTIRE LIST
5. SET CURRENT ARGUMENT TO ZERO
6. SET PROMPT MODE TO ALL
7. SET INDEX INTO PAMS OF SHORT PROMPTS
8. SET NEW TABLE NAME TO ZERO
9. INITIALIZE ARRAY OF TYPE LENGTHS
10. IF LITERAL LENGTH IS ZERO THEN
11. SET LITERAL POINTER TO FIRST AVAILABLE WORD
12. ELSE
13. SET LITERAL POINTER TO FIRST WORD OF LITERALS
14. CALL XEINT TO UNPACK LITERALS
15. ENDIF
16. CALL XINIX TO ACCEPT DEFAULT VALUES
17. CALL ZERTM TO RETURN
18. END XLINT
FORTRAN CALLING SEQUENCE:

CALL XLMOD ('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, READ, WRITE

FBS ROUTINES USED:

XLI, XLIV, XLPIL, XLS1, XRECP, XREX, XREO, XRXCP, XRXSET, XRXUP, XRXUN

COMMON USED:

1 EQUIVALENCE (XE(3), XECU),
2 (IEE(3), HASSA), (XE(6), SUBSTA),
3 (IEE(7), VALFLO), (XE(13), PPMARK),
4 (IEE(10), PRCWAM), (XE(85), EDSTOK),
5 (IEE(86), INTOK), (XE(69), HANTOK),
6 (IEE(97), HEGOK), (XE(142), ECH),
7 (IEE(105), COUNUF), (XE(266), VERI),
8 (IEE(33), SFLAG), (XE(35), CFLAG),
9 (IEE(37), ARCHGO), (XE(38), ARGO),
10 (IEE(43), LEM), (XE(44), Nrego),
11 (IEE(88), LITHSP), (XE(90), LITPRB),
12 (IEE(92), EOX), (XE(96), EOPAR),
13 (IEE(101), HRBR), (XE(101), LEB01),
14 (IEE(104), LEB02), (XE(109), LEB03)
BEGIN XLMOD
IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
SET RTN = 1

;PRMT:
CALL XLMOD TO DISPLAY SPEC INSTRUCTIONS
;INIT:
INITIALIZE MASTER AND SUB STATE FLAGS
;CALL XTOP TO PROMPT FOR PROCESSOR NAME
EXIT XLMOD IF RETURN CODE SAYS X ENTERED
;ERRRT:
ERRRT IF XCON RETURN CODE NON-ZERO OR
ERRRT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO ;PRMT:
CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
ERRRT IF THERE IS A FILE ERROR TO ;FER:
ERRRT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO ;PRMT;
SAV INTERFACE TABLE BIT AND VERSION NUMBER
PERFORM VERSION TO UPDATE VERSION NUMBER
IF THE PROCESSOR HAD AN INTERFACE TABLE THEN
SET NEW VERSION NUMBER IN INTERFACE TABLE
CREATE DEFAULT INTERFACE TABLE NAME
CALL OPEN AND READF TO READ IN MEM AND SPECS
IF THERE ARE LITERALS THEN
CALL READF TO READ IN LITERALS
ENDIF
CALL CLOSE TO CLOSE FILE
ERRRT IF THERE WAS A FILE ERROR TO ;FER:
CREATE PROMPT TABLE NAME
CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
ERRRT IF THERE WAS A FILE ERROR TO ;FER:
CALL NAMF TO RENAME PROMPT FILE >XLMRT
ERRRT IF NAMF ERROR TO ;FER:
SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPECS
PERFORM DELPRM TO DELETE PARAMETERS
PERFORM MODPRM TO MODIFY PARAMETERS
PERFORM ADDPRM TO ADD PARAMETERS
CALL XLPFL TO CREATE NEW PROMPT FILE
PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
ELSE
CALL NAMF TO RENAME PROMPT FILE >XLMRT
ERRRT IF NAMF ERROR TO ;FER:
SET CODES ARRAY TO MODIFY ABSTRACT ONLY
CALL XLPFL TO CREATE NEW PROMPT FILE
PERFORM XLMOD - NO RETURN EXPECTED
ENDIF
END
CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
ERRRT IF FILE ERROR TO ;FER:
CALL XLIIF TO CREATE NEW DEFAULT INTERFACE TABLE FILE
SET VALILE TO SAY ORIGINAL REQUEST TO MODIFY
PERFORM XLMOD - NO RETURN EXPECTED
;FER:
CALL XMSG TO DISPLAY ERROR MESSAGE
GO TO (;PRMT1:;PRMT2:;PRMT3:;PRMT4:;PRMT5:;PRMT6:;PRMT7:);RTN
;FER:
CALL XMSG TO DISPLAY FILE ERROR
END XLMOD
905 1 BEGIN VERSION
906 2 SET RTN = 2
907 2 :PRM2:
908 2 CALL XICOM TO PROMPT USER FOR VERSION NUMBER
909 3 IF RETURN CODE IS NOT CR THEN
910 4 ERREXIT IF RETURN CODE IS NON-ZERO OR
911 4 ERREXIT IF VERSION IS INVALID TO :PRM2:
912 4 IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
913 4 CALL XRSET TO PUT NEW VERSION IN ENTRY
914 4 CALL OPEN, WRITE, CLOSE TO UPDATE LIBRARY DIRECTORY
915 4 ERREXIT IF FILE ERROR TO :FILEERR:
916 3 ENDIF
917 2 ENDIF
918 1 END VERSION
919 1 =
920 1 =
921 1 =
922 1 BEGIN DELPRM
923 2 SET RTN = 3
924 2 :PRM3:
925 2 DO UNTIL RETURN CODE IS CR ENTERED
926 3 IF NUMBER OF PARAMETERS > 1 THEN
927 4 CALL XICOM 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 CHDAT TO CHECK FOR EXISTING DATA
931 5 SET ARCG0 TH WORD IN CODES TO SAY 'DELETED'
932 5 CALL XRMOV TO MOVE DATA TO DELETE PARAMETER
933 5 DECREMENT # PARAMETERS BY 1
934 4 ENDIF
935 3 ELSE
936 4 CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
937 4 EXIT DELPRM
938 3 ENDIF
939 2 ENDDO
940 1 END DELPRM
941 1 =
942 1 =
943 1 =
944 1 BEGIN MODPRM
945 2 SET RTN = 4
946 2 :PRM4:
947 2 DO UNTIL RETURN CODE IS CR ENTERED
948 3 CALL XICOM 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 CHDAT TO CHECK FOR EXISTING DATA
952 4 SET ARCG0 TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
953 4 CALL CLREE TO PROMPT USER FOR SPECIFICATIONS
954 4 SET IT COMPLETE BIT OFF
955 4 ENDIF
956 2 ENDDO
957 1 END MODPRM
959 1 BEGIN ADDPRM
960 2 SET RTN = 5
961 2 :PROMS:
962 2 : DO UNTIL RETURN CODE IS CR ENTERED
963 3 : IF NUMBER OF PARAMETERS < 63 THEN
964 4 : CALL XTCOM TO PAROMET FOR ADD PARAMETER BEFORE/AFTER PARAMETER NAME
965 4 : IF RETURN CODE IS NOT CR ENTERED THEN
966 5 : PERFORM RSPND TO INTERPRET RESPONSE
967 5 : SET ARGNO THE NON-DELETED WORD IN CODES TO SAY 'ADDED'
968 5 : INCREMENT NUMBER OF PARAMETERS BY 1
969 5 : CALL XMNV TO MOVE DATA TO MAKE SPACE FOR NEW PARAMETER
970 5 : CALL XLSPS TO SET NEW SPECS FOR THIS PARAMETER
971 5 : SET IT COMPLETE BIT OFF
972 4 : ENDIF
973 3 ELSE
974 4 : CALL XMMSG TO DISPLAY NO MORE PARAMETERS CAN BE ADDED
975 3 : EXIT ADDPRM
976 3 : ENDIF
977 2 ENDDO
978 1 END ADDPRM
979 1 *
980 1 *
981 1 *
982 1 BEGIN DEFAULT
983 2 SET RTN = 7
984 2 :PROMT:
985 2 : CALL XTCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
986 2 : CALL XTCOM TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
987 2 : ErrEXIT IF RETURN CODE IS NON-ZERO TO :PRMERR:
988 2 : IF RESPONSE IS YES THEN
989 3 : SET VALFLAG TO SAY CALL INTERFACE TABLE EDITOR
990 2 : EXIT XLMOD
991 2 : ENDIF
992 1 END DEFAULT
BEGIN RSPM2O IF RETURN CODE IS NON-ZERO TO SPINNER:

1 BEG INFPEX IF RETURN CODE IS NOT IN INTERFACE TABLE TO SPINNER:
1067 1 BEGIN XLMG
1068 2 OPEN XLMG
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 ELSEDO
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-NUL 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 1 END XLMG
1115 1 C0************
1116 1 C00
1117 1 C00      FORTRAN CALLING SEQUENCE:
1118 1 C00           CALL XLRP
1119 1 C00
1120 1 C00 1 C00************
1121 1 C00
1122 1 C01
1123 1 C01      XLRP CREATES AND INITIALIZES THE PDB LOG FILE
1124 1 C01
1125 1 C01************
1126 1 C02
1127 1 C02      INPUTS FROM COMMON:
1128 1 C02           XE(3) ISECU, XE(142) ICR
1129 1 C02
1130 1 C02
1131 1 C02************
1132 1 C03
1133 1 C03      RTE ROUTINES USED:
1134 1 C03
1135 1 C03          CLOSE, CREAT, WRITF
1136 1 C03
1137 1 C03      FDS ROUTINES USED:
1138 1 C03           XRMV, XRMSC, XTCOM
1139 1 C03
1140 1 C03      COMMON USED:
1141 1 C03
1142 1 C03        EQUIVALENCE (XE(3), ISECU),
1143 1 C03           *(XE(85), TKNERS), (XE(142), ICR),
1144 1 C03           *(XE(145), COMBDF), (XE(99), IERB),
1145 1 C03           *(XE(100), IODCF), (XE(116), EBUF),
1146 1 C03
1147 1 C03************
1 BEGIN XL_CR
2 CALL CREAT TO CREATE MDB/PDB LOG FILE
3 ERREXIT IF CREATE ERROR TO :FILERR:
4 INITIALIZE LOG RECORD BUFFER TO ZEROS
5 SET # MDB FILES CURRENTLY USED TO ZERO
6 SET MAXIMUM NUMBER MDBS TO 20
7 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
8 ERREXIT IF WRITE ERROR TO :FILERR:
9 DO FOR EACH REMAINING LOG RECORD
10 DO FOR THIS PAIR OF USER ID'S
11 CALL XCTOM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
12 SET MAXIMUM # PDB FILES TO RESPONSE
13 SET # PDB FILES CURRENTLY USED TO ZERO
14 ENDDO
15 CALL WRITE TO WRITE 1 PDB RECORD TO LOG FILE
16 ERREXIT IF WRITE ERROR TO :FILERR:
17 ENDDO
18 CALL close TO close file
19 ERREXIT IF close error TO :FILERR:
20 EXIT XL_CR
21 :FILERR:
22 CALL XRMSG TO DISPLAY FILE ERROR
23 CALL close TO close file
24 END XL_CR
FORTRAN CALLING SEQUENCE:

CALL XLPFL (NOCOD, CODES)

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XE(1)  LU, XE(3)  ISECU, XE(16)  PRCNAM,
XE(142)  ICR, XB(96)NOPARM, XB(108)PARMS

RTE FUNCTIONS USED:
CLOSE, CREAT, EXEC, OPEN,
POSN, PNGE, READF, WRIT

FDS FUNCTIONS USED:
XERTN, XRCPR, XPMOV, XRMSG,
XRCK, XRUPK, XTCOM

COMMON USED:

EQUIVALENCE, (XE(1), LU,)

+ (XE(3), ISECU), (XE(7), VALFL6),
+ (XE(16), PRCNAM), (XE(85), TOKEMS),
+ (XE(142), ICR, (XE(144), COMPR),
+ (XE(165), COMBUS), (XB(96), NOPARM),
+ (XB(108), PARMS)
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1227</td>
<td>BEGIN XLFL</td>
</tr>
<tr>
<td>1228</td>
<td>COMPUTE SIZE OF FILE AS 0 PARAMETERS +3</td>
</tr>
<tr>
<td>1229</td>
<td>CALL CREAT TO CREATE PROMPT FILE</td>
</tr>
<tr>
<td>1230</td>
<td>ERREXIT IF CREAT ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1231</td>
<td>STUFF SYNTAX RECORD 0 AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS</td>
</tr>
<tr>
<td>1232</td>
<td>CALL WRTDF TO WRITE SHORT PROMPT RECORD(S)</td>
</tr>
<tr>
<td>1233</td>
<td>ERREXIT IF WRTDF ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1234</td>
<td>IF ABSTRACT CODE IS MODIFY THEN</td>
</tr>
<tr>
<td>1235</td>
<td>CALL OPEN TO OPEN OLD PROMPT FILE &gt;XLTMP</td>
</tr>
<tr>
<td>1236</td>
<td>ERREXIT IF OPEN ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1237</td>
<td>CALL READF TO READ EXISTING ABSTRACT</td>
</tr>
<tr>
<td>1238</td>
<td>ERREXIT IF READF ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1239</td>
<td>CALL XCOM TO DISPLAY EXISTING ABSTRACT</td>
</tr>
<tr>
<td>1240</td>
<td>CALL XCOM TO PROMPT USER TO MODIFY ABSTRACT</td>
</tr>
<tr>
<td>1241</td>
<td>IF RESPONSE IS CR (NO MODIFICATION) THEN</td>
</tr>
<tr>
<td>1242</td>
<td>CALL WRTDF TO WRITE EXISTING ABSTRACT TO NEW FILE</td>
</tr>
<tr>
<td>1243</td>
<td>ERREXIT IF WRTDF ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1244</td>
<td>ELSE (NEW ABSTRACT WAS ENTERED)</td>
</tr>
<tr>
<td>1245</td>
<td>PERFORM FORMAT TO FORMAT RESPONSE INTO FILE</td>
</tr>
<tr>
<td>1246</td>
<td>ENDIF</td>
</tr>
<tr>
<td>1247</td>
<td>ELSE (ABSTRACT CODE IS AD)</td>
</tr>
<tr>
<td>1248</td>
<td>CALL XCOM TO PROMPT USER TO ENTER NEW ABSTRACT</td>
</tr>
<tr>
<td>1249</td>
<td>PERFORM FORMAT TO FORMAT RESPONSE INTO FILE</td>
</tr>
<tr>
<td>1250</td>
<td>ENDIF</td>
</tr>
<tr>
<td>1251</td>
<td>IF PARAMETERS IS NOT ZERO THEN</td>
</tr>
<tr>
<td>1252</td>
<td>PERFORM EXTPRM TO GET DEFINITIONS FOR EACH PARAMETER</td>
</tr>
<tr>
<td>1253</td>
<td>ENDIF</td>
</tr>
<tr>
<td>1254</td>
<td>CALL CLOSE TO CLOSE PROMPT FILE</td>
</tr>
<tr>
<td>1255</td>
<td>ERREXIT IF CLOSE ERROR TO :FILERR:</td>
</tr>
<tr>
<td>1256</td>
<td>IF THERE WAS AN OLD FILE THEN</td>
</tr>
<tr>
<td>1257</td>
<td>CALL CLOSE TO CLOSE OLD FILE &gt;XLTMP</td>
</tr>
<tr>
<td>1258</td>
<td>CALL PURGE TO PURGE OLD FILE &gt;XLTMP</td>
</tr>
<tr>
<td>1259</td>
<td>ENDIF</td>
</tr>
<tr>
<td>1260</td>
<td>EXIT XLFL</td>
</tr>
<tr>
<td>1261</td>
<td>:FILERR:</td>
</tr>
<tr>
<td>1262</td>
<td>CALL XRMNG TO DISPLAY ERROR CODE</td>
</tr>
<tr>
<td>1263</td>
<td>CALL CLOSE TO CLOSE NEW FILE</td>
</tr>
<tr>
<td>1264</td>
<td>CALL CLOSE TO CLOSE OLD FILE &gt;XLTMP</td>
</tr>
<tr>
<td>1265</td>
<td>CALL PURGE TO PURGE OLD FILE &gt;XLTMP</td>
</tr>
<tr>
<td>1266</td>
<td>SET VALFLG = 0</td>
</tr>
<tr>
<td>1267</td>
<td>CALL XERM TO RETURN TO MAIN <em><strong>NO RETURN TO HERE</strong></em></td>
</tr>
<tr>
<td>1268</td>
<td>END XLFL</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>1270</td>
<td>BEGIN FORMAT</td>
</tr>
<tr>
<td>1271</td>
<td>INITIALIZE TOKEN POINTER AND TOTAL WORD COUNT</td>
</tr>
<tr>
<td>1272</td>
<td>DO UNTIL EOS IS DETECTED IN RESPONSE</td>
</tr>
<tr>
<td>1273</td>
<td>ERREXIT IF RESPONSE IS NOT CHARACTER STRING TO :ERR1:</td>
</tr>
<tr>
<td>1274</td>
<td>ERREXIT IF RESPONSE IS TOO LONG (&gt;128 WORDS) TO :ERR1:</td>
</tr>
<tr>
<td>1275</td>
<td>CALL XMTO MOV RESPONSE TO BUFFER</td>
</tr>
<tr>
<td>1276</td>
<td>SET CONTROL CHARACTERS IN BUFFER</td>
</tr>
<tr>
<td>1277</td>
<td>INCREMENT TOTAL WORD COUNT BY THIS RESPONSE</td>
</tr>
<tr>
<td>1278</td>
<td>ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:</td>
</tr>
<tr>
<td>1279</td>
<td>INCREMENT TOKEN POINTER TO NEXT CHARACTER STRING</td>
</tr>
<tr>
<td>1280</td>
<td>ENDDO</td>
</tr>
<tr>
<td>1281</td>
<td>SET REMAINDER OF BUFFER TO NULL</td>
</tr>
<tr>
<td>1282</td>
<td>CALL WRTF TO WRITE NEW RESPONSE TO PROMPT FILE</td>
</tr>
<tr>
<td>1283</td>
<td>ERREXIT IF WRITF ERROR TO :FILEERR:</td>
</tr>
<tr>
<td>1284</td>
<td>1 END FORMAT</td>
</tr>
<tr>
<td>1285</td>
<td>1 :ERR1:</td>
</tr>
<tr>
<td>1286</td>
<td>1 CALL XMSG TO DISPLAY INVALID RESPONSE</td>
</tr>
<tr>
<td>1287</td>
<td>1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE</td>
</tr>
</tbody>
</table>
1299 1 BEGIN EXTYPH
1299 2 DO FOR EACH ENTRY IN CODES ARRAY
1299 3 BEGIN CASE (:MOD:, :DEL:, :ADD:, CODES+)
1299 4 :MOD: (SPECs WERE MODIFIED)
1299 5 CALL READ TO READ NEW DEFINITION FROM XLTP
1299 6 ERREXIT IF READF ERROR TO :FILERR:
1299 7 CALL EXEC TO DISPLAY EXISTING DEFINITION
1299 8 IF RESPONSE IS CR (NO RESPONSE) THEN
1299 9 CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
1299 10 ELSE (NEW DEFINITION WAS ENTERED)
1299 11 PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
1299 12 ENDF
1299 13 :DEL: (SPECs WERE DELETED)
1299 14 CALL POSNT TO POSITION XLTP OVER THIS ENTRY
1299 15 ERREXIT IF POSMT ERROR TO :FILERR:
1299 16 :ADD: (SPECs WERE ADDED)
1299 17 CALL XT:COM TO PROMPT USER TO ENTER NEW DEFINITION
1299 18 PERFOR: FORMAT TO FORMAT RESPONSE INTO FILE
1299 19 ENDCASE
1299 20 DO UNTIL RESPONSE IS CR (EXIT)
1299 21 CALL XT:COM TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
1299 22 EXIT EXTYPH IF RESPONSE IS CR
1299 23 SEARCH SHOPT PROPTS FOR RESPONSE
1299 24 ERREXIT IF NOT FOUND TO :ERR:
1299 25 CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
1299 26 ERREXIT IF READF ERROR TO :FILERR:
1299 27 CALL EXEC TO DISPLAY EXISTING DEFINITION
1299 28 CALL XT:COM TO PROMPT USER TO MODIFY DEFINITION
1299 29 IF RESPONSE IS POSITIVE
1299 30 CALL POSNT TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
1299 31 ERREXIT IF POSNT ERROR TO :FILERR:
1299 32 PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
1299 33 ENDF
1299 34 ENDF
1299 35 END EXTPRH
**FORTRAN CALLING SEQUENCE:**

```
CALL XLPMO
```

**XLPMO MODIFIES THE EXISTING PDB LOG FILE**

**INPUTS FROM COMMON:**

- `XE(1) LU`, `XE(3) QUAL`, `XE(8) SECU`,
- `XE(162) ICA`, `XE(99) IERR`, `XE(100) INCB`,

**INTERNAL VARIABLES IN COMMON:**

- `XE(52) UNO` - NUMBER OF USER ID A-Z (1-26)
- `XE(53) RECNO` - RECORD # WHERE UNO'S PDB LIST IS (2-14)
- `XE(52) PHAM` - FILE NAME OF USER'S PDB (FULL 6 CHAR)
- `XE(55) DIFF` - NUMB' OF FILES THAT MUST BE DELETED
- WHEN NEW MATH CURRENTLY IN USE
- `XE(244) OUTBUF-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

**RTE ROUTINES USED:**

- CLOSE, EXEC, KEVT, PURGE, READF, WRITF

**FDS ROUTINES USED:**

- `XRPCR, XREXT, XRMOV, XRMSG,
- XRFRM, XRFR, XTCOM`

**COMMON USED:**

- `EQUIVALENCE (XE(1), LU )`,
- `+XE(3), QUAL )`, `+XE(8), SECU )`,
- `+XE(162), ICA )`, `+XE(99), IERR )`,
- `+XE(100), INCB )`,
- `+XE(52), PHAM )`, `+XE(55), DIFF )`,
- `+XE(56), RTN )`, `+XE(57), MSG )`,
- `+XE(99), IERR )`, `+XE(100), INCB )`,
- `+XE(116), IBUF )`, `+XE(244), OUTBUF )`,
- `+XE(304), IBC02 )`
1387 1 BEGIN XLPHO
1388 2 DO UNTIL USER RESPONSE IS EXIT (3)
1389 3 CALL XTTCOM TO PROMPT USER FOR USER ID
1390 4 ERREXIT IF RESPONSE IS INVALID TO :ERR1:
1391 5 SAVE USER ID AND NUMBER (1-26) IN COMMON
1392 6 CALL READ TO READ IN THE RECORD CONTAINING THIS ID
1393 7 ERREXIT IF READ ERROR TO :FILERR:
1394 8 DISPLAY MAX # ALLOWED FOR THIS USER AND # CURRENTLY USED
1395 9 CALL XTTCOM TO PROMPT USER FOR NEW MAXIMUM
1396 10 IF RESPONSE IS A MODIFICATION (NOT CR) THEN
1397 11 ERREXIT IF RESPONSE IS NOT VALID TO :ERR1:
1398 12 COMPUTE DIFFERENCE AS CURRENTLY USED - NEW MAXIMUM
1399 13 IF DIFFERENCE <= 0 THEN
1400 14 CALL WRITE TO REWRITE UPDATED RECORD TO FILE
1401 15 ERREXIT IF WRITE ERROR TO :FILERR:
1402 16 ELSE
1403 17 DISPLAY LIST OF CURRENT FILES
1404 18 SET #PURGED = 0
1405 19 DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1406 20 CALL XTTCOM TO PROMPT USER TO DELETE #OVER FILES
1407 21 DO FOR EACH FILE NAME IN RESPONSE
1408 22 SEARCH LIST FOR FILE NAME
1409 23 ERREXIT IF INVALID NAME TO :ERR1:
1410 24 MOVE FILE NAME TO PURGE LIST
1411 25 INCREMENT #PURGED BY 1
1412 26 DECREMENT DIFFERENCE BY 1
1413 27 COMPRESS OLD FILE NAME OUT OF LIST
1414 28 ENDDO
1415 29 ENDDO
1416 30 IF RESPONSE WAS NOT EXIT THEN
1417 31 CALL WRITE TO REWRITE UPDATED LOG RECORD
1418 32 ERREXIT IF WRITE ERROR TO :FILERR:
1419 33 DO FOR #PURGED FILES
1420 34 CALL INKPN TO CREATE FILE NAME
1421 35 CALL PURGE TO PURGE FILE
1422 36 ENDDO
1423 37 ENDDIF
1424 38 ENDDIF
1425 39 ENDIF
1426 40 CALL CLOSE TO CLOSE NDB/PDB LOG FILE
1427 41 EXIT XLPHO
1428 42 :ERR1:
1429 43 CALL XHMSG TO DISPLAY ERROR IN RESPONSE
1430 44 RETURN TO REISSUE LAST PROMPT
1431 45 :FILERR:
1432 46 CALL XHMSG TO DISPLAY FILE ERROR
1433 47 CALL CLOSE TO CLOSE NDB/PDB FILE
1434 48 END XLPHO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

CLASS - (INTEGER, 1 WORD, XE(13)) IS THE PARAMETER CLASS
IPAMT - (INTEGER, 1 WORD) IS THE INDEX INTO PARM WHERE THE SHORT PROMPT IS TO BE STORED
ISPEC - (INTEGER, 1 WORD) IS THE INDEX INTO PARM WHERE THE 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, XRCPR, XRMOV, XRMSP, XRSET, XTCOM

X COMMON USED:

EQUIVALENCE (XE(8), RETC), (XE(9), RETN),
(XE(10), CLASS), (XE(11), TYPE),
(XE(145), COMBUF), (XE(37), ARGNO),
(XB(96), MOPARM), (XB(108), PARM)
1474 1 BEGIN XLSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
1476 2 :PRMPT1:
1477 2 SET RETURN TO 1
1478 2 CALL XTCOM TO PROMPT FOR NAME,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 ORDE 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 DATA ELEMENT THEN
1493 3 IF TYPE IS NOT SYMBOLIC STRING THEN
1494 4 SET RETURN TO 2
1495 4 :PRMPT2:
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 5 ELSE
1502 6 ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
1503 6 IF J DIMENSION < 2 THEN
1504 7 SET J DIMENSION TO 1
1505 7 ELSE
1506 8 STORE I DIMENSION IN ENTRY
1507 8 ENDIF
1508 8 SET SIZE =I DIMENSION * J DIMENSION * TYPE LENGTH
1509 8 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1510 3 ELSE
1511 4 SET RETURN TO 3
1512 4 THEN:
1513 4 CALL XTCOM TO PROMPT FOR MAXIMUM SIZE
1514 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1515 4 ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
1516 4 SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
1517 3 ENDIF
1518 3 ENDF
1519 2 ENDF
1520 1 EXIT XLSPS
1521 2 :PRMERR: CALL XRMSG TO DISPLAY ERROR MESSAGE
1522 2 GO TO (:PRMPT1,:PRMPT2,:PRMPT3), RETURN
1523 1 END XLSPS
1 **D1** FDS MANAGER. SCHEDULED BY FDS CONFIGURATION MANAGER. ONE PROGRAM
2 **D1** PER SIGNED ON USER.
3
4 **D2** INPUT
5 **D2** FDS STATUS TABLE (SEE SDD 6.2.2)
6 **D2** FDS MANAGEMENT REQUESTS (SEE SDD 6.2.6.2)
7 **D2** FDS MANAGER RESPONSES (SEE SDD 6.2.6.3)
8
9 **D5** NOTES
10 **D5** USES XMGR, XMAVA, XMPAW, XMRO, XMGT, XDUMP
11 **D5** RMPIR,SLIBX,ANRQ,CHUNA,XEDR
12
13 **D5** XMAVA IS A MANAGER GLOBAL DATA AREA CONTAINING THE AWA AND VARIOUS
14 **D5** CONTROL VALUES
15
16 **D5** INTRODUCTION
17 **D5**
18
19 **D5** BEGIN XMGR
20 **D5**
21
22 **D6** BEGIN XMGR
23 **D6**
24 **D6** BEGIN XMGR
25 **D6**
26 **D6** BEGIN XMGR
27 **D6**
28 **D6** BEGIN XMGR
29 **D6**
30 **D6** BEGIN XMGR
31 **D6**
32 **D6** BEGIN XMGR
33 **D6**
34 **D6** BEGIN XMGR
35 **D6**
36 **D6** BEGIN XMGR
37 **D6**
38 **D6** BEGIN XMGR
39 **D6**
40 **D6** BEGIN XMGR
41 **D6**
42 **D6** BEGIN XMGR
43 **D6**
44 **D6** BEGIN XMGR
45 **D6**
46 **D6** BEGIN XMGR
47 **D6**
48 **D6** BEGIN XMGR
49 **D6**
50 **D6** BEGIN XMGR
51 **D6**
52 **D6** BEGIN XMGR
53 **D6**
54 **D6** BEGIN XMGR
55 **D6**
56 **D6** BEGIN XMGR
57 **D6**
58 **D6** BEGIN XMGR
59 **D6**
60 **D6** BEGIN XMGR
**II=REQUEST & NW IS LU**

**SET UP FOR FDS TERMINATION**

**EXIT TO :XMEND**

**ENDIF**

**ENDIF**

**ENDDO**

**:XMEND**

**IF DUMP OPTION THEN**

**CALL XUDMP TO DUMP OUR PARTITION**

**ENDIF**

**RELEASE DWA TRACK SPACE**

**FREE EXEC'S & PROCESSOR'S CLASS NUMBERS**

**CALL RNRQ SET LOCK ON STATUS TABLE**

**UNLOCK THE MANAGER FROM THE PARTITION**

**CALL SLIBR DISABLE**

**CLEAR EXEC'S ID SEGMENT**

**CALCULATE EQT ADDRESS FOR ATTENTION ID**

**CLEAR THE STATUS TABLE ENTRY**

**CALL SLIBR ENABLE**

**CALL RNRQ CLEAR THE LOCK ON THE STATUS TABLE**

**ISSUE MESSAGE *"XMDZ 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 ATTENTION ID**

**EXIT :EXEC TO THE RRE DISPATCHER**

**END :XMGR**
CALLING PROCEDURE

J SB XMXQT

**********

SEQUENCE TABLE EXECUTION FROM &SEQTB

**********

INPUT

XMAWA, XPARM, XVSTA

**********

OUTPUT

XMCTP, XPARM, XVSTA

**********

ROUTINES USED

CMUDP, EXEC, XMAG, XMIL, XPARW (XMSCH),

XMSST, XMTFN, $LIBR, $LIBW

**********
BEGIN XMF1T
CALL XMF1T TO FIND BSEG TO ENTRY
FIND ADDRESS OF BSEG
CALL XMSST TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
CALL XMSST TO CONVERT STARTING SEQUENCE NUMBER INTO CURRENT DISPLACEMENT
COMPUTE CURRENT ENTRY ADDRESS
DO UNTIL THE LAST TABLE ENTRY IS EXECUTED OR
UNTIL THE TERMINATION ENTRY IS EXECUTED
IF PROCESSOR REQUIRE AN INTERFACE TABLE (WORD 3 BIT 8 IS SET) THEN
EXIT TO :ERROR; IF INTERFACE TABLE NOT SPECIFIED (WORD A = 0) (PARMS = 1)
CALL XMF1T TO SEARCH AWA FOR INTERFACE TABLE (CHAIN A)
EXIT TO :ERROR; IF TABLE NOT FOUND (PARMS = 2)
IF TABLE NOT IN AWA, THEN
CALL INDIR TO RETRIEVE FROM AWA
EXIT TO :ERROR; IF SPACE IS NOT FOUND
ENDIF
EXIT TO :ERROR; IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PARMS = 3)
EXIT TO :ERROR; IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PARMS = 4)
EXIT TO :ERROR; IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PARMS = 5)
ENDIF
SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED
EXIT TO :ERROR; IF NOT FOUND
CALL BLIBR TO BE PRIVILEGED
SET CURRENT TASK TO THIS PROCESSOR
CALL BLIBX TO BE UN-PRIVILEGED
IF PROCESSOR REQUIRE AN INTERFACE TABLE THEN
CALCULATE LENGTH OF INTERFACE TABLE HEADER AND Specs
CALL EXEC TO CLASS I/O WRITE HEADER AND Specs
ENDIF
CALL EXEC TO SCHEDULE PROCESSOR WITH WAIT
CALL XPATCH TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XM1T TO FIND BSEG TO ENTRY
FIND ADDRESS OF BSEG
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARM = 1)
CALL XMG1 TO HONOR AWA REQUEST
ENDD0
CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
EXIT TO :ERROR; IF PROCESSOR REQUESTED TERMINATION (PARM = 8)
EXIT TO :ERROR; IF PROCESSOR ABENDED (PARM = -32768)
IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PARM = 3) THEN
EXIT TO :ERROR; IF RESET NUMBER IS ZERO
CALL XMSST TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY
EXIT TO :ERROR; IF RESET SEQUENCE NUMBER IS NOT FOUR
IF TERMINAL ENTRY WAS JUST EXECUTED THEN
SET UP PARMS 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 XM1XIT TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAM TO RESCHEDULE PROCESSOR
EXIT TO :ERROR; TO TERMINATE SEQUENCE
ENDIF
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDDO
171  SET PARM1 = 0 (NORMAL COMPLETION)
172  PERFORM :END: - NO RETURN EXPECTED
173  :ERROR1: SET PARM1 = 1
174  :ERROR8: SET PARM1 = 8 AND PARMS TO APPROPRIATE REASON CODE
175  :END:
176  CALL SLIBR TO BECOME PRIVLEDGED
177  SET CURRENT TASK IN PGR, STATUS TABLE, AND ANA TO EXEC
178  CALL SLIBX TO BECOME UN-PRIVLEDGED
179  CALL XMPAW TO POST EXEC AND WAIT FOR NEXT REQUEST
180  END XNXQT
CALLING PROCEDURE

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

INPUT

ADDRESS OF AREA BEING FREED

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

EXTERNAL SYMBOLS FROM XMATA

EXTERNAL SYMBOLS FROM XMATA

OUTPUT (EXTERNAL SYMBOLS FROM XMATA)

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 END LOOP
227 SET NEW FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
228 SET CURRENT FE'S FCP VALUE TO 'ADDR'
229 SET NEW FE BCP TO BCP HEAD VALUE
230 SET BCP HEAD TO 'ADDR'
231 END SEARCH
232 IF NEW FE BCP NOT = END-OF-CHAIN
233 THEN
234 PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
235 END IF
236 IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW
237 (OR CONSOLIDATED) FE, THEN
238 MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
239 ENDIF
240 1 END XMAFR
241 1 BEGIN MERGE
242 IF FE 1 IS ADJACENT TO FE 2
243 THEN
244 INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
245 SET FE 1 FCP TO VALUE OF FE 2 FCP
246 IF FE 1 FCP NOT = END-OF-CHAIN
247 THEN
248 SET FE 3 BCP TO ADDRESS OF FE 1
249 ELSE
250 SET BCP HEAD TO ADDRESS OF FE 1
251 ENDIF
252 ENDIF
253 1 END MERGE
CALLING PROCEDURE

JBB XNAGT
DEF #3
DEF OPTM
DEF SIZE

FIND A BLOCK OF FREE SPACE IN THE AWA AT LEAST 'SIZE' WORDS LARGE

INPUT
OPTH - INDICATOR OF WHICH FREE CHAIN TO SEARCH
0 = FORWARD POINTER CHAIN (FOR TDC SPACE)
1 = BACKWARD POINTER CHAIN (FOR DATA SPACE)
SIZE - NUMBER OF WORDS NEEDED (A MINIMUM OF 3 WORDS WILL BE
ALLOCATED EVEN IF 'SIZE' IS 1 OR 2)

EXTERNAL SYMBOLS FROM XNAMA
XMFCP, XMFC, XMFR

OUTPUT
A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
INDICATING NONE AVAILABLE
EXTERNAL SYMBOLS FROM XNAMA
XMFCP, XMFC, XMFR
XMFC, XMFR

NOTES
IF SOME FE EXACTLY 'SIZE' WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
ERROR RETURN (A-REG = -32768) IS TAKEN
USES .EMR

**********
295 1 BEGIN XMAGT
296 2 IF TOTAL FREE SPACE < MAX(SIZE,3)
297 2 THEN
298 3 CALL XMAGT1 PURGE DNA ELEMENTS FROM AWA
299 3 SET PHASE1 FLAG
300 3 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
301 4 RETURN VALUE= -32768
302 4 EXIT TO :XMAGR
303 4 ENDIF
304 4 ENDIF
305 2 PERFORM YMERC(OPTY,SIZE)
306 2 IF RETURN CODE IS NOT FOUND, THEN
307 3 CALL XMAGT
308 3 IF RETURN CODE IS NOT FOUND, THEN
309 4 RETURN VALUE= -32768
310 4 EXIT TO :XMAGR
311 4 ENDIF
312 3 RETURN VALUE= AREA
313 3 :XMAGR
314 3 ENDIF
315 2 CLEAR PHASE1 FLAG
316 1 END XMAGT
1 BEGIN XMSRC
2 SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FHEAD(OPTH)
3 START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
4 WHILE TOC SPACE FENCE HAS NOT BEEN CROSSED
5 EXIT IF FE SIZE = MAX('SIZE', 3)
6 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
7 DECHAIN FE
8 RETURN ADDRESS OF AREA
9 EXIT IF FE SIZE >= MAX('SIZE', 3) + 3
10 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
11 IF ALLOCATING FROM HEAD OF SPACE (OPTH = 0)
12 THEN
13 CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
14 RECHAIN 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 TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
26 INCREMENT TOC SPACE FENCE 0' 'SIZE'
27 ENDF
28 END XMSRC
347 1 *00  CALLING PROCEDURE
348 1 *00
349 1 *00 JSR XMANG
350 1 *00 DEF CLINO
351 1 *00
352 1 **********
353 1 *01 PROVIDE AWA MANAGEMENT BASED ON REQUEST LIST (SEE FDS SDD
354 1 *01 TABLE 6.2 - III)
355 1 *01
356 1 *01
357 1 **********
358 1 *02 INPUT
359 1 *02 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
360 1 *02
361 1 *02 REQUEST LIST (SEE TABLE 6.2 - III)
362 1 *02
363 1 **********
364 1 *03 OUTPUT
365 1 *03 ID SEGMENT Parameters (see Table 6.2 - IV)
366 1 *03 REQUEST LIST FIELD 8
367 1 *03
368 1 **********
369 1 *05 NOTES
370 1 *05 USES EXEC, XMAFR, XMAPT, XMAPK, XMIFN
371 1 *05 XMDIN, XMDAL, XMDOA, XMDST, XMDFR
372 1 *05
373 1 *05 NEITHER EXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
374 1 *05 FOR DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
375 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
376 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
1 BEGIN XWANG
2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER
3 CLEAR RETURN PARM1
4 CLEAR RETURN PARM2
5 INITIALIZE TO FIRST REQUEST CODE
6 DO UNTIL END OF LIST (0) 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 XMAPK TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC 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 IF REQUESTS WITH CLEAN CODE 17
17 THEN
18 SET A (BDWA) FROM XMDWA
19 SAVE HEADER AND DIRECTORY-SIZE
20 CLEAR XMDWA THRU XMDWB
21 BUILD AN FE AT XMDWA FOR AWA SIZE
22 CALL XMAGT TO ALLOCATE A TOC ENTRY FOR BDWA
23 CHAIN IN TOC ENTRY TO XMDWB
24 CALL XMAGT TO ALLOCATE SPACE FOR BDWA
25 SET LOCATION, SIZE, & KEY IN THE TOC
26 SET DIRECTORY ADDRESS AT XMDWA
27 CLEAR THE DIRECTORY
28 SET LU, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY
29 ELSE
30 CALL XMAGT TO SEARCH TOC FOR INDICATED ENTRY
31 CASE (:VERIFY, :VERALO, :VERALO, :RENAMES, :DELIVER, :DELIVER, :STORE,
32 (:RETRIEVE, :RETRIEVE) REQUEST CODE
33 :VERIFY:
34 IF ENTRY NOT FOUND
35 THEN
36 SET RETURN PARM1 AND PARM2 (2 & INDEX)
37 ENDIF
38 :VERALO:
39 IF ENTRY ALREADY EXISTS
40 THEN
41 IF ALLOCATE REQUEST (3)
42 THEN
43 SET RETURN PARM1 AND PARM2 (3 & INDEX)
44 ELSE
45 IF TYPE, SIZE AND I-DIM FIELDS DO NOT MATCH
46 THEN
47 SET RETURN PARM1 AND PARM2 (4 & INDEX)
48 ENDIF
49 ELSE
50 CALL XMAGT TO ALLOCATE TOC SPACE
51 IF CLASS EQ 3 OR 8 THEN
52 CHAIN IN NEW TOC ENTRY
53 SET DATA SPACE ADDRESS TO ZERO
54 ELSE
55 CALL XMAGT TO ALLOCATE DATA SPACE
IF SPACE NOT AVAILABLE
THEN
SET RETURN PARM1 AND PARM2 (1 & INDEX)
ELSE
IF DATA ELEMENT (CLASS 2)
THEN
IF CHARACTER STRING (TYPE 4 - 8)
THEN
INITIALIZE AREA TO BLANKS
ELSE
INITIALIZE AREA TO ZEROS
ENDIF
ENDIF
CHAIN IN NEW TOC ENTRY
ENDIF
IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE
THEN CALL XMDAL DMA ALLOCATION
ENDIF
ENDIF
ENDIF

:RENAME:
IF ENTRY NOT FOUND
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ELSE
CALL XMTFN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION
IF ENTRY FOUND
THEN
SET RETURN PARM1 AND PARM2 (3 & INDEX)
ELSE
CALL XMAGT TO ALLOCATE NEW TOC ENTRY
IF SPACE NOT AVAILABLE
THEN
SET RETURN PARM1 AND PARM2 (1 & INDEX)
ELSE
COPY OLD ENTRY Attributes INTO NEW ENTRY AND CHAIN IN TO TOC
DECHAIN OLD ENTRY
CALL XMIAF TO RETURN OLD ENTRY TOC SPACE TO FE POOL
ENDIF
ENDIF
ENDIF

:DELIVER:
IF ENTRY FOUND
THEN
GENERATE KEY 1 LESS THAN FOUND KEY
CALL XMTFN FOR GENERATED KEY
DECHAIN TOC ENTRY
CALL XMIAF TO RETURN TOC ENTRY SPACE TO FE POOL
CALL XMIAF TO RETURN DATA SPACE TO FE POOL
IF CLASS EQ 4 OR 6, THEN
CALL XMDA DMA DEALLOCATION
ENDIF
ELSE
IF DELETE REQUEST ($)
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ENDIF
ENDIF

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

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

1 *00  JSB XNTFM
2 *00  DEF **E
3 *00  DEF KEYS
4 *00
5 *00  ************
6 *00  *D1  EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'
7 *00  *D1
8 *00  ************
9 *00  *D2  INPUT
10 *00  *D2  KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC
11 *00  *D2
12 *00  *D2  EXTERNAL SYMBOLS FROM XNANA
13 *00  *D2  XNHD
14 *00  *D2
15 *00  ************
16 *00  *D3  OUTPUT
17 *00  *D3  A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR
18 *00  *D3  ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY
19 *00  *D3  WHERE 'KEYS' COULD BE CHAINED IN
20 *00  *D3
21 *00  ************
22 *00  *D5  NOTES
23 *00  *D5  USES .ENTR
24 *00  *D5
25 *00  ************
26 *00
27 *00  BEGIN XNTFM
28 *00  ISOLATE CLASS FROM KEY AND INDEX "O" A-PHONY CHAIN HEAD
29 *00  START SEARCH WHILE CHAIN POINTER (O) = END-OF-CHAIN (-32768)
30 *00  COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY
31 *00  EXIT IF MATCH
32 *00  RETURN ADDRESS OF ENTRY
33 *00  EXIT IF 'KEYS' < TOC ENTRY (EXPECTED ENTRY NOT IN CHAIN)
34 *00  RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET
35 *00  OR ELSE
36 *00  INDEX TO NEXT TOC ENTRY ON CHAIN
37 *00  END LOOP
38 *00  RETURN ADDRESS OF LAST (PREVIOUS) ENTRY WITH INDIRECT BIT SET
39 *00  END SEARCH
40 *00  END XNTFM
649 1 BEGIN XMPAW
650 2 *00 ENTRY: JSB XMPAW OR CALL XMPAW
651 2 *00 DEF RETURN ADDRESS
652 2 *02 INPUTS: XMPAW HAS REPLY
653 2 *02 OUTPUTS: XMPAW HAS REQUEST,
654 2 *03 AND XUSTO IS UPDATED.
655 2 *03 DO UNTIL GOOD PARMS RECEIVED OR TOP AT TERMINATES
656 2 *03 IF ABORT CURRENT FLAG CN(- STBLU) THEN
657 2 *03 SET ABORT FLAG IN XMPAW
658 2 *03 TURN OFF ABORT CURRENT FLAG
659 2 *03 ENDIF
660 2 *03 GET CURRENT TOP ASSOCIATED TASK(AT)
661 2 *03 SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
662 2 *03 JSB XVPAN
663 2 *03 DEF *#3 RETURN
664 2 *03 DFC 0 MANAGER CALL
665 2 *03 DEF XMPAN PARM FIELD
666 3 * THIS IS AN IMPLIED WAIT
667 3 ;XMSCH GET CURRENT XUSTO ENTRY (XUSTA)
668 3 IF THERE HAS BEEN A CALL TO PRIN (P1 FIELD IS NOT TOP AT) OR
669 4 ID SEGMENT IS DORMANT OR
670 4 ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
671 4 IF PARM IS NOT 0389, 32768 THEN
672 5 SET PARM TO -32768
673 5 PRINT ERROR "INVALID REQUEST"
674 4 ELSE
675 5 SET PARM FIELD FROM MANAGER'S ID SEGMENT
676 5 ENDIF
677 3 ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PAM)
678 4 IF PARM NOT 1 OR 2 THEN
679 5 CALL XMIL TO ABarbTOP AT
680 5 PRINT ERROR "INVALID REQUEST"
681 5 ELSE
682 5 SET PARM FROM CURRENT ID SEGMENT
683 5 ENDIF
684 3 ENDIF
685 2 ENDDO
686 1 END XMPAW
BEGIN XDBAL

1. DMA ALLOCATION
2. CALLING PROCEDURE
3. JSB XDBAL
4. FUNCTION
5. ALLOCATE DMA SPACE
6. INPUTS
7. A(TOC ENTRY OF THE AWA ELEMENT)
8. IN TREG
9. OUTPUTS
10. UPDATE TQC ENTRY FOR AWA
11. RETURNS BREG= ZERO= ALLOCATION COMPLETE
12. = MINUS- ERROR CONDITION
13. NOT'S
14. CALLS XNRD, XMBST
15. IF (E(DVA) .NE. 0) THEN
16. SET START-ADDRESS TO FIRST TRACK WORD IN DWA DIRECTORY
17. GET SIZE(IN WORDS FROM TOC ENTRY)
18. SIZE(IN SECTORS)= - (SIZE+63/64)
19. DO FOR NUMBER OF TRACKS IN DWA OR CSA SIZE .GE. SIZE
20. SET BITUM TO ZERO
21. DO UNTIL DWA SIZE .GE. SITE OR BITUM .GE. 96
22. CALL XRDB (0,BITNUM,START-ADDRESS)
23. STARTBIT=BITNUM
24. CALL XRDB (1,BITNUM,START-ADDRESS)
25. DVA-SIZE=BITNUM-STARTBIT
26. END-DO
27. IF BITUM .GE. 96, THEN
28. START-ADDRESS+START ADDRESS+1 TRACK ADDRESS
29. ENDIF
30. END-DO
31. IF D: AREA FOUND, THEN
32. START-ADDRESS=0
33. BITUM DEFINES THE TRACK & SECTOR
34. SET DISC ADDRESS & SIZE IN TOC ENTRY
35. CALL XPGT (TOC-ENTRY)
36. SET NORMAL RETURN
37. ELSE
38. ISSUE MESSAGE '***XDBL NO DWA SPACE REMAINING'
39. SET ERROR RETURN
40. ENDIF
41. ENDIF
42. END XDBAL.
802 1 BEGIN XHOST
803 2 *  DMA STORE DATA
804 1 0 CALLING PROCEDURE
805 2 *0 JSB XHOST
806 2 *0 FUNCTION
807 2 *0 UPDATE DMA ELEMENT WHICH CORRESPONDS
808 2 *0 TO THE DMA ELEMENT
809 2 *0 INPUTS
810 2 *0 TO ENTRY ADDRESS OF DMA ELEMENT
811 2 *0 IN TREG
812 2 *0 OUTPUTS
813 2 *0 UPDATE DMA ELEMENT ON DISC TRACK
814 2 *0 NOTES
815 2 *0 USES WRITE
816 2 *0 IF A(EDUA) .NE. 0, THEN
817 3 SET DISC ADDRESS FOR THE WRITE
818 3 SET DATA ADDRESS FOR THE WRITE
819 3 WRITE DATA
820 2 ENDIF
821 1 END XHOST
BEGIN XMDT

# CALLING PROCEDURE

FUNCTION

RETRIEVE DATA INTO AVA

INPUTS

(ENTRY) IN YREG

OUTPUTS

ADDRESS OF DATA IN THE TOC

REG=0, RETRIEVE SUCCESSFUL

MINUS, ERROR NO AVA DATA

NOTES

USES XMACG, XMDMA, EXECREAD

IF NO DATA DIRECTORY, THEN

SET ERROR CODE -5

ELSE

CALL XMAG

GET DATA SPACE

IF NO SPACE, THEN

SET ERROR CODE -1

ELSE

SET DATA ADDRESS IN TOC

GET DISC ADDRESS

READ DATA INTO AVA

SET RETURN CODE TO 0

ENDIF

ENDIF

END XMDT
BEGIN XMST

CALLING PROCEDURE

JSB XMST

FUNCTION

SET ALLOCATION & DEALLOCATION IN DWA DIRECTORY

INPUTS

Awa Toc Entry

OUTPUTS

Updates Dwa Directory

GET TRACK-ADDRESS FROM TOC

SUBTRACT START OF TRACKS FROM DWA DIRECTORY FOR RELATIVE TRACK

DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)

QUOTIENT IS NUMBER OF RELATIVE WORDS

REMAINDER IS BIT-POSITION(BP)

WORD ADDRESS=DWA ADDRESS+RELATIVE TRACK+RELATIVE WORDS

SUBTRACT BIT-POSITION FROM 16 GIVING NBFW(NUMBER BITS IN FIRST WORD)

GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)

BITS-NBFW=REMAILING BITS(RB)

IF RB = E, D, THEN

SET NUMBER OF WORDS(NW) TO ZERO

SET NUMBER OF BITS LAST WORD(NBLW) TO ZERO

SET NBFW TO BITS

ELSE

DIVIDE RB BY 16

SET NW TO QUOTIENT

SET NBFW TO REMAINDER

ENDIF

L-WAP DATA POINTED TO BY WORD ADDRESS

ROTATE LEFT(BP+NBFW-1 BITS) SAVE BP BITS & POSITION NBFW BITS

EXCLUSIVE OR SIGN BIT; (ON TO OFF; OFF TO ON)

IF NBFW .GT. 1, THEN

SHIFT RIGHT (ARITHMETIC) BY NBFW-1 PROPAGATE BITS

ENDIF

IF DATA IS NEGATIVE, THEN

SET FILL WORD TO -1

ELSE

SET FILL WORD TO 0

ENDIF

STORE WHERE WORD-ADDRESS POINTS

DO WHILE NW .GT. 0

WORD-ADDRESS=WORD-ADDRESS+$

SET FILL WORD INTO WHERE WORD-ADDRESS POINTS

ENDIF

IF NBLW .GT. 0, THEN

WORD-ADDRESS=WORD-ADDRESS+$

LOAD DATA POINTED TO BY WORD-ADDRESS

IF NBLW .GT. 1, THEN

ROTATE LEFT NBLW-1 BITS

ENDIF

EXCLUSIVE OR SIGN BIT

IF NBLW .GT. 1, THEN

SHIFT RIGHT ARITHMETIC BY NBLW-1

ENDIF

STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00  CALLING PROCEDURE
2 *00  JSB XMAPK
2 *00  DEF OPTION
2 *00  DEF PHASE 1 FLAG
2 *00  GENERAL COLLAPSE INTERFACE
2 *00
2 *00  OUTPUT
2 *00  RETURNS AREA ADDRESS FOR AREA FOUND
2 *00  RETURNS -32768 AREA NOT FOUND
2 *00
2 *00  NOTES
2 *00  USES XMKP1,XMKP2,XMKP3,XMSRC
2 *05
2 *05  ENDIF
2 *05  IF PHASE 1 FLAG NOT SET, THEN
2 *05  CALL XMKP3  PURGE DWA ELEMENTS FROM AWA
2 *05  PERFORM XMSRC(OPTN,SIZE)
2 *05  ELSE
2 *05  SET RETURN CODE TO NOT FOUND
2 *05
2 *05  ENDIF
2 *05  IF RETURN CODE IS NOT FOUND, THEN
2 *05  IF OPTN=1(BACKWARD CHAIN), THEN
2 *05  CALL XMKP2  PACK TOG CHAIN
2 *05  CALL XMSRC(OPTN,SIZE)
2 *05  ENDIF
2 *05  IF RETURN CODE IS NOT FOUND, THEN
2 *05  CALL XMKP3  PACK AWA DATA AREAS
2 *05  PERFORM XMSRC(OPTN,SIZE)
2 *05
2 *05  ENDIF
2 *05
2 *05
2 *05  SET RETURN VALUE TO RETURN CODE
1 END XMAPK
BEGIN XMPK2    CALLING PROCEDURE
JSB XMPK2

PHASE 2 OF COLLAPSE(COLLAPSE TOC ENTRIES)

OUTPUT
REORDERS TOC ENTRIES
UPDATES XMFCN( TOC FENCE ADDRESS)

NOTES
USES XMFCN,XMFCP,XMTPN,XMPSPM,XMSPR,XMSPZ

INCREMENT XMSPZ UPDATE PHASE 2 COUNT
DO WHILE XMFCN .GT. XMFCP AND XMFCP .NE. -32768
ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
WILL A TOC COMPRESS BE DONE.
GET FIRST-FREE(FREE),F) FROM XMFCP
GET OLDsz FROM THE FREE ELEMENT
NWSIZE=OLDSZ-8
IF HWNSZ .GT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
GET NEXT-FREE FROM FIRST-FREE's FCP
SET INTO XMFCP   DELETE FROM THE FORWARD CHAIN
IF XMFCP .NE. -32768, THEN IF NOT THE LAST FREE ELEMENT
SET NEXT-FREE's BCP TO -32768
ELSE
SET XMSPZ TO -32768 DELETE FROM BACKWARD CHAIN:
ENDIF
ELSE
OLDSZ=NWSIZE  UPDATE LENGTH IN FREE ELEMENT
ENDIF

NEXT=FA(ORSTP)+NWSIZE  THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
OLTOC=XMFCN-8  OLD TOC IS THE ENTRY MOVE THE FENCE
TMPKEY=OLTOC'S KEY - 1  CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
CALL XMTPN(TMPKEY)  FIND PREVIOUS
SET PTROC FROM ARG
COPY TOC ENTRY FROM OLTOC TO NEXT
STORE NEXTOC ADDRESS INTO PTROC'S CHAIN
CALL XMSPR(OLTOC,TOC-SIZE)  FREE THE OLD TOC ENTRY
ENDIF
END
1031 1 END XMPK2
BEGIN XMPK3
CALLING PROCEDURE
JSB XMPK3
FUNCTION
COMPRESS AWA DATA AREA BY MIGRATING DATA AREAS
TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
OUTPUT
UPDATES XMBCP,XMFPC,XMFNC,XMFRE,XMPK3
NOTES
USES XMAFR

INCREMENT XMPK3 (NUMBER OF PHASE 3)
IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
CALL XMPK3, INSURE TOC IS COMPRESSED
DO WHILE XMBCP .GT. XMFC . UNTIL 1 FREE AREA ADJACANT TO THE FENCE
IF TCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
SET HIGH-WATER TO XMFC
ELSE
MULIPLE AREAS
SET HIGH-WATER TO BCP OF LAST-FREE
ENDIF
SET LOW-WATER TO A(LAST-FREE)
PERFORM XNTCC(LOW-WATER,HIGH-WATER)
EXIT IF TOC-ADDRESS .EQ. 0
PERFORM XHAMV(TOC-ADDRESS)
ENDIF
END XMPK3
1064 1 BEGIN XMTSC TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR 0 (NONE FOUND).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XMMAA, THE
1070 2 * LAST TOC-ENTRY IS AT XMFC-0.
1071 2 TOC-ENTRY=X(MMMAA)
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 DO UNTIL TOC-ENTRY. GE. XMFC
1074 3 IF DATA ADDRESS IN TOC-ENTRY IS:
1075 4 ME 0, AND IS
1076 4 LT LOW-WATER, AND IS
1077 4 GT HIGH-WATER, AND IS
1078 4 GT TEST-AD,
1079 3 THEN
1080 4 TEST-AD=DATA ADDRESS
1081 4 TEST-TOC=TOC-ENTRY
1082 3 ENDFIF
1083 3 ADD 8 TO TOC-ENT
1084 2 ENDOO
1085 2 TOC-ADDRESS=TEST-TOC RETURN 0 OR A TOC ADDRESS
1086 1 END XMTSC
1088 1 BEGIN XMAV
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 XMAFR (DATA ADDRESS, SIZE)
1096 2 * GET FREE AREA FROM XMBEP
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 XMAV
1 BEGIN XPATHR
2 PERFORM XPMINI(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
3 EXIT TO :XPM13: (XPGET) IF PARAMETER IS OUT OF RANGE
4 EXTRACT NAME FROM INTERFACE TABLE
5 SET DSPTT TO ZERO
6 IF LITERAL PARAMETER (NAME IS ZERO)
7 THEN
8 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
9 ELSE
10 IF SUBSCRIPED (INTERFACE TABLE DISP OR S FIELDS ARE NON-ZERO)
11 THEN
12 PERFORM XPMIC(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
13 STORE TYPE, SIZE, IDIM AND DSPTT
14 ELSE
15 IF IDIM1 = 4 THEN
16 PERFORM XPMIFM(XPGET) TO QUALIFY FILE NAME
17 IF INPUT PARAMETER
18 THEN
19 CALL XPREQ TO RETRIEVE TOC ENTRY
20 COPY TYPE, SIZE, IDIM AND DSPTT FROM TOC ENTRY
21 ELSE
22 SET TYPE, SIZE AND IDIM TO ZERO
23 ENDIF
24 ENDIF
25 ENDIF
26 END XPATHR
**FORTRAN CALLING PROCEDURE**

**CALL** INPUT, INTL, ROWNF, N, INPS, INC(1), ..., 

**INPUT**
- **INPS** - ORILOWE NUMBER OF USER TERMS
- **INC** - ORILOWE NUMBER OF USER TERMS
- **N** - ORILOWE NUMBER OF USER TERMS
- **INTL** - ORILOWE NUMBER OF USER TERMS
- **IOUT** - ORILOWE NUMBER OF USER TERMS

**OUTPUT**
- **INPS** - ORILOWE NUMBER OF USER TERMS
- **INC** - ORILOWE NUMBER OF USER TERMS
- **N** - ORILOWE NUMBER OF USER TERMS
- **INTL** - ORILOWE NUMBER OF USER TERMS
- **IOUT** - ORILOWE NUMBER OF USER TERMS

**DESCRIPTION**
- **PROCEDURE** CALLS XTP/INPUT NORMAL PROCESSING (C1) ON THE INPUT STORAGE.
- **INPUT** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.
- **OUTPUT** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.
- **CLASS** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.
- **OUTPUT** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.
- **INTL** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.
- **IOUT** STORAGE CONTENTS MESSAGES FOR USE IN THE CALLING PROGRAM.

**GROUP**
- **CALL** - ORILOWE NUMBER OF USER TERMS
- **INPUT** - ORILOWE NUMBER OF USER TERMS
- **OUTPUT** - ORILOWE NUMBER OF USER TERMS
- **INTL** - ORILOWE NUMBER OF USER TERMS
- **IOUT** - ORILOWE NUMBER OF USER TERMS

**LABEL**
- **LABEL** - ORILOWE NUMBER OF USER TERMS
**PROCESSING LOOP**

155 1 **4** MP = VALUE OF -$P$, WHERE $P$ IS THE NUMBER OF PARAMETERS IN THE
156 1 **4** INTERFACE TABLE
157 1 **4** HANFL = ADDRESS OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
158 1 **4** ARRAY REST
159 1 **4** MARY = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
160 1 **4** SEQUENCE
161 1 **4** BEEST = EIGHT WORD ARRAY USED FOR CONSTRUCTING AW MANAGEMENT
162 1 **4** REQUESTS FOR XPRE
163 1 **4** USRFD = USER FILE IDENTIFIER CHARACTER (=IPARM(3))
164 1 **4**
165 1 **4** EXTERNAL VARIABLES (SEE XPRE)
166 1 **4**
167 1 **4** XPLCS
168 1 **4** XPLU
169 1 **4** XPRQD
170 1 **4**
171 1 ********
172 1 **4** NOTES
173 1 **4**
174 1 **5** USES ENTH, EXEC, XPRE, XPEIT, XUSTB
175 1 **5**
176 1 **5** XGET MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.
177 1 **5**
178 1 **5** SOME TRUST MUST BE USED BY XGET, XPUT, AND XPATH AND MCEOS
179 1 **5** TO BE INITIALIZED ONCE ONCE BY ANY OF THE TIME: ROUTINE:
180 1 **5** SINCE REQUESTS FOR INPUT DATA FROM THE WAV MAY BE PAGE BY UP TO
181 1 **5** EIGHT AT A TIME, XGET RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
182 1 **5** REQUESTED IN MULTIPLES OF EIGHT.
183 1 ********
FORTRAN CALLING PROCEDURE

CALL XPPUT (LU, INBUF, INTLNG, MRBUFF, N, NUMS, OUT(1), ..., OUT(N))

**********

XPPUT ALLOWS PROCESSORS TO STORE DATA INTO DATA ELEMENTS AND
DATA'S REFERENCED IN THE INTERFACE TABLE AS FOLLOWS:
1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC.
   IF FOUND, DATA IS STORED
   IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING
   TERMINATES.
2) DATA 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
   PREPARED WITH A / SYMBOL, SUFFIXED WITH A ONE
   CHARACTER USER CODE AND RETURNED SO THE CALLING
   PROGRAM CAN STORE DATA.

**********

** INPUT
1  ** LU - LOGICAL UNIT NUMBER OF USER TERMINAL
2  ** INBUF - INPUT/OUTPUT BUFFER OF 7* (# PARAMETERS + 13) WORDS,
3  ** ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
4  ** INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
5  ** FIRST USE TO CAUSE INITIALIZATION.
6  ** INTLNG - LENGTH OF INBUF
7  ** MRBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
8  ** WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH
9  ** AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
10 ** N - NUMBER OF PARAMETERS TO BE STORED. IF N=0, ALL
11 ** OUTPUT PARAMETERS ARE ASSUMED TO BE SUPPLIED.
12 ** NUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
13 ** INTERFACE TABLE REFERENCING DATA ELEMENTS AND ODBE'S
14 ** OUTPUT ORDER OF THE DATA. THE ORDER OF THE
15 ** OUT(N) - LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE
16 ** STORED FROM.
17 ** SHARE EXTERNAL SYMBOL
18 ** XPGF - FLAG INDICATING XPGT/XPPUT NORMAL PROCESSING (-1) OR
19 ** XPGT/XPPUT SPECIAL PROCESSING BY-PASSING SUBSCRIPT
20 ** RESOLUTION (2=10)

**********

** OUTPUT
1  ** NONE
2  ** NUMS
3  ** INTERNAL VARIABLES
4  ** (SEE XPGT)
5  ** (SEE XPGT)

** NOTES
1  ** (SEE XPGT)
2  ** (SEE XPGT)
243 1 BEGIN XGET
244 2 SET FOR 'GET'
245 2 PERFORM ACCESS TO RETRIEVE DATA
246 1 END XGET
247 1 BEGIN XPUT
248 2 SET FOR 'PUT'
249 2 PERFORM ACCESS TO STORE DATA
250 1 END XPUT
251 1 BEGIN ACCESS
252 2 PERFORM SPINL TO INITIALIZE GLOBALS AND INTERFACE TABLE
253 2 DO FOR EACH PARAMETER REQUESTED
254 3 IF SELECTED PARAMETER IS OUT OF RANGE
255 4 CALL XPROG TO PURGE QUEUED REQUESTS
256 5 EXIT TO XPE13:
257 6 ENDIF
258 3 IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/ 'PUT' PROCESSING
259 3 THEN
260 4 CALL XPROG TO PURGE QUEUED REQUESTS
261 5 EXIT TO :ERROR2:
262 5 ENDIF
263 5 IF OVER RUNNING CALLING SEQUENCE
264 5 THEN
265 5 CALL XPROG TO PURGE QUEUED REQUESTS
266 5 EXIT TO :XPE13:
267 5 ENDIF
268 5 IF PROCESSING FOR "GET"
269 5 THEN
270 6 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
271 7 THEN
272 8 IF INPUT IN LITERAL FORM
273 9 THEN
274 10 BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT
275 11 ELSE
276 12 IF NORMAL XGET/PUT PROCESSING (XPSPF = -1)
277 13 THEN
278 14 IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
279 15 THEN
280 16 PERFORM XPSCB TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
281 17 ELSE
282 18 DISPLACEMENT IS ZERO
283 19 ENDIF
284 20 ELSE
285 21 USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
286 22 ENDIF
287 23 ENDIF
288 23 ENDIF
289 23 ENDIF
290 23 CALL XPROG TO QUEUE RETRIEVAL AND STORAGE OF INPUT
291 24 ELSE PARAMETER IS OROE (CLASS 3)
292 25 THEN
293 26 BUILD REQUEST WITH DE NAME AND DISPLACEMENT
294 27 CALL XPROG TO IMMEDIATELY RETRIEVE TOC ENTRY
295 28 PERFORM XPFW TO CONSTRUCT AND STORE QUALIFIED FILE NAME
296 29 "STORE FILE ATTRIBUTES"
297 30 ENDIF
298 31 ELSE PROCESSING FOR "PUT"
299 32 THEN
300 33 IF NORMAL XGET/PUT PROCESSING (XPSPF = -1)
301 34 THEN
302 34
IF SUBSCRIPTED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRDE
CALL XPREQ TO QUEUE ALLOCATION
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AHA
CALL XPREQ TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS ORDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD RF IST TO REALLOCATE ORDE WITH NEW ATTRIBUTES
EXIT TO XERR: IF FILE TYPE NOT 1-13, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
CALL XPREQ TO QUEUE REALLOCATION OF ORDE
ENDIF
ENDIF
ENDDO
CALL XPREQ TO COMPLETE QUEUED REQUESTS
EXIT ACCESS
:ERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:ERR13: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:ERR14: CALL XPREQ TO PURGE QUEUED REQUESTS
:ERR15: TERMINATE PROCESSOR FOR INVALID ORDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
BEGIN XPRIN
  INITIAlIZE GLOBAL VALUES FROM L1 AND XVSTB
  TERMINATE PROCESSOR WITH XP10 ERROR IF L1 NOT IN XVSTB
  IF INTERFACE TABLE BUFFER NOT INITIALIZED
    THEN
      RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER
      IF RETRIEVAL NOT SUCCESSFUL
        THEN
          TERMINATE PROCESSOR WITH 'XP10 PROCESSOR INITIALIZATION ERROR'
      ENDIF
    ENDIF
  EXIT TO XPE13: IF N < 0
END XPRIN

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

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

1 *DO
2 CALL XPGET (LU, INTBUF, INTLNG, MBUFF, INUM, IN, SIZE, DISP)
3
4 ***********
5
6 *DO
7 XPGET ALLOCS THE CALLING PROGRAM TO RETRIEVE A SPECIFIED AMOUNT
8 OF DATA FROM AN INPUT PARAMETER BEGINNING AT ANY POINT IN THE
9 PARAMETER ARRAY.
10
11 ***********
12
13 *DO
14 INPUT
15
16 *DO
17 LU - LOGICAL UNIT NUMBER OF USER TERMINAL
18
19 *DO
20 INTBUF - INPUT/OUTPUT BUFFER OF (N PARAMETERS + 1) WORDS,
21 ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
22 INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
23 FIRST USE TO CAUSE INITIALIZATION
24
25 *DO
26 INTLNG - LENGTH OF INTBUF
27
28 *DO
29 MBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
30 WITH THE FD's MANAGER. MAY BE USED AS A SCRAWL AREA BY
31 THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS.
32
33 *DO
34 INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM
35 WHICH TO RETURN DATA
36
37 *DO
38 SIZE - TOTAL NUMBER OF WORDS TO BE RETURNED
39
40 *DO
41 DISP - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER
42 AT WHICH TO BEGIN DATA RETRIEVAL. A VALUE OF ZERO
43 INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER
44 IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT
45 BE SPECIFIED, I.E., RETRIEVL FROM AN AREA PREVIOUS TO
46 THE USERS'SUBSCRIPTS IS NOT SUPPORTED.
47
48 ***********
49
50 *DO
51 OUTPUT
52
53 *DO
54 IN - ARRAY OF AT LEAST SIZE WORDS INTO WHICH INPUT DATA IS TO
55 BE STORED.
56
57 *DO
58 INUM
59
60 *DO
61 ***********
62
63 *DO
64 INTERNAL
65
66 *DO
67 BDSUMP - DISPLACEMENT FROM THE REAL ORIGIN OF THE DATA ELEMENT AS
68 A RESULT OF USER SUBSCRIPTING. DISP IS ADDED TO THIS
69 VALUE IN THE REQUEST TO THE MANAGER
70
71 *DO
72 JGPT - DISPLACEMENT INTO JGTP DEPENDING ON WHETHER GET (D) OR
73 PUT (I) PROCESSING IS INVOLVED
74
75 *DO
76 JGTP - TWO WORD VECTOR CONTAINING 'JSB XPGET' OR 'JSB XPPUT'.
77
78 *DO
79 INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGET AND
80 XPPUT CALLS.
81
82 ***********
83
84 *DO
85 NOTES
86
87 *DO
88 USES EXEC, ENTR, XPATR, XPE13(XPATR), XPGET(XPATR), XPPUT(XPATR)
89
90 *DO
91 USES ENTR, EXEC, XPATR, XPER (XPATR), XPE13(XPATR), XPGET(XPATR),
92 XPPUT(XPATR)
93
94 ***********
95
96 5-202
FORTAN CALLING PROCEDURE

CALL XPTI (LU, INBUF, INTLN, MRBUF, INUM, OUT, SIZE, DISP)

*********

XPTI 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

INTLF - 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

INTLN - LENGTH OF INTLB

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

*********

NOTES

XPTII IS AN ENTRY POINT INTO XPTII

*********
1 BEGIN XPGETI
2 SET FOR GET PROCESSING
3 PERFORM XPPXI TO RETRIEVE DATA
4 END XPGETI
5 BEGIN XPPTI
6 SET FOR PUT PROCESSING
7 PERFORM XPPXI TO STORE DATA
8 END XPPTI
9 BEGIN XPPIXI
10 SET XPPFI(XPGET) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
11 CALL XPATR TO ASSURE INITIALIZATION OF INTBUF AND RETURN BASE DISPLACEMENT
12 EXIT TO :XE13; (XPATR) IF INDICATED PARAMETER IS A DROG FILE
13 SAVE INTERFACE TABLE ENTRY CLASS/TITLE WORD, FLAG/DISP WORD AND SIZE WORD
14 IF LITERAL (NAME = 0)
15 THEN
16 EXIT TO :ERR15: IF DISP + SIZE > SAVED SIZE
17 ELSE
18 INCREMENT BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
19 SET TYPE FIELD TO FREE
20 EK01F
21 CLEAR ENTRY SUBSCRIPT BIT
22 SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
23 SET SIZE FIELD TO SIZE
24 CALL XPGET/XPPUT TO TRANSFER DATA
25 RESTORE ORIGINAL INTERFACE TABLE ENTRY
26 RESTORE XFGPF(XPGET) TO NOMINAL VALUE
27 EXIT XPPXI
28 :ERR15: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
29 END XPPXI
CALLING PROCEDURE
JSB XPREQ
DEF **+1 WHERE W IS THE NUMBER OF ACTUAL ARGUMENTS
DEF OPTN REQUIRED OPTION
DEF REQST REQUIRED REQUEST
DEF ADRES OPTIONAL ADDRESS (SEE OPTN)

**********
XPGET/XPPUT BUFFERED AWS 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
REQT - EIGHT WORD REQUEST (SEE SDD 6.2-6.2)
ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT
VALUES USED ONLY WHEN OPTN(RIGHT BIT) = 0
EXTERNAL SYMBOLS
XPLU - TERMINAL LOGICAL UNIT NUMBER
XPDB - ADDRESS OF 64 WORD MANAGER REQUEST BUFFER
XPCLS - ADDRESS OF 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
ADD0 - ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDS)
ADD1 - TABLE OF ADDRESSES FOR STORAGE OF Fetched DATA
APRMS - ADDRESS OF PARMS AREA
PARMS - PARAMETER RETURN AND SCRATCH AREA
PTR - SAVE AREA FOR ADDS AND XPDB POINTER

**********
USES .ENTR, EXEC., XPIT, XPAW
IF DATA IS TO BE TRANSFERRED TO/FROM 'ADRES', THEN 'REQT' SIZE FIELD ( WORD 6) MUST BE SET.

**********
1 BEGIN XPREG
2 MOVE REQUEST INTO BUFFER
3 IF OPTION IS TO TRANSFER DATA
4 THEN
5 IF REQUEST IS TO RETRIEVE DATA (B)
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 MANAGER WITH REQUEST FOR MAN 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 (B, 9 OR 16)
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 ENDIF
41 ENDDO
42 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
43 ENDIF
44 ENDIF
45 1 END XPREG
46
608 1 CD----------
609 1 CD0       FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE
610 1 CD0       CALL XPRES (LUN,PRMLEN,PROMPT,TYPE,DATLEN,IDIM,DATA,RETC)
611 1 CD0       CALL XPROM, DECODES COMMUNICATIONS BUFFER LOOKING FOR A SPECIFIC
612 1 CD0       "TYPE" AND STORES IT IN "DATA" FOR "DATLEN" LOGICAL ELEMENTS
613 1 CD0       
614 1 CD0       INPUTS FROM CALLING SEQUENCE:
615 1 CD0           LUN - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
616 1 CD0           PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
617 1 CD0           STRING USED FOR THE USER PROMPT
618 1 CD0           PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
619 1 CD0           USED AS THE USER PROMPT
620 1 CD0           TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE
621 1 CD0           EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
622 1 CD0           0 - FREE
623 1 CD0           1 - INTEGER
624 1 CD0           2 - REAL
625 1 CD0           3 - DOUBLE PRECISION
626 1 CD0           4 - CHARACTER STRING LENGTH 2
627 1 CD0           5 - CHARACTER STRING LENGTH 4
628 1 CD0           6 - CHARACTER STRING LENGTH 8
629 1 CD0           7 - CHARACTER STRING LENGTH 16
630 1 CD0           8 - CHARACTER STRING LENGTH 32
631 1 CD0           9 - CHARACTER STRING LENGTH 64
632 1 CD0           DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN
633 1 CD0           THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE
634 1 CD0           NUMBER OF WORDS.
635 1 CD0           IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA
636 1 CD0           AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0
637 1 CD0           IF IT IS A VECTOR.
638 1 CD0       
639 1 CD0       OUTPUTS FROM CALLING SEQUENCE:
640 1 CD0           DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE)
641 1 CD0           AREA TO CONTAIN THE USER'S RESPONSE
642 1 CD0           RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO
643 1 CD0           CALLER:
644 1 CD0           0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
645 1 CD0           1 - USER ENTERED 2 CONTENTS OF BUFFER UNPREDICTABLE.
646 1 CD0           2 - USER ENTERED A CR, THERE IS NO RESPONSE.
647 1 CD0           3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
648 1 CD0           34 CHARACTERS OR 17 WORDS.
649 1 CD0           4 - PARAMETER LIST IS INVALID. EITHER:
650 1 CD0           A. IDIM LESS THAN ZERO
651 1 CD0           B. DATLEN LESS THAN ONE
652 1 CD0           C. TYPE NOT SPECIFIED.
653 1 CD0       
654 1 CD0       INTERNAL VARIABLES
DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT ELEMENT IS TO BE STORED.
EFFTP - THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER STRING RESPONDER. IF TYPE IS FREE, 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 - 4 WORD ARRAY FOR STACK OF REPEAT COUNTS.
STKTOK - 4 WORD ARRAY FOR STACK OF TOKEN POINTERS.
TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED.
TYPEI - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE.

------------------------

SUBROUTINES AND FUNCTIONS CALLED:
EXEC, XPRDS, ERRMSG

POL ROUTINES INCLUDED:
XPRDS, STRING, SUBSCR, REPET, ERRMSG

------------------------
1 BEGIN XPRDS
2 IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3 CALL XPRDM TO PROMPT USER AND DECODE RESPONSE
4 IF XPRDM RETURN CODE IS ZERO THEN
5 DO UNTIL EOS TOKEN IS FOUND
6 CASE TOKEN (:NUMBER,:NUMBER,:NUMBER,:STRING,:NULL,)
7 ,:SUBSCR,:REPEP,:
8 :NUMBER:
9 ERREXIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERRMSG
10 SET MESSAGE NUMBER TO XPO2
11 ERREXIT IF THERE IS NO ROOM IN DATA AREA PERFORM ERRMSG
12 SET PREVIOUS TOKEN TO DATA
13 CALL XMOV TO MOVE DATA INTO DATA AREA
14 INCREMENT POINTER IN DATA AREA
15 INCREMENT POINTER TO NEXT TOKEN
16 :STRING:
17 PERFORM STRING
18 :NULL:
19 SET MESSAGE NUMBER TO XPO2
20 ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERRMSG
21 SET PREVIOUS TOKEN TO DATA
22 INCREMENT POINTER IN DATA AREA
23 INCREMENT POINTER TO NEXT TOKEN
24 :SUBSCR:
25 PERFORM SUBSCR
26 :REPEP:
27 PERFORM REPET
28 ENCASE
29 DO UNTIL TOKEN IS NOT A CLOSED PARENTHESES
30 IF STACK IS NOT EMPTY AND
31 (PREVIOUS TOKEN ISDATA AND TOP OF STACK PAREN FLAG =0) OR
32 (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG =1) THEN
33 DECREMENT TOP OF STACK REPEAT COUNT BY 1
34 IF TOP OF STACK REPEAT COUNT > 0 THEN
35 RESET TOKEN POINTER TO TOP OF STACK INDEX
36 ELSE
37 POP TOP ENTRY ON STACK
38 IF TOKEN IS A CLOSED PAREN THEN
39 INCREMENT TO NEXT TOKEN
40 ENDFI
41 7 ENDIF
42 6 ENDIF
43 5 ENDOF
44 4 ENDO.
46 3 ENDFI
47 2 SET XPRDM RETURN CODE = XPRDM RETURN CODE
48 1 EL5
49 3 SET XPRDM RETURN CODE TO 'INVALID DEPARTMENT LIST'
50 1 ENDFI
51 1 END XPRDS
FORTAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE

CALL XPROM (LU, PRMLEN, PROMPT, COMLEN, COMBUF, RETC)

- PRMLEN - (INTEGER, 1 WORD) RESPONSE TO TOKENS IN "COMBU" AND PASSES
- PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
- COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE
- COMBINATIONS BUFFER (COMBU)

INPUTS FROM CALLING SEQUENCE:
- LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
- PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
- PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
- COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE COMMUNICATIONS BUFFER (COMBU)

OUTPUTS FROM CALLING SEQUENCE:
- COMBUF - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE ENCODED USER RESPONSE
- RETC - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO THE CALLING PROGRAM AS FOLLOWS:
  0 - NORMAL RETURN BUFFER CONTAINS USERS RESPONSE
  1 - USER ENTERED X CONTENTS OF BUFFER UNPREDICTABLE.
  2 - USER ENTERED CR BUFFER CONTAINS NO INFORMATION
  3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS 34 CHARACTERS OR 17 WORDS.

INTERNAL VARIABLES:
- COUNT - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN A CHARACTER STRING AND NUMER DIGITS IN A NUMBER.
- DOLLT - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE
- DOLLVD - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE
- VALUE
- FLCOM - COMMA FLAG
- LFLCOM - CONTINUANCE FLAG
- FLGPOW - POWER FLAG
```assembly
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 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
8 CALL EXEC TO READ RESPONSE
9 CALL XRPK TO CONVERT A2 RESPONSE TO R1 FORMAT
10 IF NUMBER OF WORDS READ IS NOT ZERO THEN
11 SET COMMA FLAG ON
12 DO WHILE COMMA FLAG IS OFF
13 IF INPUT BUFFER IS COMPLETELY SCANNED THEN
14 SET COMMA FLAG ON
15 IF COMMA FLAG IS ON THEN
16 CALL EXEC TO WRITE CONTINUE
17 GO TO :COMLOP:
18 ELSE
19 ERREXIT IF COMBUF IS FULL PERFORM COMFUL
20 IF INPUT CHARACTER IS A COMMA THEN
21 IF COMMA FLAG IS ON THEN
22 ERREXIT IF COMBUF CANNOT HOLD TOKEN PERFORM COMFUL
23 STORE NULL FIELD TOKEN IN COMBUF
24 INCREMENT WORDS IN COMBUF BY 1
25 INCREMENT TOKENS IN COMBUF BY 1
26 ENDIF
27 SET COMMA FLAG ON
28 SET NEXT INPUT CHARACTER
29 ELSE
30 SET COMMA FLAG OFF
31 PERFORM TOKENS
32 ENDIF
33 ENDIF
34 ELSE
35 IF CONTINUE FLAG IS OFF THEN
36 SET XPRDM RETURN CODE TO SAY USER ENTERED CR
37 ENDIF
38 ENDIF
39 ELSE
40 ELSE
41 SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
42 ENDIF
43 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 ("(" + " + -")

: 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 COMPL
SET 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
SET INTEGER = 0
PERFORM DECP

: G:
IF INPUT CHARACTER IS A - THEN
SET NEGATIVE NUMBER FLAG ON
ENDIF
GET NEXT CHARACTER
IF INPUT CHARACTER IS A DIGIT THEN
PERFORM DIGIT
ELSE
IF INPUT CHARACTER IS A . THEN
GO TO : F:
ELSE
PERFORM INVAL - NO RETURN
ENDIF
ENDIF
ENDIF
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     MOVE CHARACTER INTO TEMPORARY BUFFER (#CHARACTERS)
10 IF INPUT CHARACTER IS A QUOTE THEN
11     GET NEXT CHARACTER
12     ENDIF
13 GET NEXT CHARACTER
14 END QUOTE
15 END DO
16 ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
17 EXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
18 ERREXIT 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 XRCPK 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 "1" THEN
1023 5 PERFORM DECP
1024 6 ELSE
1025 7 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1026 8 PERFORM END
1027 9 ELSE
1028 10 IF INPUT CHARACTER IS AN "R" THEN
1029 11 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1030 12 ERREXIT IF INTEGER IS ZERO PERFORM INVAL
1031 13 STORE REPEAT TOKEN IN COMBUF
1032 14 INCREMENT #WORDS IN COMBUF 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 1
1045 27 1
1046 28 BEGIN DCOL
1047 29 SET INTEGER=D
1048 30 SET COUNTER=D
1049 31 DO WHILE CHARACTER IS A DIGIT AND
1050 32 WHILE INPUT BUFFER IS NOT EXHAUSTED
1051 33 SET INTEGER= (INTEGER * 10) + INPUT CHARACTER - 48
1052 34 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1053 35 INCREMENT COUNTER BY 1
1054 36 GET NEXT CHARACTER
1055 37 ENDDO
1056 38 ENDDO
1057 39 END DCOL
1059 1 BEGIN DECPT
1060 2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061 3 GET NEXT CHARACTER
1062 3 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063 4 IF INPUT CHARACTER IS A DIGIT THEN
1064 4 PERFORM DEC
1065 4 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066 3 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067 3 ENDIF
1068 3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069 4 PERFORM EORD
1070 3 ELSE
1071 4 PERFORM REAL
1072 3 ENDIF
1073 2 ELSE
1074 3 PERFORM REAL
1075 2 ENDIF
1076 1 END DECPT
1 BEGIN INTEGER
2 EXPREXIT IF NEXT TOKEN IS NOT A COMMA AND
3 EXPREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
4 EXPREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
5 EXPREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
6 CONVERT DOUBLE TO INTEGER
7 EXPREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
8 IF NEGATIVE NUMBER FLAG IS ON THEN
9 SET INTEGER = -INTEGER
10 ENDF
11 STORE INTEGER TOKEN IN COMBUF
12 INCREMENT #WORDS IN COMBUF BY 2
13 INCREMENT #TOKENS IN COMBUF BY 1
14 END INTEGER
15 *
16 *
17 BEGIN REAL
18 EXPREXIT IF NEXT TOKEN IS NOT A COMMA AND
19 EXPREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
20 EXPREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
21 EXPREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
22 SET REAL = DOUBLE * 10 ** POWER
23 EXPREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
24 IF NEGATIVE NUMBER FLAG IS ON THEN
25 SET REAL = -REAL
26 ENDF
27 STORE TOKEN IN COMBUF
28 INCREMENT #WORDS IN COMBUF BY 3
29 INCREMENT #TOKENS IN COMBUF BY 1
30 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
1174 1 CD0 FORTRAN CALLING PROCEDURE
1175 1 CD0 CALL XPXIT (LU, RPAMS)
1176 1 CD0
1177 1 CD0
1178 1 CD0
1179 1 CD0 TERMINATE A PROGRAM AFTER WAITING ON ANY BUFFERED I/O TO LU TO
1180 1 CD0 COMPLETE THEN PASS PARAMETERS BACK TO THE FATHER TASK.
1181 1 CD0
1182 1 CD0
1183 1 CD0
1184 1 CD2 INPUT
1185 1 CD2 LU - LOGICAL UNIT OF TERMINAL OR OTHER OUTPUT DEVICE FOR WHICH
1186 1 CD2 PRINT MAY NOT BE COMPLETED. A VALUE OF ZERO WILL
1187 1 CD2 INDICATE NO I/O WAIT
1188 1 CD2 RPAMS - ARRAY OF FIVE PARAMETERS TO BE RETURNED TO FATHER TASK
1189 1 CD2
1190 1 CD2
1191 1 CD3 OUTPUT
1192 1 CD3 RPAMS IS RETURNED TO THE FATHER TASK
1193 1 CD3
1194 1 CD3 NOTES
1195 1 CD5 USES EXEC, PRTM
1196 1 CD5
1197 1 CD5
1198 1 CD5 XPXIT DOES NOT RETURN TO THE CALLER.
1199 1 CD5
1200 1 CD5
1201 1 * 1202 1 * 1203 1 * 1204 1 *
1205 1 BEGXPXIT 1 XPXIT
1206 2 IF LU IS NON-ZERO 1 XPXIT
1207 2 THEN 1 XPXIT
1208 3 WAIT ON ANY CLASS I/O TO COMPLETE (CLASS I/O CONTROL THEN GET)
1209 2 ENDIF 1 XPXIT
1210 2 RETURN PARAMETERS TO FATHER 1 XPXIT
1211 2 TERMINATE PROGRAM 1 XPXIT
1212 1 ENDPXIT 1 XPXIT
1 **0** FORTRAN CALLING PROCEDURE

2 1 **0** CALL XRBIT (BIT, BITHUM, STRING)

3 ********

4 7 1 **0** BIT SET/CLEAR ROUTINE FOR MULTIPLE WORD BIT STRINGS. SET THE

5 9 1 **0** VALUE OF BIT NUMBER 'BITHUM' IN STRING 'STRING' TO 'BIT'.

8 10 1 **0**

11 ********

12 11 1 **0** INPUT

13 13 1 **0** BIT - INTEGER VALUE OF ZERO OR ONE TO BE SET IN BIT NUMBER

14 14 1 **0** 'BITHUM' OF 'STRING'

15 15 1 **0** BITHUM - INTEGER BIT NUMBER OF BIT TO BE SET/CLEARED (FIRST BIT OF

16 16 1 **0** STRING IS BIT NUMBER ZERO)

17 17 1 **0** STRING - BIT STRING OF AT LEAST 'BITHUM' BITS IN LENGTH INTO WHICH

18 18 1 **0** 'BIT' IS TO BE SET

19 19 1 **0**

20 ********

21 20 1 **0** OUTPUT

22 22 1 **0** STRING - BIT NUMBER 'BITHUM' HAS THE VALUE OF 'BIT'

23 23 1 **0**

24 ********

25 23 1 **0** NOTES

26 24 1 **0** USES .ENTR, XRSET

27 25 1 **0**

28 26 1 **0**

29 27 1 **0**

30 28 1 **0**

31 29 1 **0**

32 30 1 **0** BEGIN XRBIT

33 31 2 **0** TRANSFER CALLING SEQUENCE

34 32 2 **0** COMPUTE ADDRESS OF WORD CONTAINING BITHUM

35 33 2 **0** COMPUTE BIT NUMBER WITHIN WORD

36 34 2 **0** CALL XRSET TO SET/CLEAR BIT

37 35 2 **0**

38 36 1 **0** END XRBIT
INTEGER FUNCTION
XRCPR(LENGTH, ARRAY1, ARRAY2)

1 **01**
COMPARE 'LENGTH' WORDS OF 'ARRAY1' TO 'ARRAY2'. IF THE ARRAYS
ARE IDENTICAL, A VALUE OF 0 IS RETURNED. IF THE COMPARISON
FAILS, A VALUE IS RETURNED INDICATING WHETHER THE FIRST
UNEQUAL ELEMENT OF ARRAY1 IS LESS THAN ARRAY2 (-1) OR GREATER (+1)

**INPUT**
LENGTH - POSITIVE INTEGER NUMBER OF WORDS TO BE COMPARED
ARRAY1 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY2
ARRAY2 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY1

**OUTPUT**
FUNCTION VALUE - 0, ARRAYS EQUAL
-1, ARRAYS NOT EQUAL, ARRAY1 < ARRAY2
+1, ARRAYS NOT EQUAL, ARRAY1 > ARRAY2

**NOTES**
USES .ENTR

BEGIN XRCPR
TRANSFER CALLING SEQUENCE
SETUP COMPARE
COMPARE ARRAY1 AND ARRAY2
RETURN RESULT FLAG
END XRCPR
FORTRAN CALLING PROCEDURE.

CALL XRD18 (DOUBLE, ASCII)

CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN 'D10.1Y' FORMAT

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

OUTPUT

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, IAND, IDINT, IOR, KCVT, XREXT

BEGIN XRD18

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 XRD18
CD0          FORTRAN CALLING PROCEDURE
CD3          CALL XREQ
CD0          CALL XREQ
CD0          CALL XREQ
CD2          COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPTR
CD2          ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER
CD0          CALL XREQ
CD0          CALL XREQ
CD3          COMMON XE - REQPTR
CD3          CLASS I/O WRITE/READ TO CLASS 'CLASNO'
CD3          REQBUF AND RESPONSE IS PRINTED IF REQUESTED
CD3          CALL XREQ
CD5          COMMUNICATES WITH FDS MANAGER FATHER TASK
CD5          USES EXEC, IAND, XRMOV, XRMIG, XRO6, XRSFL, XUDPL, XVPAW
CD5          CALL XREQ
CD5          CALL XREQ
CD5          BEGIN XREQ
CD5          BEGIN TRACE
CD5          OUTPUT TRACE TO MANAGER
CD5          REQUEST AREA MANAGEMENT AND WAIT FOR RESPONSE
CD5          RETRIEVE MANAGER RESPONSE
CD5          BEGIN TRACE
CD5          RETURN RESPONSE IN REQPTR
CD5          BEGIN XREQ
CD5          BEGIN TRACE
CD5          IF TRACE REQUESTED
CD5          THEN
CD5          DO FOR EACH REQUEST
CD5          CALL XUDPL TO FORMAT LINE
CD5          OUTPUT LINE
CD5          ENDDO
CD5          OUTPUT PARM1 AND PARM2
CD5          ENDIF
CD5          END TRACE
INTEGER FUNCTION
XREXT(START, LENGTH, SOURCE)

**********
#1 EXTRACT 'LENGTH' BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
AND RIGHT ADJUST

**********
#2 INPUT
#2 START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
#2 (SIGN BIT = 0)
#2 LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
#2 SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED

**********
#5 NOTES
#5 USES .ENTR

**********

BEGIN XREXT
200 2 TRANSFER CALLING SEQUENCE
201 2 IF START NOT = 0
202 2 THEN
203 3 CONSTRUCT SHIFT
204 3 LOAD A WITH SOURCE
205 3 SHIFT BA LEFT START BITS
206 2 ELSE
207 3 LOAD A WITH SOURCE
208 2 EN0IF
209 2 SAVE A
210 2 CLEAR B
211 2 CONSTRUCT SHIFT
212 2 RESTORE A
213 2 SHIFT BA LEFT LENGTH BITS
214 2 MOVE RESULT FROM B TO A
215 1 END XREXT
FORTRAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

********

CD1 CONVET A SINGLE PRECISION REAL NUMBER TO AN ASCII STRING IN

1PE14.6 FORMAT

********

CD2 INPUT

CD2 REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED

********

CD3 ASCII - SEVEN WORD ASCII CHARACTER STRING REPRESENTATION OF

'REAL'

********

CD4 LOCAL

CD4 R - WORKING LOCATION FOR ABSOLUTE VALUE OR 'REAL' REPEATEDLY

CD4 MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

********

CD5 NOTES

CD5 USES FLOAT, IAND, IFIX, IOR, KCVT, XREXT, XRSFL

********

XRE14 1 BEGIN XRE14

2 SET SIGN FIELD

2 MOVE ABSOLUTE VALUE OF 'REAL' INTERNAL

2 COMPUTE EXPONENT

2 SET SIGN AND VALUE OF EXPONENT FIELD

2 REDUCE VALUE TO RANGE OF 1 <= VALUE < 10

2 EXTRACT FIRST DIGIT AND STORE FIELD

2 SET DECIMAL FIELD

2 DO FOR NEXT THREE PAIRS OF DIGITS

3 MULTIPLY BY 100 TO EXTRACT PAIR

3 EXTRACT DIGITS AND STORE FIELD

END XRE14

END XRE14
FORTRAN CALLING PROCEDURE

CD0       CALL XR16 (INTEGER, ASCII)

CD1       CONVERT A SIXTEEN BIT SIGNED BINARY INTEGER TO A SIX CHARACTER ASCII STRING

CD2       INPUT

CD3       OUTPUT

CD5       THREE WORD CHARACTER STRING REPRESENTATION OF 'INTEGER'

CD6       LOCAL

CD4       I - INTERNAL LOCATION FOR 'INTEGER' REPEATEDLY MODIFIED TO PRODUCE 'ASCII'

CD4       WRK - SEVEN WORD WORKING BUFFER FOR CONSTRUCTION OF 'ASCII'

CD5       NOTES

CD5       USES XMOV AND XRCK

CD6       NOTES

*           *

BEGIN XR16

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
FORTRAN CALLING PROCEDURES

CALL XRLCK (RCODE)
CALL XRULK (RCODE)

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

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

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

INPUT

`VSTB RESOURCE NUMBER`

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

OUTPUT

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

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

LOCAL

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

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

NOTES

`USES .ENTR, RNR`.

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

THIS ROUTINE MAY NOT BE OVERLAYED

************
337  1 BEGIN XRLCK
338  2 SET REQUEST FOR LOCK FUNCTION
339  2 PERFORM RLOCK
340  1 END XRLCK
341  1 BEGIN XRUlk
342  2 SET REQUEST FOR UNLOCK FUNCTION
343  2 PERFORM RLOCK
344  1 END XRUlk
345  1 BEGIN RLOCK
346  2 IF REQUEST IS CONSISTENT WITH STATUS
347     2 THEN
348     2 SET NEW STATUS
349     3 IF RM 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     3 ENDIF
353     3 CLEAR RETURN CODE
354     2 ELSE
355     3 SET RETURN CODE
356     2 ENDIF
357  1 END RLOCK
INTEGER FUNCTION

XRLOC(A)

RETURN THE 16-BIT MAPPED ADDRESS OF A

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

INPUT

OUTPUT

XRLOC - 16-BIT ADDRESS OF A

NOTES

NO EXTERNAL REFERENCES

BEGIN XRLOC

TRANSFER CALLING SEQUENCE

LOAD THE ADDRESS OF THE CALLING PARAMETER

END XRLOC
389 1 *00   FORTRAN CALLING PROCEDURE
390 1 *00
391 1 *00  CALL XRMV (LENGTH, 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 *01
398 1 *02  INPUT
399 1 *02  LENGTH - POSITIVE INTEGER INDICATING NUMBER OF WORDS TO MOVE.
400 1 *02  SOURCE - ARRAY OF WORDS TO BE MOVED
401 1 *02
402 1 *02
403 1 *03  OUTPUT
404 1 *03  OBJECT - ARRAY RECEIVING MOVED WORDS
405 1 *03
406 1 *03
407 1 *04  NOTES
408 1 *04  USES .ENTR
409 1 *05
410 1 *05
411 1 *  
412 1 *  
413 1 *  
414 1 *  
415 1 BEGIN XRMV
416 2 TRANSFER CALLING SEQUENCE
417 2 INITIALIZE MOVE
418 2 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
419 2 END XRMV
FORTRAN CALLING PROCEDURE
CALL XIMSG (NUMBER, LOCATE, LENGTH, SOURCE)

INSERT 'LENGTH' WORDS OF TEXT FROM 'SOURCE' INTO MESSAGE.
'HISTORY' BEGINNING AFTER 'LOCATE' WORDS, CONCATENATE TO PREFIX.
TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL.

INPUT
NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
A - AREA INDICATOR AS FOLLOWS
1 - AS
2 - XD
3 - XI
4 - XH
5 - XS
6 - YT
7 - XH
8 - XH
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 PRECEDE
'SOURCE' (NOT USED IF 'M' 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)
COMMON
LU - USERS LOGICAL UNIT NUMBER

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

NOTES
USES FDS SYSTEM MESSAGE FILE XIMSG
USES CLOSE, EXEC, IAND, KCUT, OPEN, READY, XMOV, XUDOS
BEGIN XMNG
SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
SET NUMBER IN PREFIX
READ MESSAGE DIRECTORY RECORD
IF AREA VALID
THEN
SET AREA CODE IN PREFIX
IF MESSAGE NUMBER > 0
THEN
IF VALID MESSAGE NUMBER
THEN
COMPUTE MESSAGE RECORD NUMBER
READ RECORD
CALL XRMV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
ELSE
EXIT TO :ERROR:
ENDIF
CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
IF MESSAGE NUMBER > 0
THEN
CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
ENDIF
ELSE
SET AREA IN PREFIX
:ERROR: CALL XRMV TO MOVE 'XRMNG ERROR' INTO BUFFER
CALL XRMV 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 XUDNB
ENDIF
END XMNG
INTEGER FUNCTION
XMMFB(Bit, BitNum, String)

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

***
*10 INPUT
*11 *1 BIT - INTEGER VALUE THE LAST BIT OF WHICH IS TO BE COMPARED TO
*12 BITS OF 'STRING' FOR A MATCH
*13 *1 BITNUM - UNINTED SIXTEEN BIT INTEGER INDICATING THE BIT NUMBER IN
*14 'STRING' WITH WHICH TO BEGIN THE SEARCH (FIRST BIT OF
*15 'STRING' IS BIT NUMBER ZERO)
*16 *1 STRING - BIT STRING TO BE SEARCHED. SEARCH WILL CONTINUE THROUGH
MEMORY UNTIL A VALUE OF 'BIT' IS DETECTED

***
*20 OUTPUT
*21 *1 FUNCTION VALUE - BIT NUMBER OF NEXT OCCURRENCE OF 'BIT' >= 'BITNUM'

***
*30 NOTES
*31 *1 USES .ENTR
*32 *1 DS SEARCH WILL NOT TERMINATE UNTIL A VALUE OF 'BIT' IS DETECTED OR
*33 DS ALL OF MEMORY HAS BEEN EXAMINED. THUS, Appropriate STEPS SHOULD
*34 DS BE TAKEN TO FORC A MATCH AT THE END OF THE STRING.
*35 *1 DS A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4096 WORDS) CAN BE
*36 *1 DS MEANINGFULLY ACCOMODATED.
1 BEGIN IRKED
2 COMPUTE STARTING WORD NUMBER
3 COMPUTE STARTING BIT NUMBER
4 LOAD STARTING WORD
5 SHIFT WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
6 INITIALIZE BIT COUNT
7 DO WHILE SIGN BIT IS NOT EQUAL TO BIT
8 IF WORD NOT FINISHED
9 THEN
10 SHIFT WORD LEFT
11 INCREMENT BIT COUNT
12 ELSE
13 DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
14 LOAD NEXT WORD
15 ENDDO
16 CLEAR BIT COUNT
17 ENDF
18 ENDDO
19 RETURN VALUE OF MATCHING BIT NUMBER
20 END IRKED
1 90  FORTRAN CALLING PROCEDURE
2 90  CALL XR06 (BINARY, OCTAL)
3 90  
4 90  ********
5 90  
6 90  ** CONVERT A WORD FROM BINARY TO SIX CHARACTER OCTAL REPRESENTATION **
7 90  
8 90  ********
9 90  ** INPUT ***
10 90  ** BINARY - BINARY WORD TO BE CONVERTED ***
11 90  
12 90  ********
13 90  ** OUTPUT  ***
14 90  ** OCTAL - THREE WORD ARRAY CONTAINING OCTAL REPRESENTATION OF ***
15 90  ** 'BINARY' IN ASCII FORMAT (06) ***
16 90  
17 90  ********
18 90  ** NOTES ***
19 90  ** USES ENTR ***
20 90  
21 90  
22 90  ********
23 90  
24 90  BEGIN XR06
25 90  
26 90  1  BEGIN CALLING SEQUENCE
27 90  2  FORM 16-BIT WORD USING B AND 2 MOST SIGNIFICANT BITS OF A
28 90  3  SET BYTE FLAG HIGH
29 90  4  DO FOR EACH PAIR OF OCTAL DIGITS (3)
30 90  5  SET PREFIX BITS (000000)
31 90  6  SHIFT IN DIGIT
32 90  7  IF BYTE FLAG SET HIGH
33 90  8  THEN
34 90  9  SHIFT FOR ACCOMODATION OF LOW BYTE
35 90  10  ELSE
36 90  11  STORE PAIR OF DIGITS IN OCTAL(1)
37 90  12  CLEAR FOR NEXT PAIR OF DIGITS
38 90  13  ENDIF
39 90  14  FLIP BYTE FLAG
40 90  15  END XR06
41 90  1  END XR06
42 90  
43 90  

FORTRAN CALLING PROCEDURE

CALL XRPACK (LENGTH, UNPKED, PACKED)

+---------------------+
|                      |
| **INPUT**            |
|                      |
| ###1) CONVER** 'LENGTH' CHARACTERS OF 'UNPKED' FROM R1 FORMAT TO A2 PACKED |
|                      |
|                       |
| ***                 |
| +---------------------+
|                      |
| **OUTPUT**           |
|                      |
| +---------------------+
|                      |
| +---------------------+
|                      |
| **NOTES**            |
|                      |
| USES .ENTR            |
|                      |

BEGIN XRPACK

TRANSFER CALLING SEQUENCE

SET BYTE FLAG FOR HIGH BYTE

INITIALIZE PACKED POINTER

DO FOR EACH CHARACTER IN UNPKED

IF BYTE FLAG SET HIGH THEN

LOAD A WITH NEXT WORD OF UNPKED

SHIFT CHARACTER INTO HIGH BYTE

ELSE

INCLUSIVE OR NEXT WORD OF UNPKED INTO A

STORE A IN PACKED

INCREMENT POINTER

ENDIF

FLIP BYTE FLAG

ENDIF

IF BYTE FLAG SET LOW THEN

INCLUSIVE OR BLANK INTO LOW BYTE

ENDIF

STORE A IN PACKED

END XRPACK
657 1 *00 FORTRAN CALLING PROCEDURE
658 1 *00
659 1 *00 CALL XRQFN (PREFIX, NAME4, NAME6)
660 1 *00
661 1 *********
662 1 *01 XRQFN BUILDS A QUALIFIED FILE NAME OF UPTO SIX CHARACTERS IN
663 1 *01 LENGTH BY PREFIXING THE INPUT ONE TO FOUR CHARACTER NAME WITH
664 1 *01 THE PREFIX CHARACTER AND APPENDING A USER QUALIFIER CODE TO THE
665 1 *01 END
666 1 *01
667 1 *01
668 1 *01 *********
669 1 *02 INPUT
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 *02 *********
675 1 *03 OUTPUT
676 1 *03 NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME
677 1 *03
678 1 *03 *********
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 XRQFN
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 XRQFN
FORTRAN CALLING PROCEDURE

CALL XRSET (START, LENGTH, SOURCE, OBJECT)

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

**********

INPUT

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

LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED

SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED

**********

OUTPUT

OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED

**********

NOTES

USES ENTR

**********

BEGIN XRSET

TRANSFER CALLING SEQUENCE

CONSTRUCT SHIFT INSTRUCTIONS

SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER

SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD

CONSTRUCT MASK AND CLEAR FIELD OF OBJECT

INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT

END XRSET
734 1 *DO INTEGER FUNCTIONS
735 1  *DO XRSFL(COUNT, SOURCE)
736 1  *DO XRSFR(COUNT, SOURCE)
737 1  *DO XRSFL(COUNT, SOURCE)
738 1  *DO XRSFR(COUNT, SOURCE)
739 1 ************
740 1 *D1 SHIFT 'SOURCE' LEFT/RIGHT LOGICALLY 'COUNT' BITS
741 1 *D1 ************
742 1 ************
743 1 *D2 INPUT
744 1 *D2 COUNT - POSITIVE INTEGER SPECIFYING NUMBER OF BITS TO SHIFT
745 1 *D2 SOURCE - WORD TO BE SHIFTED
746 1 *D2 ************
747 1 *D5 NOTES
748 1 *D5 USES .ENTR
749 1 *D5 ************
750 1 *
751 1 *
752 1 *
753 1 *
754 1 *
755 1 *
756 1 *
757 1 BEGIN XRSFL
758 2 SET FOR LEFT SHIFT
759 2 PERFORM SHIFT (FLAG, COUNT, SOURCE)
760 1 END XRSFL
761 1 BEGIN XRSFR
762 2 SET FOR RIGHT SHIFT
763 2 PERFORM SHIFT (FLAG, COUNT, SOURCE)
764 1 END XRSFR
765 1 BEGIN SHIFT
766 2 TRANSFER CALLING SEQUENCE
767 2 CONSTRUCT SHIFT INSTRUCTION
768 2 LOAD A WITH SOURCE
769 2 CLEAR B
770 2 SHIFT BA AS SPECIFIED
771 1 END SHIFT
773 1 *DO FORTRAN CALLING PROCEDURE
774 1 *DO CALL XRNG (NAME6, NAME4)
775 1 *DO
776 1 *DO
777 1 *********
778 1 *D1 XRNG REMOVES THE PREFIX AND SUFFIX QUALIFYING CHARACTERS FROM
779 1 *D1 A SIX CHARACTER FILE NAME
780 1 *D1
781 1 *D1
782 1 *********
783 1 *D2 INPUT
784 1 *D2 NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME
785 1 *D2
786 1 *********
787 1 *D3 OUTPUT
788 1 *D3 NAME4 - ONE TO FOUR CHARACTER PACKED NAME WITH PREFIX AND SUFFIX
789 1 *D3 REMOVED
790 1 *D3
791 1 *********
792 1 *D5 NOTES
793 1 *D5 USES .EMTR
794 1 *D5
795 1 *********
796 1 *
797 1 *
798 1 *
799 1 *
800 1 BEGIN XRNG
801 2 MOVE CHARACTERS 2-5 OF NAME6 INTO NAME4
802 2 IF SIXTH CHARACTER IS BLANK, THEN
803 3 LOCATE LAST NON-BLANK CHARACTER OF NAME4
804 3 BLANK THAT CHARACTER
805 2 ENDIF
806 1 END XRNG
FORTRAN CALLING PROCEDURE

808   1 *00         CALL XRUPK (LENGTH, PACKED, UNPKED, COUNT)
809   1 *00         *********
810   1 *00         *********
811   1 *00         *********
812   1 *00         *********
813   1 *00         *********
814   1 *01         CONVERT 'LENGTH' WORDS OF 'PACKED' FROM A2 FORMAT TO R1 FORMAT,
815   1 *01         REMOVING BLANKS NOT DELIMITED BY QUOTE MARKS, AND RETURN IN
816   1 *01         'UNPKED' WITH THE NON-BLANK CHARACTER COUNT IS 'COUNT'.
817   1 *01         *********
818   1 *01         *********
819   1 *02         INPUT
820   1 *02         LENGTH - POSITIVE INTEGER NUMBER OF WORDS IN PACKED
821   1 *02         PACKED - ARRAY OF CHARACTER DATA IN A2 FORMAT
822   1 *02         *********
823   1 *02         *********
824   1 *03         OUTPUT
825   1 *03         UNPKED - ARRAY OF NON-BLANK CHARACTERS IN R1 FORMAT
826   1 *03         COUNT - NUMBER OF CHARACTERS IN UNPKED
827   1 *03         *********
828   1 *05         NOTES
829   1 *05         USES .ENTR
830   1 *05         IF PACKED AND UNPKED ARE THE SAME ADDRESS SPACE UNPKED MAY OVERLAY
831   1 *05         PACKED
832   1 *05         *********
833   1 *05         ** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER
834   1 *05         STRINGS.
835   1 *05         *********
836   1 *05         *********
837   1 *05         *********
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 WOPD
864 3 ENDDO
865 2 ENDDO
866 2 RETURN VALUE OF COUNT
867 1 END XRUPK
FORTRAN CALLING PROCEDURE

CALL XRISP (CHSTR, LENTH)

*********

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

*********

INPUT

CHSTR - CHARACTER STRING OF LENTH WORDS IN A2 FORMAT
LENTH - LENGTH OF CHSTR IN WORDS

*********

OUTPUT

CHSTR - CHARACTER STRING OF LENTH WORDS IN A2 FORMAT WITH
DUPLICATE BLANKS REMOVED
LENTH - NEW LENGTH OF CHSTR IN WORDS

*********

EXTERNAL REFERENCES

.ENTRY

*********

SPECIAL REMARKS

THIS ROUTINE ASSUMES
1. THE INPUT CHARACTER STRING HAS A POSITIVE LENGTH > 0
2. SPECIAL HANDLING OF STRINGS WITHIN QUOTATION MARKS DOES NOT
BEGIN UNTIL THE FIRST OCCURRENCE OF QUOTATION MARKS ON EACH
ENTRY
3. THE FIRST/LAST CHARACTER IN A CHARACTER STRING HAS THE
HIGH ORDER BIT OF THE QUOTE CHARACTER 'ON' TO INDICATE
BEGIN/END OF A CHARACTER STRING.THIS BIT IS TURNED 'OFF'
BY THIS PROGRAM PRIOR TO EXIT.THIS CHANGE MADE TO
INTEGRATE QUOTE MARKS W/I CHARACTER STRINGS.
OTHER PROCESSORS CHANGED FOR THIS MODIFICATION WERE:
A. XILAN,XPRM
B. XILSS/XILS8
## PORTINUS CALLING PROCEDURE

**CALL SELLS(CICERO)**

| INPUT | COMMON XE - LU | COMMOM XE - SUNG | EETC; | NEXTAD; | NUMERLEN;
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>COMMON XE - ESQUE;</td>
<td>COMMON XE - NUMERLEN;</td>
<td>PRIM;</td>
<td>PROMPT;</td>
<td>WEUFE;</td>
</tr>
</tbody>
</table>

**USES Routines**

- EFF1
- EFF2
- EFF3
- EFF4
- EFF5
- EFF6
- EFF7
- EFF8
- EFF9
- EFF0

**NOTES**

- PORTINUS is the main routine of the sequence table editor.

- SELLUS is the procedure for handling selectives and variables.

- SELLS(CICERO) is the main calling procedure for PORTINUS.

- The table above lists the input and output specifications for various routines used by PORTINUS.

- The notes section provides additional information about the usage and limitations of the routines.

---

5-250
BEGIN XSERE

DO UNTIL A % OR 'EXIT' IS ENTERED

CALL XSPEN TO BUILD A PROMPT BASED ON PROMPT MODE FOR THE NEXT TABLE ENTRY (INDICATED BY TAMDIX)

CALL XICOM TO ISSUE THE PROMPT AND RETURN RESPONSE

ERROR: IF ERROR IN XICOM :ERROR:=

EXIT XSERE IF X WAS ENTERED

IF NOTHING (ONLY CR) ENTERED, THEN

IF PROMPT MODE IS NOT 'ALL', THEN

CALL XRMSG - 'INVALID INPUT'

ENDIF

ELSE

CALL XSNPT TO PROCESS INPUT BASED ON PROMPT MODE,

CURRENT TABLE ENTRY (TAMDIX), AND PROMPTED SERENCE NUMBER (PRNUM)

ENDIF

ENDIF

BUILD AWA REQUEST TO DELETE/VERIFY ABSENCE OF NE/TAB

BUILD AWA REQUEST TO PACK THE TABLE BUFFER (REMOVE DELETED ENTRIES)

BUILD AWA REQUEST TO STORE NEWTAB

IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN

CALL XEN to PROCESS THE REQUESTS

IF THE ALLOCATE REQUEST FAILED, THEN

CALL XRMSG - 'ANA/ONA FULL, SERIENCE TABLE NOT STORED'

CALL EXEC TO FREE CLASS NO. AND SAM BUFFER

ELSE

CALL XRMSG - 'VACUOUS TABLE -- NOT STORED'

ENDIF

ENDIF

EXIT XSERE

ERROR: CALL XRMSG - 'SYSTEM ERROR'

END XSERE
156 1 CD0        FORTRAN CALLING PROCEDURE
157 1 CD0
158 1 CD0      CALL XSNPT
159 1 CD0
160 1 CD0
161 1 CD0
162 1 CD0
163 1 CD0      XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE
164 1 CD0      TABLE EDITOR
165 1 CD0
166 1 CD0
167 1 CD0      INPUT
168 1 CD0
169 1 CD0      COMMON XE - COMBUF, COMPTR, LU, TOKENS
170 1 CD0
171 1 CD0
172 1 CD0      COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, P3MTMD
173 1 CD0      SEQNO, TABNOX, WBUF
174 1 CD0
175 1 CD0      OUTPUT
176 1 CD0
177 1 CD0
178 1 CD0      COMMON XE - COMPTR
179 1 CD0
180 1 CD0
181 1 CD0      COMMON XB - INSERT, IRETC, NUMENT, P3MTMD, SEQNO,
182 1 CD0      TABNOX, TABSIZ, WBUF
183 1 CD0
184 1 CD0
185 1 CD0
186 1 CD0      NOTES
187 1 CD0
188 1 CD0      USES ROUTINES
189 1 CD0
190 1 CD0      XRMG
191 1 CD0
192 1 CD0      XSDL
193 1 CD0
194 1 CD0      XSNMT
195 1 CD0
196 1 CD0
197 1 CD0
198 1 CD0
BEGIN XSMPT

.F PROMPT MODE IS UPDATE, THEN

IF TOKEN INPUT IS AN INTEGER, THEN

ERREXIT IF INTEGER < 1 :ERROR1:
RETAIN INTEGER AS SEQUENCE NO. (SERNO)
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT "#" :ERROR1:
INCREMENT TO NEXT TOKEN
START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED
EXITIF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SERNO.)
SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
SET INSERT FLAG TO ZERO INDICATING REPLACEMENT OF ENTRY
EXITIF SEQUENCE NO. OF ENTRY > SEQUENCE NO. INPUT (SERNO)
SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
SET INSERT FLAG TO 1 INDICATING INSERT NEEDED
ORELSE
INCREMENT TO NEXT TABLE ENTRY
ENDLOOP
SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE
SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE
ENDSEACH
CALL XSENT TO REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG

ELSE
ERREXIT IF TOKEN IS NOT A NAME :ERROR1:
START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED
EXITIF NAME INPUT IS DIRECTIVE
SET INDEX TO DIRECTIVE LIST ENTRY
ORELSE
INCREMENT TO NEXT DIRECTIVE
ENDLOOP
ENDORCEACH
CASE (XSLIS, XSDRL, XSPMT, X5NUM, X5MRC), INDEX
ENDIF
ELSE, PROMPT MODE IS NOT UPDATE
IF TOKEN IS #, THEN
SET PROMPT MODE TO UPDATE
F & IS INPUT, THEN
ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR1:
MARK THIS TABLE ENTRY AS DELETED
DO FROM END OF TABLE UNTIL A NONDELETED ENTRY IS FOUND
IF TABLE ENTRY IS MARKED FOR DELETION, THEN
DECREMENT NUMBER OF TABLE ENTRIES BY ONE
ENDIF
ENDDO
ELSE IF PROMPT MODE IS ALL, THEN
SET INSERT FLAG TO TWO TO INDICATE EXTENSION OF TABLE
ELSE
SET INSERT FLAG TO ZERO TO INDICATE REPLACEMENT OF TABLE ENTRY
ENDIF
CALL XSENT TO BUILD ENTRY BASE ON INSERT FLAG
ENDIF
ENDIF
ENDIF
EXIT XSMPT

:ERROR1: CALL XRMESG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
243 2 CD0  FORTAN CALLING PROCEDURE
244 2 CD0
245 2 CD0
246 2 CD0
247 2 CD0
248 2 CD0
249 2 CD0
250 2 CD0
251 2 CD0
252 2 CD0
253 2 CD0
254 2 CD0
255 2 CD0
256 2 CD0
257 2 CD0
258 2 CD0
259 2 CD0
260 2 CD0
261 2 CD0
262 2 CD0
263 2 CD0
264 2 CD0
265 2 CD0
266 2 CD0
267 2 CD0
268 2 CD0
269 2 CD0
270 2 CD0
271 2 CD0
272 2 CD0
273 2 CD0
274 2 CD0
275 2 CD0
276 2 CD0
277 2 CD0
278 2 CD0
279 2 CD0
280 2 CD0
281 2 CD0
282 2 CD0
283 2 CD0
284 2 CD0
285 2 CD0
286 2 CD0
287 2 CD0
288 2 CD0
289 2 CD0
290 2 CD0
291 2 CD0
292 2 CD0
293 2 CD0
294 2 CD0
295 2 CD0
296 2 CD0
297 2 CD0
298 2 CD0
299 2 CD0
300 2 CD0
301 2 CD0
302 2 CD0
303 2 CD0
304 3 CD0  ERREXIT IF TOKEN INPUT IS NOT A NAME :ERRO1:
305 3 CD0  RETAIN THIS NAME AS PROC. NAME
306 3 CD0  INCREMENT TO NEXT TOKEN
307 3 CD0  START SEARCH UNTIL ALL ENTRIES OF XLIBD SEARCHED
308 3 CD0  EXIT IF XLIBD ENTRY = PROC. NAME
309 3 CD0  ORELSE
310 4 CD0  INCREMENT TO NEXT ENTRY
311 3 CD0  ENDOLOOP
312 4 CD0  ERREXIT :ERRO3:
313 3 CD0  ENDSWARCH
314 3 CD0  IF COMMA IS NEXT TOKEN, THEN
315 4 CD0  ERREXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERRO4:
316 4 CD0  INCREMENT TO NEXT TOKEN
317 4 CD0  ERREXIT IF NEXT TOKEN IS NOT A NAME :ERRO1:
318 4 CD0  RETAIN THIS NAME AS INTERFACE TABLE NAME
319 4 CD0  INCREMENT TO NEXT TOKEN
320 3 CD0  ELSE
321 4 CD0  SET INTERFACE TABLE NAME TO ZERO
ENDIF

323  IF NEXT TOKEN IS NOT EOS :ERROR:
324  IF INSERT FLAG DOES NOT INDICATE REPLACE, THEN (I.E. INSERT OR ADD)
325  IF NUMBER OF TABLE ENTRIES (NUMENT) = 150, THEN
326  CALL XSPCK TO PACK TABLE BUFFER (REMOVE DELETED ENTRIES)
327  IF NUMBER OF TABLE ENTRIES STILL = 150, THEN
328  SET PROMPT MODE TO UPDATE
329  ERREXIT :ERROR5:
330  ENDIF
331  EMDIF
332  IF INSERT FLAG INDICATES INSERT (=1), THEN
333  IF ENTRY ABOVE INDEXED ENTRY (TABNX) IS MARKED DELETED, THEN
334  SET TABLE ENTRY INDEX (TABIDX) TO BE THIS DELETED ENTRY
335  SET INSERT FLAG TO 0 INDICATING ENTRY REPLACEMENT
336  ELSE
337  SET MOVLEN = MIN (5, 150-NUMENT) + 7
338  FOR ALL TABLE ENTRIES FROM BOTTOM OF TABLE TO TABIDX
339  MOVE THE ENTRY DOWN MOVLEN WORDS
340  ENDDO
341  IF MOVLEN > 7 (I.E. MORE THAN 1 ENTRY), THEN
342  MARK FOLLOWING ENTRIES AS DELETED
343  ENDIF
344  ENDIF
345  ENDIF
346  SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SEQNO) INPUT/PROMPTED
347  ENDIF
348  MOVE PROC. NAME AND INT. NAME INTO TABLE ENTRY AT TABIDX
349  IF INSERT FLAG DOES NOT INDICATE ENTRY REPLACEMENT, THEN
350  INCREMENT NUMBER OF ENTRIES (NUMENT) BY 1
351  ENDIF
352  2 EXIT XSENT

353  3 :ERROR: CALL XRMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
354  3 :ERROR: CALL XRMSG - '..... IS NOT A VALID PROCESSOR NAME'
355  3 :ERROR: CALL XRMSG - '..... DOES NOT USE AN INTERFACE TABLE'
356  3 :ERROR: CALL XRMSG - 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
357  2 END XSENT
395 2 BEGIN XPCK
396 3 IF THE TABLE IS NOT EMPTY, THEN
397 4 DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
398 5 IF THIS ENTRY IS MARKED DELETED, THEN
399 6 SET MOVE LENGTH (MOVLEN) TO 7
400 7 DO UNTIL A NON-DELETED ENTRY IS FOUND
401 8 INCREMENT MOVLEN BY 7
402 9 ENDDO
403 6 MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
404 6 THE DELETED ENTRY
405 6 DECREMENT NUMENT BY MOVLEN/7
406 6 IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
407 7 DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
408 6 ENDIF
409 5 ENDDO
410 4 ENDDO
411 3 ENDIF
412 2 END XPCK
CD0  FORTRAN CALLING PROEDURE
CD0
CD0  CALL XLST
CD0
CD0
C*********
CD1
CD1  XLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
CD1  CONTAINED IN THE WORKING BUFFER
CD1
C*********
CD2
CD2  INPUT
CD2
CD2  COMMON XE - LU, REBUF, SUBSTA
CD2
CD2
CD2  COMMON XB - BEGNO, DEBUG, ENDNO, LISTLU,
CD2  NEWTAB, NUMENT, WBUF
CD2
CD2
C*********
CD5
CD5  NOTES
CD5
CD5
CD5
CD5  USES ROUTINES
CD5
CD5
CD5
CD5  XRIG
CD5
CD5
CD5  XRMOV
CD5
CD5
CD5  XRSET
CD5
CD5
CD5  XUDBG
CD5
C*********
BEGIN XLST
IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
SET LIMITS (BEGIN AND ENDMO) OF SEQ. ENTRIES LISTED
SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
ENDIF
WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
DO FROM BEGNO TO ENDMO
MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
IF INT. TABLE NAME = 0, THEN
SET LENGTH OF PRINT TO BE 7 WORDS (14 CHAR.
ELSE
SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHAR.
ENDIF
CALL XRIG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
WRITE PRINT BUFFER
ENDIF
ENDIF
END XLST
BEGIN XSCAN
SET LIST LIMITS (BEGINO AND ENDMO) TO ZERO
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
EXIT IF VALUE IS < 1 :ERROR;
SET BEGIN LIMIT (BEGINO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
EXIT IF VALUE IS < 1 :ERROR;
SET END LIMIT (ENDMO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
IF BEGIN LIMIT (BEGINO) = 0, THEN
SET BEGIN LIMIT (BEGINO) TO 1 (BEGINO IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGINO)
SET BEGIN LIMIT (BEGINO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
EXIT :ERROR;
ENDIF
ENDIF
IF END LIMIT (ENDMO) = 0, THEN
SET END LIMIT (ENDMO) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGINO) TO LAST TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDMO)
SET END LIMIT (ENDMO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
EXIT :ERROR;
EXIT SEARCH
ENDIF
SET RETURN CODE TO INDICATE NO ERROR
SET RETURN CODE TO INDICATE AN ERROR
EXIT XSLIC
:ERROR1: CALL XRMIG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERROR6: CALL XRMIG - 'INVALID SEQUENCE NUMBER'
:ERROR2: CALL XRMIG - 'INVALID SEQUENCE NUMBER RANGE'
END XSCAN
SUBROUTINE CALLING PROCEDURE

CALL XPMT

*********

XPMT PROCESSES THE SEQUENCE TABLE EDITOR PRACTICE DIRECTIVE

*********

INPUT

COMMON AE - CONBUF, COMPTR, LU, TOKENS

COMMON XB - DEBUG

*********

OUTPUT

COMMON XB - PRMTND, TABNDX

*********

NOTES

USES ROUTINES

XRMSG

XUDSG

*********

BEGIN XPMT

ERREXIT IF TOKEN IS NOT COMMA :ERR01:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT A NAME :ERR01:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT EOS :ERR01:

IF NAME IS 'M', THEN

SET PROMPT MODE TO CREATE

ELSE

ERREXIT IF NAME IS NOT 'A' :ERR09:

SET PROMPT MODE TO ALL

SET TABLE ENTRY INDEX (TABNDX) TO 0

ENDIF

EXIT XPMT

:ERR01: CALL XRMSG - 'SYNTAX ERROR - MISSING ON EXTRANEOUS FIELD'

:ERR09: CALL XRMSG - 'SYNTAX ERROR - INVALID QUALIFIER'

END XPMT
FORTRAN CALLING PROCEDURE
CALL XSDEL

COMMENTS
INPUT
COMMON XE - LU
COMMON XB - BEGMO, DEBUG, EMDNO, IRETG, NUMENT

OUTPUT
COMMON XB - NUMENT, WKBUF

NOTES

USING ROUTINES

BEGIN XSDEL
CALL XSCAN TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE
IF NO ERROR INDICATED, THEN
DO FROM THE BEGIN LIMIT (BEGMO) TO THE END LIMIT (EMDNO)
MARK THIS SEQ. TABLE ENTRY AS DELETED
ENDDO
DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR
UNTIL A NON-DELETED ENTRY FOUND
IF ENTRY IS MARKED DELETED, THEN
DECREMENT NUMBER OF TABLE ENTRIES (NUMENT) BY ONE
ENDIF
ENDDO
ENDIF
END XSDEL
716  2  CO5
717  2  C****************
718  3  BEGIN XSMUM
719  4  ERREXIT IF TOKEN IS NOT EOS :ERROR1:
720  5  IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
721  6  SET SEQUENCE NUMBER (SEQNO) TO 100
722  7  DO FOR ALL ENTRIES IN TABLE
723  8  IF TABLE ENTRY IS NOT MARKED DELETED, THEN
724  9  SET SEG. NO. FIELD OF ENTRY TO SEQUENCE NUMBER (SEQNO)
725 10  INCREMENT SEQUENCE NUMBER (SEQNO) BY 100
726 11  ENDIF
727 12  ENDDO
728 13  ENDIF
729 14  EXIT XSMUM
730  3  :ERROR1: CALL XMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
731  2  END XSMUM
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:

CALL XICOM (PROMPT, PRMLN, RETCOD)

WRITE "PRMLN" WORDS OF CHARACTERS 'PROMPT' TO THE USER.
READ THE USER'S RESPONSE, CALL XICOM TO ANALYZE THE RESPONSE,
AND RETURN A 'RETCOD' TO THE USER INDICATING THE SUCCESS OF
XICOM AND THE USEFULNESS OF THE RESPONSE.

INPUTS FROM CALLING SEQUENCE:

PROMPT - (INTEGER, PRMLN WORDS) AN ARRAY OF PRMLN WORDS
USED AS A PROMPT TO THE USER.
PRMLN - (INTEGER, 1 WORD) THE LENGTH IN WORDS OF
THE PROMPT ARRAY. MAXIMUM LENGTH IS 38 WORDS.

OUTPUTS IN CALLING SEQUENCE:

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

INTERNAL VARIABLES

PRMS - 6 WORD ARRAY CONTAINING THE CONTINUATION MESSAGE
PREFIX - 4 CHARACTERS USED AS PREFIXES TO PROMPT,
CORRESPONDING TO CODES IN XE(5)
RETCOD - RETURN CODE FROM XICOM EXTENDED PROMPTING
RETCODA - RETURN CODE FROM XICOM EXTENDED PROMPTING
SUFFIX - 1_ CHARACTERS APPENDED TO END OF PROMPT

COMMON:

XICOM COMMON:

EQUIVALENCE (XE(1), LU), (XE(2), ILASS),
+ (XE(5), LFL), (XE(145), LOMRIP),
+ (XE(145), MOTOK), (XE(146), MOWDS)

XS COMMON:


BEGIN XTCOM

IF PRIORITY 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
ENDIF

:LOOP:
  ISSUE WRITE TO PROMPT USER
  INITIALIZE COMMUNICATIONS BUFFER
  TURN SYMBOLIC STRING FLAG OFF
  INITIALIZE LA RETURN CODE TO NORMAL RETURN
  PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
  DO WHILE LEXICAL ANALYSIS (LA) RETURN CODE SAYS CONTINUE AND
    (EXEC LEVEL IS NOT INTERFACE TABLE EDITOR OR
    SYMBOLIC STRING FLAG IS ON)
    CALL EXEC TO WRITE CONTINUATION MESSAGE
  PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
ENDDO

IF LA RETURN CODE SAYS ERROR IN RESPONSE THEN
  CALL KCVT TO CONVERT OCTAL TO ASCII
  CALL XRMSP 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 XRMSP 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 XRMSP TO WRITE ERROR MESSAGE
  ENDIF
  GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF

IF LA RETURN CODE SAYS COMBUFF IS FULL THEN
  CALL XRMSP 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
131 1 BEGIN READSEG
132 2 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
133 3 CALL EXEC TO READ RESPONSE TO PROMPT
134 4 CALL XRUPK ROUTINE TO CONVERT A2 FORMAT BUFFER TO R1
135 5 IF NUMBER OF WORDS READ IS NOT ZERO THEN
136 6 CALL XTLAM ROUTINE TO BUILD COMMUNICATIONS BUFFER
137 7 ELSE
138 8 IF LAST LA RETURN CODE WAS A CONTINUE THEN
139 9 REMOVE TRAILING COMMAS FROM COMBUF
140 4 SET LA RETURN CODE TO NORMAL RETURN
141 3 ELSE
142 4 SET LA RETURN CODE TO SAY USER ENTERED CR
143 3 ENDIF
144 2 ENDIF
145 1 END READSEG
FORTRAN CALLING PROCEDURE FOR LEXICAL ANALYSIS:

CALL XTLAN (RETC)

CONVERT 'INBUF' USER'S RESPONSE TO 'COMBUF' OF TOKENS
INDICATING CHARACTERS, INTEGERS, REALS, ETC.

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 1. 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
DINT - DOUBLE PRECISION LOCATION TO ACCUMULATE AN
INTEGER VALUE
DBLWD - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER
AND FRACTIONAL VALUE FOR DOUBLE PRECISION OR REM.
FLGCOM - COMMA FLAG
FLGCHG - LAST CHARACTER WAS NOT A COMMA
FLGCMA - LAST CHARACTER WAS A COMMA
FLGEN - END LOOP FLAG
FLGNO - CONTINUE LOOP
FLGNEC - NEGATIVE EXPONENT FLAG
FLGGNP - EXPONENT WAS POSITIVE
FLGGNE - EXPONENT WAS NEGATIVE
FLGTP - TYPE OF REAL VALUE
FLGSP - SINGLE PRECISION
FPLD - DOUBLE PRECISION
FODER - EXPONENT PART OF A REAL NUMBER
RELU - SINGLE PRECISION LOCATION FOR REAL VALUE
SPCHAR - 256 SPECIAL CHARACTER ARRAY CONTAINING
THE 80 TRUMPET REPRESENTATION FOR:
"4>*/O#$&?'(1)!XX.YZ. : DEW
X IS A CLOSED BRACKET
Y IS AN OPEN BRACKET
Z IS A BACK SLASH
206 1 CD4
207 1 CD5
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 CD5

XE COMMON USED:

EQUIVALENCE (XE(85), TOKENS), (XE(145), COMBUF),
+ (XE(145), NOTOK), (XE(145), NOWD3),

XS COMMON USED

EQUIVALENCE (XS(1), INBUF), (XS(S1), NOCHAIR),
+ (XS(122), FLAG5), (XS(186), SCRATCH)

SUBROUTINES AND FUNCTIONS CALLED

XRPKC, XRMOV

PDL ROUTINES INCLUDED:

XLAM, COMMA, ALPHA, DIGIT, DCOL, DECP, EORD
INTEG, REAL, DBL, REPET, INVAL, SCHARS,QUOTE
BEGIN XILAN
SET END FLAG OFF
IF 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 49 CONTINUATION REQUESTED
ENDIF
SET END FLAG ON
ELSE
ERRINIT IF COMBUF IS FULL PERFORM COMFUL
ENDIF
IF INPUT CHARACTER IS A COMMA THEN
PERFORM COMPA
ELSE
SET LAST CHARACTER WAS A COMMA OFF
IF INPUT CHARACTER IS AN ALPHABETIC CHARACTER THEN
PERFORM ALPHA
ELSE
IF INPUT CHARACTER IS A DIGIT THEN
PERFORM DIGIT
ELSE
PERFORM SCHARS
ENDIF
ENDIF
PERFORM COMPA
STORE END OF BUFFER TOKEN IN COMBUF
INCREMENT #TOKEMS BY 1
END XILAN
1
* *
BEGIN COMMA
SET LAST CHARACTER WAS A COMMA ON
ERRINIT 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
ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
CALL XRMOV TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
  INPUT CHARACTER IS A NUMERIC OR
  INPUT CHARACTER IS AN EXCLAMATION POINT) AND
  INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
  MOVE CHARACTER INTO TEMPORARY BUFFER
GET NEXT INPUT CHARACTER
ENDDO
SET CHARACTER COUNT = 6
STORE CHARACTER NAME TOKEN IN COMBUF
CALL XRPACK ROUTINE TO PACK CHARACTERS INTO COMBUF
INCREMENT #WORDS IN COMBUF BY 4
INCREMENT #TOKENS BY 1
END ALPHA
BEGIN DIGIT
INITIALIZE POWER TO ZERO
PERFORM DECPL
IF INPUT BUFFER IS NOT EXHAUSTED THEN
  IF INPUT CHARACTER IS A "9" THEN
    PERFORM DECPT
  ELSE
    IF INPUT CHARACTER IS AN "E" OR A "D" THEN
      PERFORM EOMD
    ELSE
      IF INPUT CHARACTER IS AN "Z" THEN
        PERFORM REPET
      ELSE
        PERFORM INTEGR
      ENDIF
    ENDIF
  ELSE
    PERFORM INTEGR
  ENDIF
ELSE
  PERFORM INTEGR
ENDIF
END DIGIT
1  BEGIN DCOL
  2      SET INTEGER = 0
  3      SET COUNTER = 0
  4      DO WHILE CHARACTER IS A DIGIT AND
  5          INPUT BUFFER IS NOT EXHAUSTED
  6          SET INTEGER = (INTEGER + 10) * CURRENT CHARACTER - 48
  7          EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
  8          INCREMENT COUNTER BY 1
  9          GET NEXT CHARACTER
 10      END DO
11  1 END DCOL
12
13  1 BEGIN DECP
14    1 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
15    1 GET NEXT INPUT CHARACTER
16    2 IF INPUT BUFFER IS NOT EXHAUSTED THEN
17      3 IF INPUT CHARACTER IS A DIGIT THEN
18      4      PERFORM DCOL
19      5      ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
20      6      EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
21    7      ENDIF
22    8      IF INPUT CHARACTER IS AN "E" OR A "D" THEN
23      9      PERFORM EORD
24      10 ELSE
25      11      PERFORM REAL
26      12      ENDIF
27      13 ELSE
28      14      PERFORM REAL
29      15      ENDIF
30      16 ELSE
31      17      PERFORM REAL
32      18      ENDIF
33      19 END DECP

5-278
<table>
<thead>
<tr>
<th>Line</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>379</td>
<td>BEGIN INTEGER</td>
</tr>
<tr>
<td>380</td>
<td>ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL</td>
</tr>
<tr>
<td>381</td>
<td>CONVERT NUMBER TO INTEGER</td>
</tr>
<tr>
<td>382</td>
<td>ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>383</td>
<td>STORE INTEGER TOKEN IN COMBUF</td>
</tr>
<tr>
<td>384</td>
<td>INCORRECT WORDS IN COMBUF BY 2</td>
</tr>
<tr>
<td>385</td>
<td>INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>386</td>
<td>END INTEGER</td>
</tr>
<tr>
<td>387</td>
<td>1</td>
</tr>
<tr>
<td>388</td>
<td>1</td>
</tr>
<tr>
<td>389</td>
<td>BEGIN REAL</td>
</tr>
<tr>
<td>390</td>
<td>ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL</td>
</tr>
<tr>
<td>391</td>
<td>SET REAL = DOUBLE PRECISION * 10 ** POWER</td>
</tr>
<tr>
<td>392</td>
<td>ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>393</td>
<td>STORE REAL TOKEN IN COMBUF</td>
</tr>
<tr>
<td>394</td>
<td>INCORRECT WORDS IN COMBUF BY 3</td>
</tr>
<tr>
<td>395</td>
<td>INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>396</td>
<td>END REAL</td>
</tr>
<tr>
<td>397</td>
<td>1</td>
</tr>
<tr>
<td>398</td>
<td>1</td>
</tr>
<tr>
<td>399</td>
<td>BEGIN DBL</td>
</tr>
<tr>
<td>400</td>
<td>ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL</td>
</tr>
<tr>
<td>401</td>
<td>SET DOUBLE = DOUBLE PRECISION * 10 ** POWER</td>
</tr>
<tr>
<td>402</td>
<td>ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW</td>
</tr>
<tr>
<td>403</td>
<td>STORE DOUBLE TOKEN IN COMBUF</td>
</tr>
<tr>
<td>404</td>
<td>INCORRECT WORDS IN COMBUF BY 4</td>
</tr>
<tr>
<td>405</td>
<td>INCREMENT TOKENS BY 1</td>
</tr>
<tr>
<td>406</td>
<td>END DBL</td>
</tr>
<tr>
<td>407</td>
<td>1</td>
</tr>
<tr>
<td>408</td>
<td>BEGIN REPET</td>
</tr>
<tr>
<td>409</td>
<td>ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL</td>
</tr>
<tr>
<td>410</td>
<td>ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL</td>
</tr>
<tr>
<td>411</td>
<td>END REPET</td>
</tr>
<tr>
<td>412</td>
<td>BEGIN</td>
</tr>
<tr>
<td>413</td>
<td>ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL</td>
</tr>
<tr>
<td>414</td>
<td>END</td>
</tr>
<tr>
<td>415</td>
<td>BEGIN</td>
</tr>
<tr>
<td>416</td>
<td>ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL</td>
</tr>
<tr>
<td>417</td>
<td>END</td>
</tr>
<tr>
<td>418</td>
<td>GET NEXT CHARACTER</td>
</tr>
<tr>
<td>419</td>
<td>END REPET</td>
</tr>
</tbody>
</table>
1 BEGIN SCNARS
2  * CHARACTER TABLE:
3  "-"/"0-9"="?"/"28-"X:NY;
4  * NH ARE INVALID CHARACTERS HERE
5  X IS A CLOSED BRACKET
6  Y IS AN OPEN BRACKET
7  Z IS A BACK SLASH
8
9 SET J=1
10 STARTSEARCH WHILE J<>#CHARACTERS IN TABLE
11 EXITS IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
12 INCREMENT J BY 1
13 ENDCASE
14 PERFORM INVAL - NO RETURN
15 ENDCASE
16 SET NEXT FLAG ON
17 CASE J: A: F; F: F; F: F; F; F; F; B;
18  ( ) : = 1 2 3 4 5 6 7 8 9 0
19  #: F; F: F; F: F; F: F; F: F; F; INVAL; INVAL; F;
20  :B:
21  :A:
22  :D:
23  SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED
24  PERFORM QUOTE
25
26 SET END FLAG ON
27
28 IF SYMBOLIC STRING FLAG IS OFF THEN
29  SET SYMBOLIC STRING FLAG TO CURRENT COMBUF INDEX + 1
30 ELSE
31  SET COMBUF(SYMBOLIC STRING FLAG) = CURRENT COMBUF INDEX - SYMBOLIC STRING FLAG
32  SET J = J+1 TO STORE SYMBOLIC STRING CLOSE TOKEN
33  SET SYMBOLIC STRING FLAG OFF
34  ENDCASE
35
36 SET RETURN CODE TO SAY I ENTERED
37 SET END FLAG ON
38
39 IF FOLLOWING CHARACTER IS A DIGIT THEN
40  SET NEXT FLAG OFF
41  SET INTEGER = 0
42  SET POWER = 0
43  PERFORM DECP
44  ENDCASE
45
46 IF NEXT FLAG IS ON
47  STORE TOKEN (J) IN COMBUF
48  INCREMENT WORDS IN COMBUF BY 1
49  IF TOKEN IS BEGIN SYMBOLIC STRING THEN
50  INCREMENT WORDS IN COMBUF BY 1
51  ENDCASE
52  ENDCASE
53  GET NEXT CHARACTER
BEGIN QUOTE
GET NEXT CHARACTER
SET #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
MOVE CHARACTER INTO TEMORARY BUFFER
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
ENDDO
ENDDO
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 XRCPCK 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 INVALID
504 2 SET RETURN CODE TO SAY INVALID RESPONSE
505 2 SET END FLAG ON
506 1 EXIT XLAN
507 1 END INVALID
508 1 *
509 1 *
510 1 *
511 1 BEGIN COMBUF
512 2 SET RETURN CODE TO SAY COMBUF IS FULL
513 2 SET END FLAG ON
514 1 EXIT XLAN
515 1 END COMBUF
516 1 *
517 1 *
518 1 *
519 1 BEGIN OVFLOW
520 2 SET RETURN CODE TO SAY OVERFLOW/UNDERFLOW
521 2 SET END FLAG ON
522 1 EXIT XLAN
523 1 END OVFLOW
**FORTRAN CALLING PROCEDURE**

**CALL XTPRM**

**XTPRM IS CALLED BY XTCOM TO PROVIDE EXTENDED PROMPTING WHEN A**

? IS ENTERED FROM THE USER TERMINAL.

**INPUT**

**EXTERNAL - CARTR, COMBUF, FLAGS, LU, NOPROC, PRKAM, SUBSTA,**

**TOKENS, **

**XB COMMON - ARGPTR, WKBUF (FROM INTERFACE TABLE EDITOR) **

**XS COMMON - PRDLEN (FROM XTCOM) **

**VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)**

**OUTPUT**

**EXTERNAL - COMBUF (USED FOR SCRATCH SPACE)**

**LISTING OF APPROPRIATE EXTENDED PROMPTS**

**INTERNAL VARIABLES**

**CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY**

**FILE - FILE NAME OF CURRENT TABLE ENTRY**

**INDEX - INDEX TO CURRENT TABLE ENTRY**

**L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO**

**FIRST LIST ITEM IN RECORD 1 (SEE SDD SECTION 6.2.4.3)**

**MESSAGE - MESSAGE NUMBER, IF ANY, FOR CURRENT TABLE ENTRY**

**N - NUMBER OF LIST ITEMS IN RECORDS 1 AND 2 (SEE SDD **

**SECTION 6.2.4.3)**

**RECORD - RECORD NUMBER WITH WHICH TO BEGIN PROCESSING FOR CURRENT**

**TABLE ENTRY**

**SEARCH - LIST SEARCH FLAG (1) FOR CURRENT TABLE ENTRY**

**SIZE - LIST ITEM SIZE OF CURRENT TABLE ENTRY (NEGATIVE INDICATES**

**LAST CHARACTER TO BE MASKED)**

**TABLE - PROCESSING CONTROL TABLE FOR VARIOUS SYNTAX CONDITIONS**

**INDEX SYNTAX CONDITION**

**FILE REC SIZE SRC MESSAGE CONTINUE**

**1 I X:? 1 I >>XPRM 1 2 NO NONE NO I**

**2 I X?:R 1 9 I >>XPRM 1 2 YES XT06 NO I**

**3 I #:? 1 17 I >>XPRM 1 2 NO NONE YES I**

**4 I #:, $P:? $P:? 1 25 I XLIBD 2 -3 NO NONE NO I**

**5 I #:R 1 33 I >>XPRM 1 2 YES NONE YES I**

**6 I #:R, $P R $P R 1 41 I >>XPRM 3 128 NO XT07 NO I**

**7 I /:? 1 49 I >>XPRM 1 2 NO NONE NO I**

**8 I /:R 1 57 I >>XPRM 1 3 NO NONE NO I**

**9 I /:R 1 65 I >>XPRM 1 2 YES NONE YES I**

**10 I /:P, $P:P 1 73 I >>XPRM 1 3 YES XT08 NO I**

**WHERE P AND R INDICATE A PROMPT AND RESPONSE**

**NOTES**
1 BEGIN XTPRM
2 PERFORM SETUP TO COMPLETE CONTROL TABLE AND INDEX TO APPROPRIATE ENTRY
3 DO UNTIL 'NO CONTINUE' FOUND (0 IN ENTRY CONTINUE FIELD)
4 IF OPEN SUCCESSFUL, THEN
5 UNLOAD 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 READ RECORD AND APPEND TO FIRST RECORD DATA
10 EXIT TO :ERROR9: IF FAILURE
11 ENDIF
12 ELSE
13 SET SIZE POSITIVE
14 DO FOR EACH LIST ITEM (1-N)
15 BLANK LAST CHARACTER
16 ENDDO
17 ENDIF
18 IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), THEN
19 START SEARCH WHILE LIST ITEMS REMAIN TO BE EXAMINED
20 EXIT IF ITEM LOCATED IN LIST
21 POSITION TO APPROPRIATE RECORD (E+L-1) AND READ
22 EXIT TO :ERROR9: IF FAILURE
23 SET SIZE TO 128 (ENTIRE RECORD TO BE DISPLAYED)
24 END LOOP
25 IF TABLE MESSAGE NUMBER FIELD > 0, THEN
26 CALL XRMG TO DISPLAY 'NOT VALID ...' MESSAGE
27 ENDIF
28 EXIT TO ENDDO
29 END SEARCH
30 ENDIF
31 PERFORM DISPLAY
32 ELSE OPEN ERROR
33 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
34 CALL XRMG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
35 ELSE
36 :ERROR9: CALL XRMG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
37 CLOSE FILE
38 ENDIF
39 ENDDO
40 END XTPRM
637 1 BEGIN SETUP
638 1 EXIT TO :ERROR2: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
639 2 CASE (:X:, :S:, :F:, :/) SUBST
640 3 :XI
641 3 IF FIRST TOKEN = ?, THEN
642 4 SET TABLE INDEX TO FIRST ENTRY
643 4 ELSE
644 4 SET TABLE INDEX TO SECOND ENTRY
645 4 ENDIF
646 3 :X2
647 3 IF FIRST TOKEN = ?, THEN
648 4 SET TABLE INDEX TO FOURTH ENTRY
649 4 ELSE
650 4 SET TABLE INDEX TO SIXTH ENTRY
651 4 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
652 3 ENDIF
653 3 :F:
654 3 IF PROMPT LENGTH = 0, THEN
655 4 ASSUME TABLE INDEX OF THIRD ENTRY
656 3 ELSE
657 4 ASSUME TABLE INDEX OF FOURTH ENTRY
658 3 ENDIF
659 3 IF FIRST TOKEN IS A NAME, THEN
660 4 INCREMENT TABLE INDEX BY TWO ENTRIES (NOW FIFTH OR SIXTH)
661 4 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
662 3 ENDIF
663 3 :F2
664 3 FORM TENTH ENTRY FILE NAME FROM > AND IT EDITOR CURRENT PROCESSOR NAME
665 3 ASSUME TABLE INDEX OF TENTH ENTRY
666 3 IF FIRST TOKEN = ?, THEN
667 4 IF PROMPT LENGTH = 0, THEN
668 5 CHANGE TABLE INDEX TO SEVENTH ENTRY
669 5 COPY FILE NAME FROM ENTRY TEN TO ENTRY EIGHT
670 4 ELSE
671 5 CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY IT EDITOR
672 4 ENDIF
673 3 ELSE
674 4 IF PROMPT LENGTH = 0, THEN
675 4 CHANGE TABLE INDEX TO NINTH ENTRY
676 4 ENDIF
677 3 ENDIF
678 2 END CASE
679 1 EXIT SETUP
680 2 :ERROR2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
681 1 END SETUP
BEGIN DISPLAY 128
THEN ITEMS DISPLAYED
DO UNTIL ALL LIST ITEMS DISPLAYED
BLANK LINE
DISPLAY LINE
ENDDO
ELSE DISPLAY EXTENDED PROMPT
ENDIF
END DISPLAY

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>#: 653</td>
</tr>
<tr>
<td>IA: 640</td>
</tr>
<tr>
<td>ALPHA: 274</td>
</tr>
<tr>
<td>ID: 444</td>
</tr>
<tr>
<td>IC: 511</td>
</tr>
<tr>
<td>COMMA: 265</td>
</tr>
<tr>
<td>D: 454</td>
</tr>
<tr>
<td>DBL: 401</td>
</tr>
<tr>
<td>DCOL: 316</td>
</tr>
<tr>
<td>DECP: 330</td>
</tr>
<tr>
<td>DIGIT: 293</td>
</tr>
<tr>
<td>DISPLAY: 483</td>
</tr>
<tr>
<td>E: 457</td>
</tr>
<tr>
<td>EORD: 349</td>
</tr>
<tr>
<td>ERROR: 480</td>
</tr>
<tr>
<td>ERROR2: 630</td>
</tr>
<tr>
<td>F: 464</td>
</tr>
<tr>
<td>INTGR: 379</td>
</tr>
<tr>
<td>INVAL: 503</td>
</tr>
<tr>
<td>LOOP: 92</td>
</tr>
<tr>
<td>OVFLOW: 519</td>
</tr>
<tr>
<td>QUOTE: 477</td>
</tr>
<tr>
<td>READS: 131</td>
</tr>
<tr>
<td>REAL: 390</td>
</tr>
<tr>
<td>REPET: 412</td>
</tr>
<tr>
<td>SCHR: 421</td>
</tr>
<tr>
<td>SETUP: 637</td>
</tr>
<tr>
<td>XCOM: 87</td>
</tr>
<tr>
<td>XTLAB: 230</td>
</tr>
<tr>
<td>XTPRM: 392</td>
</tr>
<tr>
<td>&amp;: 646</td>
</tr>
<tr>
<td>X: 640</td>
</tr>
<tr>
<td>/= 663</td>
</tr>
</tbody>
</table>
FORTRAN CALLING PROCEDURE

CALL XUSBG (I,U, ID)

**XUSBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY**

DUMP AND/OR MODIFICATION FACILITY

**INPUT (CALLING SEQUENCE)**

**LU** - LOGICAL UNIT TO INTERACT WITH IN INVOKING XUSBG OPTIONS

**ID** - THREE WORD ASCII ARRAY USED AS A HEADER TO IDENTIFY XUSBG CALLER

**OPTION (S,M,E):** ONE ASCII CHARACTER IDENTIFYING XUSBG OPTION

S = SNAPSHOT (DUMP) MEMORY

M = MODIFY MEMORY

E = EXIT XUSBG

**START:** OCTAL MEMORY ADDRESS IN THE USERS 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** "*** XUSBG FROM IDIDO" 

**PROMPTS** (SEE INPUT)

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

**HEADER** "*** XUSBG FROM IDIDO"

**SNAP** Y02 WORD DUMP FORMATTED LINE (SEE XUOPL)

**BASE** NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE

**CLASS** CLASS I/O NUMBER FOR TERMINAL INPUT

**LENGTH** LENGTH OF CHARACTER STRING BEING MANIPULATED

**LINE** EIGHT CHARACTER BUFFER OF WORD TO BE SNAPPED

**LUI** TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET

**LUO** LOGICAL UNIT FOR SNAP OUTPUT

**OPN** ONE CHARACTER EXECUTION OPTION CODE

**ORIGIN** REFERENCE POINT FOR MEMORY ACCESS OFFSET COMPUTATION

**ORG** ADDRESS OF ORIGIN

**STP** "ORIGIN" RELATIVE END OF DATA TO BE SNAPPED

**STR** "ORIGIN" RELATIVE BEGINNING OF MEMORY TO BE ACCESSED

**RTN** RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND CONVERSION'

**NOTES**
BEGIN XU86
WRITE SNAP HEADER TO TERMINAL
DO UNTIL OPTION IS EXIT (E)
IF OPTION IS NOT EXIT (E)
THEN
PERFORM PROMPT AND CONVERSION FOR STARTING ADDRESS
IF OPTION IS MODIFY (%)
THEN
PERFORM PROMPT AND CONVERSION FOR VALUES AND STORE IN MEMORY
ELSE
PERFORM PROMPT AND CONVERSION FOR ENDING ADDRESS
PERFORM PROMPT AND CONVERSION FOR OUTPUT UNIT
WRITE SNAP HEADER TO OUTPUT UNIT
DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
CALL XU86 TO FORMAT LINE
OUTPUT LINE
END DO
ENDIF
END IF
END DO
BEGIN XU86
BEGIN PROMPT AND CONVERSION
ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
RETRIEVE RESPONSE
CLEAR SUM
DO FOR EACH CHARACTER
SHIFT SUM AND ADD NEXT DIGIT
END DO
END PROMPT AND CONVERSION
GENERAL FILE DUMP PROGRAM FOR FILE MANAGER FILES

NAME - NAME OF FN FILE TO BE DUMPED
IREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED
RECORD IS RECORD NUMBER ONE
MREC - NUMBER OF LOGICAL RECORDS TO DUMP
FMT - RUN TIME FORMAT FOR RECORDS (MAXIMUM OF 72 CHARACTERS) OR BLANK INDICATING THE DEFAULT OF OCTAL AND ASCII DUMP TYPE FOR FORMATTING OR THE CHARACTERS OF INDICATING UNFORMATTED OUTPUT
LU - LOGICAL UNIT NUMBER OF OUTPUT DEVICE

FORMATTED DUMP OF THE INDICATED PORTION OF THE INDICATED FILE

NOTES

USES EXEC, MAXQ, OPEN, POINT, READF, XMPAR, XPRBS, XNV0V, XUXPL

ANY FILE WITH VARIABLE LENGTH RECORDS WILL BE DUMPED USING A RECORD BUFFER OF 1024 WORDS, THUS LIMITING THE MAXIMUM DUMPABLE LENGTH.

BEGIN XUXPF

DO FOREVER

READ FILE NAME
EXIT XUXPF IF NAME IS NULL
READ INITIAL RECORD NUMBER
READ NUMBER OF RECORDS TO DUMP
READ DUMP FORMAT IF FORMAT IS NULL THEN
SET DEFAULT OCTAL/ASCII FORMAT
END IF
READ LU OF PRINT DEVICE
OPEN FILE IF SUCCESSFUL THEN
DO FOR NUMBER OF RECORDS TO DUMP
READ RECORD EXIT TO :ERROR: IF FAILED
FORMAT AND PRINT RECORD ENDDO ELSE
:ERROR: OUTPUT MESSAGE ENDIF ENDDO
FORTRAN CALLING PROCEDURE

1 CALL XUDPL (ADDRESS, LINE, BUFFER)

1 PRODUCE AN OCTAL AND ASCII PRINT FORMATTED MEMORY DUMP LINE IMAGE

1 INPUT

1 ADDRESS - TWO WORD INTEGER ARRAY CONTAINING THE ABSOLUTE AND RELATIVE ADDRESS TO BE FORMATTED WITH THE LINE

1 LINE - EIGHT WORD ARRAY TO BE CONVERTED TO OCTAL AND ASCII FORMATTED AND SPACED INTO A LINE IMAGE

1 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.

1 OUTPUT

1 BUFFER - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE

1 COLUMNS CONTENTS

1 3-8 FIRST ADDRESS

1 11-16 SECOND ADDRESS

1 21-42 OCTAL REPRESENTATION OF 'LINE'

1 87-102 ASCII REPRESENTATION OF 'LINE'

1 NOTES

1 USES XREX, XR06, XRSET

1 BEG

1 BEGIN XUDPL

1 CALL XR06 TO CONVERT EACH WORD OF ADDRESS TO OCTAL

1 DO FOR EACH WORD OF LINE

1 CALL XR06 TO CONVERT WORD TO OCTAL

1 DO FOR EACH BYTE OF WORD

1 IF BYTE < 40 OR BYTE > 1368

1 THEN

1 REPLACE BYTE WITH ASCII PERIOD

1 ENDIF

1 END DB

1 END XUDPL
CD1 FORTRAN MAIN PROGRAM XUFNT IS SCHEDULED BY XUDMP TO PRINT
CD2 A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD3
CD5 INPUT
CD6 ICNTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
CD7 LE NUMBER OF WHERE DUMP IS ON DISK
CD8 TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
CD9 CONTAINING THE DUMP
CD10 OUTPUT
CD11 FORMATTED DUMP TO LU 6
CD12 EXTERNAL REFERENCES
CD13 EXEC
CD14 RMPAR
CD15 XRCPR
CD16 ZRMV
CD17 XUDPL
CD18
CD19 BEGIN XUFNT
CD20 CALL RMPAR TO GET LU AND STARTING TRACK M0S.
CD21 READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
CD22 COUNT OF ID-SEG TO BE DUMPED
CD23 .UP TO 7 ID-SEGMENT ADDRESSES
CD24 LOW AND HIGH BASE PAGE ADDRESSES
CD25 LOW AND HIGH MAIN MEMORY ADDRESSES
CD26 POINT TO 2ND SECTOR OF DUMP DATA
CD27 DO UNTIL ALL ID-SEG PRINTED
CD28 DO UNTIL A EIGHT-W0RD LINES PRINTED
CD29 PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
CD30 END0
CD31 INCREMENT TO NEXT SECTOR OF DUMP DATA
CD32 ENDD0
CD33 COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
CD34 PERFORM COMPARE AND PRINT FUNCTION
CD35 COMPUTE TRK, NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ
CD36 DO UNTIL TRK ARE READ
CD37 READ NEXT TRACK FROM DISK
CD38 COMPUTE W, THE NUMBER OF 8-WORD LINES TO DUMP
CD39 PERFORM COMPARE AND PRINT FUNCTION
CD40 ENDD0
CD41 RELEASE THE GLOBALLY ALLOCATED TRACKS
CD42 EXIT XUFNT
CD43 BEGIN COMPARE AND PRINT FUNCTION
CD44 DO UNTIL N LINES PROCESSED
CD45 IF NOT 1ST LINE, THEN
CD46 CALL XRCPR TO COMPARE WITH PREVIOUS LINE
CD47 IF LINES ARE IDENTICAL, THEN
CD48 IF THIS IS 1ST OF A SERIES, THEN
CD49 WRITE 'DUPLICATE LINE'
CD50 ENDF0
CD51 ELSE
CD52 CALL XUDPL TO FORMAT THE DUMP LINE
CD53 WRITE FORMATTED DUMP LINE
CD54 ENDF0
CD55 ELSE
CD56 CALL XUDPL TO FORMAT THE DUMP LINE
CD57 WRITE FORMATTED DUMP LINE
CD58 ENDF0
CD59 END0
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
*1* ENTRY XVABN

*2* TYPE 7A ROUTINE TO ABORT CURRENT ID AND

*3* REQUEST A DUMP FOR THE ID.

*4* ENTRY XVABN

*5* TYPE 7A ROUTINE TO DUMP FROM CURRENT ID

*6* THE REQUESTED DATA AREAS AND THE CURRENT

*7* ID SEGMENT AND ITS FATHER CHAIN SEGMENTS

*8* ABEND

*9* INPUTS: NONE

*10* OUTPUTS: SETS CURRENT ID TO ABORTED STATE

*11* EXTERNALS: $ABRT,$ABRD,$XEN,$LIBR

*12* CALL $LIBR: REENTRANT ROUTINE CALL

*13* GET CURRENT ID FROM XEKT (LOCATION 1717)

*14* INCREMENT TO ADDRESS BOUNDARIES

*15* GET ADDRESS BOUNDARIES

*16* STORE := XVABN PARAMETER LIST

*17* PERFORM XVABN DUMP THIS PARTITION

*18* CALL $ABRD: ABORT THIS PARTITION

*19* CALL $ABRE: FREE CURRENT REENTRANT DATA BLOCK

*20* EXIT TO $XEN: GO TO DISPATCHER

*21* EN D XVABN

*22* ENTRY XVABN

*23* OUTPUTS TO DISK

*24* HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW

*25* CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS

*26* MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION

*27* MEMORY IMAGE AS SPECIFIED BY INPUT ADDRESS RANGE

*28* SCHEDULES XVABN TO FORMAT DATA FROM DISK TO PRINT

*29* EXTERNALS: $LIBR, $LIBX, EXEC, XVABN

*30* CALL $LIBR TO NOTIFY OF REENTRANCY AND BECOME PRIVILEGED

*31* RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH

*32* CALL $LIBX TO BECOME NON-PRIVILEGED

*33* CALL EXEC FOR A GLOBAL ALLOCATION OF DISK

*34* CALL EXEC TO WRITE HEADER DATA USING CLASS 1/0

*35* DO UNTIL ALL ID SEGMENTS PROCESSED

*36* WRITE ID SEGMENT TO DISK

*37* ENDO

*38* WRITE BASE PAGE SEGMENT TO REMAINING PROTON OF THIS TRACK

*39* DO UNTIL END ADDRESS REACHED

*40* WRITE ONE SECTOR OF DATA

*41* ENDO

*42* SCHEDULE XVABN WITH LU AND STARTING TRACK NOS.

*43* CALL $LIBX TO RETURN FROM REENTRANT ROUTINE

*44* XVABN

*45* XVABN

*46* XVABN

*47* XVABN

*48* XVABN

*49* XVABN

*50* XVABN

*51* XVABN

*52* XVABN

*53* XVABN

*54* XVABN

*55* XVABN

*56* XVABN
58 1 *01 TYPE 14 ROUTINE TO CONTROL COMMUNICATION BETWEEN AMD
59 1 *01 EXECUTION OF FDS MANAGER AND IT'S ASSOCIATED TASKS
60 1 *01 (EXECUTIVE, PROCESSORS, AND UTILITIES).
61 1 *01 ENTRY XVPAW AND XVSTB
62 1 *02 INPUTS
63 1 *02 FROM AN ASSOCIATED TASK
64 1 *02 CALL XVPAW(PARMS)
65 1 *02 ASSEMBLY FORM
66 1 *02 JSB XVPAW
67 1 *02 DEF +2 RETURN ADDRESS
68 1 *02 DEF PARMS A(PARMS)
69 1 *02 WHERE PARMS ARE P1,P2,P3,P4,P5
70 1 *02 P1 IS THE SERVICE REQUEST
71 1 *02 0 = NORMAL TERMINATION (P2-P5 NOT USED)
72 1 *02 1 = WORK AREA REQUEST (P2-P5 NOT USED)
73 1 *02 2 = EXECUTE A SEQUENCE TABLE
74 1 *02 (P2-P4 HAS TABLE NAME)
75 1 *02 (P5 INDICATES EXECUTION CONTROL IN CLASS I/O BUFFER)
76 1 *02 3 = RESET SEQUENCE POINTERS
77 1 *02 (P2 HAS SEQUENCE NUMBER) (P3-P5 NOT USED)
78 1 *02 8 = TERMINATE SEQUENCE (P3-P5 NOT USED)
79 1 *02 9 = TERMINATE FDS FUNCTION (P2-P5 NOT USED)
80 1 *02 -32767 = ABNORMAL TERMINATION OF ASSOCIATED TASK
81 1 *02 FROM AN FDS MANAGER
82 1 *02 ASSEMBLY FORM
83 1 *02 JSB XVPAW
84 1 *02 DEF (RETURN POINT)
85 1 *02 OCT 0
86 1 *02 DEF PARMS A(FDS MANAGER RESPONSE)
87 1 *02 OUTPUTS
88 1 *03 TO AN FDS MANAGER
89 1 *03 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
90 1 *03 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
91 1 *03 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
92 1 *03 MANAGER IS ACTIVATED VIA SLIST
93 1 *03 TO AN ASSOCIATED TASK
94 1 *03 RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
95 1 *03 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
96 1 *03 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
97 1 *03 ASSOCIATE TASK IS ACTIVATED VIA SLIST
103 1 BEGIN XVPAM
104 2 CALL BLMRN  BECOME PRIVILEGED
105 2 SET STOP-ID FROM XRT (OCT 1717)
106 2 IF THIS IS A MANAGER RESPONSE
107 2 THEN SET UP TO ACTIVATE ASSOCIATED TASK AND SUSPEND MANAGER
108 3 * CALL SEQUENCE IS RETURN,0,A(PARMS)
109 3 PERFORM XVPAM Post REQUESTOR AND WAIT
110 2 ELSE SET UP TO ACTIVATE MANAGER AND SUSPEND ASSOCIATED TASK
111 3 * CALL SEQUENCE IS RETURN,A(PARMS)
112 3 PERFORM XVPAM Post MANAGER AND WAIT
113 2 ENDIF
114 2 * STOP-ID HAS ID-SEGMENT TO BE SUSPENDED.
115 2 A WAKKEN-ID HAS ID-SEGMENT TO BE ACTIVATED.
116 2 CALL $LIST (SCHEDULE,AWAKEN-ID)
117 2 MOVE RETURN ADDRESS TO X SUSP OF STOP-ID.
118 2 CALL $LIST (WAIT,STOP-ID)
119 1 EXIT TO X:EXEC RTE DISPATCHER
120 1 END XVPAM
1 \textbf{BEGIN} XVPMAN

2 \textbf{* DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER}

3 \textbf{* FIELD IN ID SEGMENTS AS A BACKWARD CHAIN}

4 \textbf{SET TARGET-ID FROM CURRENT-ID-SEGMENT}

5 \textbf{DO WHILE FATHER-ID-NUMBER \textbf{.EQ.} 0 OR FATHER IS WAITING}

6 \textbf{COMPUTE FATHER-ID-SEGMENT FROM FATHER-ID-NUMBER IN TARGET-ID}

7 \textbf{PERFORM MGFRND(FATHER-ID-SEGMENT,COUNT)}

8 \textbf{EXIT IF COUNT \textbf{.GT.} 0}

9 \textbf{SET TARGET-ID TO FATHER-ID-SEGMENT}

10 \textbf{ENDDO}

11 IF FATHER-ID-NUMBER \textbf{.EQ.} 0, OR FATHER NOT WAITING THEN

12 CALL $SYM (12,*XVO3,SEGMENT-NAME) 'XVO3 NAME' REQUESTING PROG

13 \textbf{PERFORM PUMP}

14 EXIT TO X.IOX TO ENABLE AND REDISPATCH

15 \textbf{ENDIF}

16 \textbf{SET AWAKEN-ID FROM FSD-ENTRY SYMN}

17 \textbf{GET REQUEST PARMs MOVE INTO ID-SEGMENT}

18 \textbf{SET STBAT FROM CURRENT-ID}

19 \textbf{FND}

20 \textbf{XVPMAN}

21 \textbf{*}

22 \textbf{*}

23 \textbf{*}

24 \textbf{*}

25 \textbf{*}

26 \textbf{BEGIN XVPMAN}

27 \textbf{* DETERMINE IF CALLER IS A VALID FDS MANAGER}

28 \textbf{*}

29 \textbf{PERFORM MGFRND (CURRENT-ID,COUNT)}

30 \textbf{COUNT WILL BE 0 FOR NO MATCH.}

31 \textbf{COUNT NOT EQUAL ZERO IMPLIES A MATCH}

32 \textbf{AND FSD-ENTRY HAS MATCHING FSDTAB ENTRY ADDRESS}

33 \textbf{IF COUNT \textbf{.LT.} 0 THEN CALLER IS NOT A FDS MANAGER}

34 \textbf{CALL $SYM (12,*XVO1,SEGMENT-NAME) 'XVO1 PROGRAM' REQUESTING PROG.}

35 \textbf{PERFORM PUMP}

36 \textbf{EXIT TO :SEND THE DISPATCHER}

37 \textbf{ENDIF}

38 \textbf{FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER}

39 \textbf{SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK}

40 \textbf{IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST}

41 \textbf{CALL $SYM (12,*XVO2,SEGMENT-NAME) 'XVO2 PROGRAM' ASSOCIATED PROG.}

42 \textbf{PERFORM PUMP}

43 \textbf{EXIT TO :SEND THE DISPATCHER}

44 \textbf{ENDIF}

45 \textbf{AWAKEN-ID-SEGMENT-STAB}

46 \textbf{IF MANAGER HAS REQUEST FOR ABORT, THEN}

47 \textbf{CALL SBART FOR CURRENT AF}

48 \textbf{ENDIF}

49 \textbf{MOVE FD MANAGERS INPUT PARMs TO ASSOCIATED TASK ID SEGMENT}

50 \textbf{MOVE PARMs TO ID-SEGMENT WORDS 2-6}

51 \textbf{END XVPMAN}
172 1 BEGIN PDUMP
172 2  
173 2  
174 2  
175 2  *  SET A CALL TO XVABM AT CALLER'S SUSPEND POINT
176 2  
177 1 END PDUMP
178 1  
179 1  
180 1  
181 1  
182 1 BEGIN MGRFND INPUT IS TARGET-ID
183 2  
184 2  
185 2  SET COUNT TO NUMBER IN TABLE
186 2  
187 2  
188 2  EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMENT
189 2  
190 2  
191 2  
192 2  COUNT = COUNT + 1
193 1 END MGRFND
194 1  
195 1  
196 1  
197 1  
198 1 BEGIN XVSTB
199 2  
200 2  
201 2  
202 2  
203 2  
204 2  
205 2  
206 2  
207 2  
208 2  
209 2  
210 2  
211 2  
212 2  
213 2  
214 2  
215 2  
216 2  
217 2  
218 2  
219 1 END XVSTB
1 BEGIN XXAUT
2 IF ENTRY IS FROM A DIRECTIVE THEN
3 SET MASTERTOT STATE TO INDICATE REENTRY
4 DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
5 SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
6 ERREXIT IF PROCESSOR NOT FOUND TO :ERR:
7 STUFF INTERFACE TABLE BIT AND VERSION INTO SEQUENCE TABLE ENTRY
8 ENDDO
9 CALL IXSTO TO STORE REVISED SEQUENCE TABLE IN ANA AS $SERTB
10 ELSE IF AM BEING REENTERED FROM I.N.
11 CALL IXSTO TO SET UP TEMPORARY EX-Y W$IN $INTAB
12 CALL IXEXE TO EXECUTE FROM TEMPORARY ENTRY
13 IF RESET SEQUENCE NUMBER IS NOT REQUESTED THEN
14 EXIT XXAUT IF TERMINAL ENTRY WAS JUST EXECUTED
15 SET STARTING ENTRY TO NEXT ENTRY
16 ENDDO
17 ENDF
18 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
19 CALL IXEXE TO EXECUTE REMAINDER OF TABLE
20 ENDDO
21 EXIT XXAUT
22 :ERR1:
23 CALL IXMSG TO DISPLAY INVALID PROCESSOR NAME
24 1 END XXAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELDS (XICNT)

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

INPUTS FROM CALLING SEQUENCE:

XICNT - (INTEGER, 3 WORDS) ARRAY CONTAINING THE NAME "XICNT" USED BY XELDS TO CALL EXEC 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 (XE(5), NASTA)

F6S ROUTINES USED

XERT, XEXT, XRMSE, XHAR

NOTE: CONTAINS DUMMY CALL TO XEXEC
BEGIN
100 CASE 12 AND 15 OF MASTER
200 CALL NAME; : NAME; : AUTO:
300 END
132 1 C0************
134 1 C0                   FORTRAN CALLING SEQUENCE:
135 1 C0                   CALL XDEC (RETC)
137 1 C0************
140 1 C0                   XDEC DECODES A RESPONSE OF PROCESSOR NAME (.INT TABLE NAME)
141 1 C0                   INTO A SEQUENCE TABLE ENTRY.
143 1 C0************
145 1 C0                   INPUTS IN COMMON:
146 1 C0                   XE(85) TOKENS, XE(145) COMBUF, XB(1) NPROC, XB(2) LIBD
149 1 C0************
150 1 C0                   OUTPUTS IN CALLING SEQUENCE:
152 1 C0                   RETC - RETURN CODE (O IS NORMAL RETURN)
154 1 C0                   OUTPUTS IN COMMON:
156 1 C0                   XE(16) PRCNAM, XB(251) SEQTAB
158 1 C0************
161 1 C0                   COMMON USED:
162 1 C0                   EQUIVALENCE (XE(16), PRCNAM),
164 1 C0                   + (XE(85), TOKEHS), (XE(144), TOKPTR),
165 1 C0                   + (XE(145), COMBUF),
166 1 C0                   + (XB(1), NPROC), (XB(2), LIBD ),
167 1 C0                   + (XB(251), SEQTAB)
168 1 C0                   FDS ROUTINES USED:
170 1 C0                   XRCPR, XREX, XRMV, XRMSG
173 1 C0                   RTE ROUTINES USED:
174 1 C0                   IAND
177 1 C0************
179 1 BEGIN XXDEC
180 2 INITIALIZE RETURN CODE TO ZERO
181 2 SET SEQUENCE ENTRY TO ZEROS
182 2 ERREXIT IF FIRST TOKEN IS NOT A PROCESSOR NAME TO :ERR1:
183 2 SEARCH LIBRARY DIRECTORY FOR PROCESSOR
184 2 ERREXIT IF NAME NOT FOUND TO :ERR1:
185 2 MOVE PROCESSOR NAME, IT BIT AND VERSION INTO SEQUENCE ENTRY
186 2 IF AN INTERFACE TABLE NAME WAS ENTERED THEN
187 2 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
188 2 ENDF
189 2 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
190 2 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
191 2 IF AN INTERFACE TABLE IS NEEDED BUT NOT SPECIFIED THEN
192 2 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'EinTab'
193 2 ENDF
194 1 EXIT XXDEC
195 2 :ERR1:
196 2 CALL XIAGSE TO DISPLAY ERROR
197 2 SET RETURN CODE TO SAY ERROR
198 1 END XXDEC
**FORTRAN CALLING PROCEDURE:**

CALL XXDEF

XXDEF READS IN THE DEFAULT INTERFACE TABLE FOR A PROCESSOR AND STORES IT IN THE ANA AS GINTAB

**INPUTS FROM COMMON:**

PROCAB - (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 DEFAULT INTERFACE TABLE CAN BE READ INTO

DEFNAM - (INTEGER, 3 WORDS) ARRAY WHERE INTERFACE TABLE NAME IS CREATED FROM PROCESSOR NAME

**COMMON USED:**

EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA),

+ (XE(13), INTMAN), (XE(16), PRCHN),

+ (XE(19), RECPT), (XE(20), REBUF),

+ (XE(142), ICRI),

+ (XS(16), TMPTAB), (XS(14), DEFNAM),

+ (XS(18), LEN1), (XS(19), LEN2),

+ (XS(20), IDC), (XS(200), RETC)

**RTE ROUTINES USED:**

CLOSE, EXEC, KCVT, OPEN, READF

**FDX ROUTINES USED:**

XERTN, XREQ, XREXT, XRMV, XRMG, XRPRK, XRUPK
BEGIN XXDEF
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 :MGERR:
IF DEFAULT TABLE IS INCOMPLETE THEN
SET INTERFACE TABLE NAME IN XE TO GINTAB
SET SUBSTATE TO INTERFACE TABLE EDITOR
CALL XERTN TO IMPLICITLY CALL INT EDITOR **NO RETURN**
ENDIF
1 EXIT XXDEF

:FILERR:
CALL CLOSE TO CLOSE FILE
CALL XRMG TO DISPLAY FILE ACCESS ERROR
1 EXIT XXDEF

:MGERR:
CALL XRMG TO DISPLAY SPACE ERROR
CALL EXEC TO FREE CLASS NUMBER
SET MASTER STATE TO DIRECTIVE LEVEL
CALL XERTN TO RETURN TO EXEC **NO RETURN**
1 END XXDEF
FORTRAN CALLING PROCEDURE:

CALL XXEXE

INPUTS FROM COMMON:
X(10) SERSTA, X(11) SEREND, X(12) SERPTR,
X(250) SERLEN, X(251) SECTAB, X(13) FLGTAB

OUTPUTS TO COMMON:
X(6) SUBSTA, X(10) SERSTA, X(12) SERPTR,
X(13) INTHAM, X(16) PRCHAM, X(235) RESIND

COMMON USED:
EQUIVALENCE (X(E5), MASSTA), (X(E6), SUBSTA),
+ (X(E10), SERSTR), (X(E11), SEREND),
+ (X(E12), SERPTR), (X(E13), INTHAM),
+ (X(E16), PRCHAM), (X(E23), REBUFF),
+ (X(250), SEGLT), (X(249), SERNO),
+ (X(251), SECTAB),
+ (X(15), PARM), (X(13), FLGTAB)

FDS ROUTINES USED:
XREQ, XREXT, XR16, XRMOV, XRMSG,
XSET, XVPAW, XXDEF, XXTFP

RTE ROUTINES USED:
AMPAR
1 BEGIN XX:
2 INITIATE RESET INDEX TO ZERO
3 IF TABLE FLAG SAYS SENTAB IN X8 THEN
4 CALL XVPAM WITH SEQ# START AND END
5 ELSE - AM BEING CALLED TO EXECUTE A TEMPORARY SEQUENCE TABLE
6 CALL XVPAM WITH SEQUENCE POINTER AS START AND END
7 ENDIF
8 CALL RMPAR TO RETRIEVE RETURN PARAMETERS
9 IF THE MANAGER DETECTED A ERROR TRYING TO EXECUTE THE SEQUENCE THEN
10 FIND ENTRY IN ERROR AND SAVE IN XE
12 :INT1: INTERFACE TABLE NOT SPECIFIED
13 CALL X DEFIN TO READ UP DEFAULT TABLE
14 SET SUBSTATE TO INTERFACE TABLE EDITOR
15 CALL XEIN TO RETURN TO EXEC **NO RETURN**
16 :INT2: INTERFACE TABLE NOT COMPLETE
17 :ERR2: SET MESSAGE TO INT TABLE NOT FOR PROCESSOR
18 :ERR3: SET MESSAGE TO VERSIONS DO NOT MATCH
19 :ERR4: SET MESSAGE TO SEQ NOT FOUND
20 :ERR5: SET MESSAGE TO PROCESSOR ABENDED
21 :ERR6: SET MESSAGE TO AWA OVERFLOW
22 ENDCASE
23 CALL XRMG TO DISPLAY ERROR
24 Format SEQUENCE ENTRY INTO ASCII
25 CALL XRMG TO DISPLAY SEQUENCE TABLE ENTRY IN ERROR
26 IF MODE IS SEMI-AUTO AND ENTRY IS NOT AN OVERRIDE THEN
27 RESET OLD INDEX TO RE-EXECUTE THIS ENTRY
28 ENDIF
29 IF MODE IS AUTO THEN
30 SET MASTER STATE TO ZERO
31 CALL XEIN TO ABORT SEQUENCE *** NO RETURN ***
32 ENDIF
33 ENDIF
34 RETURN
35 END XX:
### Procedure for Calling XIMAN:

**CALL XIMAN**

XIMAN IS THE MANUAL EXECUTION CONTROLLER

**INPUTS FROM COMMON:** XE(249) SENO

**Outputs to Common:**
- XE(10) SERTA, XE(11) SEHON, XE(249) SEKO, XE(250) SELON, XE(251) SEF7A8.

**Common Used:**
- EQUIVALENT (XEC5), XEC5, XEC5, XEC5, SERTA, SERTA, SERTA,
- EQUIVALENT (XEC5), XEC5, XEC5, XEC5, SERTA, SERTA, SERTA.

**RTE Routines Used:**
- XEC5, XESE, XESE, XESE, XESE, XESE.

**PS Routines Used:**
- XEC5, XESE, XESE, XESE, XESE.

---

5-315
1 BEGIN XXMANG
2 IF ENTRY IS FROM A DIRECTIVE THEN
3 SET HASHTO TO INDICATE RE-ENTRY
4 DO UNTIL PERCENT IS ENTERED
5
6 :PROMPT: CALL XICOM TO PROMPT FOR PRNAME,"ITNAME
7 IF PERCENT IS NOT ENTERED THEN
8 ERREXIT IF CR ENTERED TO :PROMPT:
9 CALL XIDEC TO DECODE RESPONSE
10 ERREXIT IF INVALID RESPONSE TO :PROMPT:
11 SET SER #5 IN ZE .1 ZEROS
12 SET ENTRENTS IN SERTAC IN XE TO 1
13 CALL XIXTO TO STORE SEQUENCE TABLE
14 IF IT NAME IN SENTAB IS 8INTAB THEN
15 PROCESSOR REQUIRES AN IT THEN
16 CALL XIXDEF TO READ UP DEFAULT INTERFACE TABLE
17 ENIF
18 CALL XXEXE TO EXECUTE SERTAB
19 ENIF
20 ENDDO
21 ELSE - AM BEING REENTERED FROM INTERFACE TABLE EDITOR
22 CALL XIXMP TO SET UP TO EXECUTE A TEMPORARY TABLE
23 CALL XXEXE TO EXECUTE ENTRY
24 PERFORM XXMANG **NO RETURN**
25 ENDIF
26 RETURN
27 :PRMERR: CALL XRMSC TO DISPLAY ERROR
28 PERFORM XXMANG **NO RETURN**
29 END XXMANG
CALL XSEM

**Fortran Calling Sequence:**

XSEM

**Inputs in Common:**

- XE(5) MASSTA
- XE(10) SERSRA
- XE(11) SEREND
- XE(12) SEQ'TRA
- XE(140) TABEND
- XE(1) NPROC
- XE(2) LIBD
- XE(245) SENNO
- XE(250) SELEN
- XE(251) SERTAB

**Outputs in Common:**

- XE(5) MASSTA
- XE(13) FLGTAB

**Internal Common Used:**

- XE(139) EXEND - Ending sequence number used to terminate sequence
- XE(161) CURIND - Index to the current entry being executed
- XE(235) RESIND - Index to reset current index to
- XE(236) ASCENT - Exit sequence table entry used to prompt the user
- XE(246) OLDIND - Index to the last entry executed in the sequence table

**Common Used:**

- (XE(5), MASSTA)
- (XE(10), SERSRA)
- (XE(11), SEREND)
- (XE(12), SERPRT)
- (XE(85), EOS )
- (XE(86), INTR )
- (XE(100), AMPER )
- (XE(139), EXEND)
- (XE(141), CURIND)
- (XE(143), COMBUS )
- (XE(17), NPROC)
- (XE(22), LBD )
- (XE(235), RESIND)
- (XE(236), ASCENT )
- (XE(240), SENNO )
- (XE(249), SERTAB )
- (XE(251), SERTE )
- (XE(13), FLGTAB )

**FDS Routines Used:**

- XREP, XREX, XRIG, XRMDY
- XREPG, XRES, XREDC, XRED, XXEYF
- XSTO, XSTHP, XSTCM
509 1 BEGIN XSEM:
510 2 IF ENTRY IS FROM A DIRECTIVE THEN
511 3 SET MASTER STATE TO SAT REENTRY
512 4 DO FOR # ENTRY IN SEQUENCE TABLE
513 5 SEARCH LIST FOR PROCESSOR NAME
514 6 ERROR IF NOT FOUND TO :ERRJ:
515 7 MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
516 8 ENDDO
517 9 SAVE ORIGINAL ENDING SEQUENCE #
518 10 SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
519 11 SET FLSTAB TO SAT EXECUTE ENTIRE SEQUENCE
520 12 CALL XISTO TO STORE REST OF AWA
521 13 ELSE (I AM BEING REENTERED FROM INTE)
522 14 CALL XETLP TO SET UP TEMPORARY EXECUTION
523 15 CALL XERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQUENCE
524 16 IF RESET WAS REQUESTED THEN
525 17 RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
526 18 ELSE
527 19 EXIT XEXT IF TERMINAL ENTRY WAS JUST EXECUTED
528 20 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
529 21 ENDF
530 22 ENDF
531 23 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
532 24 CALL XTCMP TO PROMPT USER WITH CURRENT ENTRY
533 25 EXIT XSEM IF RESPONSE IS X
534 26 IF RESPONSE IS CR THEN
535 27 IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
536 28 CALL XDEF TO READ UP DEFAULT TABLE
537 29 ENDF
538 30 SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
539 31 SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
540 32 CALL XERE TO EXECUTE
541 33 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
542 34 ELSE
543 35 IF RESPONSE WAS A SEQUENCE # THEN
544 36 ERROR IF NUMBER IS ZERO TO :ERR1:
545 37 SEARCH SEQUENCE TABLE FOR SEQUENCE #
546 38 ERROR IF NUMBER IS NOT FOUND TO :ERR1:
547 39 SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
548 40 ELSE
549 41 IF RESPONSE IS AN AMPIERSAND THEN
550 42 SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
551 43 ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
552 44 CALL XEXED TO DECODE PROCESSOR NAME, IT NAME RESPONSE
553 45 IF RESPONSE IS VALID THEN
554 46 CALL XISTO TO STORE OVERRIDING ENTRY
555 47 SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
556 48 SET CURRENT SEQUENCE # TO FIRST ENTRY
557 49 ENDF
558 50 ENDF
559 51 ENDF
560 52 ENDO
561 53 ENDF
562 54 END XSEM
563 55 :ERR1:
564 56 CALL XERMS TO DISPLAY ERROR MESSAGE
565 57 END XSEM
CALLING PROCEDURE FOR XXSTO:

CALL XXSTO

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

INPUTS FROM COMMON:
XB(250) SERLEN, XB(251) SERTAB, XS(13) FLGTAB

OUTPUTS TO COMMON:
XE(5) MASSTA

COMMON USED:

EQUIVALENCES: (XE(5), MASSTA), (XE(19), REQ(19)), (XE(20), REQ(20)), (XE(250), SERLEN), (XE(251), SERTAB), (XE(13), FLGTAB)

FDS ROUTINES USED:
XREQ, XRMOV, XRMSG, XERTM

RTE ROUTINES USED:
EXEC
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>609</td>
<td>BEGIN XXSTO</td>
<td>BEGIN XXSTO</td>
</tr>
<tr>
<td>610</td>
<td>SET CLASS NUMBER TO ZERO</td>
<td>SET CLASS NUMBER TO ZERO</td>
</tr>
<tr>
<td>611</td>
<td>IF TABLE FLAG SAYS STORE ENTIRE TABLE THEN</td>
<td>IF TABLE FLAG SAYS STORE ENTIRE TABLE THEN</td>
</tr>
<tr>
<td>612</td>
<td>CALL EXEC TO WRITE ENTIRE TABLE</td>
<td>CALL EXEC TO WRITE ENTIRE TABLE</td>
</tr>
<tr>
<td>613</td>
<td>SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN XB</td>
<td>SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN XB</td>
</tr>
<tr>
<td>614</td>
<td>CALL XMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER</td>
<td>CALL XMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER</td>
</tr>
<tr>
<td>615</td>
<td>ELSE</td>
<td>ELSE</td>
</tr>
<tr>
<td>616</td>
<td>CALL EXEC TO WRITE ONLY FIRST ENTRY</td>
<td>CALL EXEC TO WRITE ONLY FIRST ENTRY</td>
</tr>
<tr>
<td>617</td>
<td>SET LENGTHS IN REQUEST BUFFER TO 7 WORDS</td>
<td>SET LENGTHS IN REQUEST BUFFER TO 7 WORDS</td>
</tr>
<tr>
<td>618</td>
<td>CALL XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER</td>
<td>CALL XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER</td>
</tr>
<tr>
<td>619</td>
<td>ENDBF</td>
<td>ENDBF</td>
</tr>
<tr>
<td>620</td>
<td>CALL XER TO REQUEST MANAGER TO STORE .SERST</td>
<td>CALL XER TO REQUEST MANAGER TO STORE .SERST</td>
</tr>
<tr>
<td>621</td>
<td>IF RETURN CODE IS NOT ZERO THEN</td>
<td>IF RETURN CODE IS NOT ZERO THEN</td>
</tr>
<tr>
<td>622</td>
<td>CALL XMPG TO WRITE SPACE ERROR</td>
<td>CALL XMPG TO WRITE SPACE ERROR</td>
</tr>
<tr>
<td>623</td>
<td>CALL EXEC TO RELEASE CLASS #</td>
<td>CALL EXEC TO RELEASE CLASS #</td>
</tr>
<tr>
<td>624</td>
<td>SET MASTER STATE TO ZERO</td>
<td>SET MASTER STATE TO ZERO</td>
</tr>
<tr>
<td>625</td>
<td>CALL XERR TO RETURN TO EXEC <strong>NO RETURN</strong></td>
<td>CALL XERR TO RETURN TO EXEC <strong>NO RETURN</strong></td>
</tr>
<tr>
<td>626</td>
<td>ENDBF</td>
<td>ENDBF</td>
</tr>
<tr>
<td>627</td>
<td>RETURN</td>
<td>RETURN</td>
</tr>
<tr>
<td>628</td>
<td>1 END XXSTO</td>
<td>1 END XXSTO</td>
</tr>
</tbody>
</table>
FORTRAN CALLING SEQUENCE:

CALL XXTMP

XXTMP SETS UP A ONE ENTRY SEQUENCE TABLE USING BINTAB AND STORES IT IN THE AREA TO EXECUTE WHEN EXECUTION WITH A TEMPORARY ENTRY IS NECESSARY.

INPUTS FROM COMMON:

XE(12) SEQPTR, XB(250) SEQLEN, XB(251) SEQTAB

OUTPUTS TO COMMON:

XE(6) SUBSTA, XB(249) SEQNO, XB(250) SEQLEN, XB(251) SEQTAB, XE(13), FLGTAB

COMMON USED:

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

* (XE(20), REBUF),

* (XB(249), SEQNO), (XB(250), SEQLEN),

* (XB(251), SEQTAB), (XE(6), TMPTAB),

FDX Routines Used:

XCR, XINOV, XSTO

RTE Routines Used:

EXEC
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>675</td>
<td>BEGIN XXTMP</td>
<td></td>
</tr>
<tr>
<td>676</td>
<td>SET SUBSTATE TO EXECUTION CONTROLLER</td>
<td></td>
</tr>
<tr>
<td>677</td>
<td>CALL XREQ TO RETRIEVE EVENTS</td>
<td></td>
</tr>
<tr>
<td>678</td>
<td>CALL EXEC TO GET SCHEDULE TABLE</td>
<td></td>
</tr>
<tr>
<td>679</td>
<td>SEARCH SMTAB TO FIND ENTRY IN ERROR</td>
<td></td>
</tr>
<tr>
<td>680</td>
<td>CREATE TEMPORARY ENTRY IN SMTAB</td>
<td></td>
</tr>
<tr>
<td>681</td>
<td>SET ITNAME IN TEMP ENTRY TO SMTAB</td>
<td></td>
</tr>
<tr>
<td>682</td>
<td>SET TABLE FLAG TO SAT TEMPORARY ENTRY ONLY</td>
<td></td>
</tr>
<tr>
<td>683</td>
<td>CALL XXTO TO STORE EVENTS</td>
<td></td>
</tr>
<tr>
<td>684</td>
<td>RETURN</td>
<td></td>
</tr>
<tr>
<td>685</td>
<td>END XXTMP</td>
<td></td>
</tr>
</tbody>
</table>
ASSIGN - DATA ASSIGNMENT PROCESSOR

- SCHEDULED BY FDS

ASSIGN ALLOWS THE FDS USER TO COMPUTE VALUES AND STORE THEM IN AN EXISTING DATA ELEMENT IN THE ANA. ASSIGN SUPPORTS EXTENDED FORTRAN TYPE MIXED-MODE EXPRESSIONS AND FUNCTIONS AND ALLOWS REPEATED 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 - ASGC 434 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>PARM(1) = LU, PARM(2) = DEBUG FLAGS</td>
</tr>
<tr>
<td>TOKENS</td>
<td>32</td>
<td>6</td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STWIDE</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>LASTZY</td>
<td>1</td>
<td>40</td>
<td>LAST SYMBOL TABLE ENTRY DEFINED</td>
</tr>
<tr>
<td>SYNTAB</td>
<td>12,80</td>
<td>41</td>
<td>SYMBOL TABLE (WORDS 1-8 = TOC ENTRY OR APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR SUBSCRIPTED DATA ELEMENT)</td>
</tr>
<tr>
<td>SSTRING</td>
<td>247</td>
<td>1013</td>
<td>RESULT STACK USED DURING POST-FIX STRING EVALUATION (EACH ENC</td>
</tr>
<tr>
<td>OPERAND</td>
<td>VALUE</td>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>61 C04</td>
<td></td>
<td>-2 DISPLACEMENT</td>
<td></td>
</tr>
<tr>
<td>62 C04</td>
<td></td>
<td>-3 CHARACTER STRING INDEX</td>
<td></td>
</tr>
<tr>
<td>63 C04</td>
<td>OPRNDS</td>
<td>4,9 STACK CONTAINING OPERANDS FOR FUNCTIONS AND ARITHMETIC OPERATIONS AND RESULTS FOR STORAGE (EA. ENTRY: WORDS 1-3 CONTAIN VALUE; WORD 4 = DATA TYPE) SIZED FOR 72 CHARACTER STRING</td>
<td></td>
</tr>
<tr>
<td>64 C04</td>
<td>OPINFO</td>
<td>7 CONTENTS OF FNCBL OR SYNTAX FOR FUNCTION OR MATHEMATICAL OPERATION BEING EVALUATED</td>
<td></td>
</tr>
<tr>
<td>65 C04</td>
<td>NUMWDS</td>
<td>1 NUMBER OF WORDS TO BE STORED IN OBJECT DATA ELEMENT</td>
<td></td>
</tr>
<tr>
<td>66 C04</td>
<td>DATTPS</td>
<td>9 DATA TYPES</td>
<td></td>
</tr>
<tr>
<td>67 C04</td>
<td>RSLTPY</td>
<td>1 RESULT STACK POINTER</td>
<td></td>
</tr>
<tr>
<td>68 C04</td>
<td>EXPRTR</td>
<td>1 POST-FIX STRING (POLISH) POINTER</td>
<td></td>
</tr>
<tr>
<td>69 C04</td>
<td>CLSREN</td>
<td>1 XPREF OPTION WORD FOR QUEUE REQUEST AND CLOSE BUFFER - NO</td>
<td></td>
</tr>
<tr>
<td>70 C04</td>
<td>CLSTRN</td>
<td>1 1500 XPREF OPTION WORD FOR QUEUE REQUEST, CLOSE BUFFER AND TRANSFER DATA</td>
<td></td>
</tr>
<tr>
<td>71 C04</td>
<td>MAPWDS</td>
<td>9 NUMBER OF WORDS PER LOGICAL UNIT OF DATA FOR EACH DATA TYPE</td>
<td></td>
</tr>
<tr>
<td>72 C04</td>
<td>DECLAS</td>
<td>1 DATA ELEMENT CLASS</td>
<td></td>
</tr>
<tr>
<td>73 C04</td>
<td>RNGSTR</td>
<td>4,4 ACFIN RANGE, END RANGE, INCREMENT AND SYMBOL INDEX FOR EACH RANGE SPECIFICATION</td>
<td></td>
</tr>
<tr>
<td>74 C04</td>
<td>POLISH</td>
<td>161 POST-FIX REPRESENTATION OF EXPRESSION</td>
<td></td>
</tr>
<tr>
<td>75 C04</td>
<td>SYNTAX</td>
<td>7,40 SYNOPSIS TABLE FOR VALIDITY TESTS ON EXPRESSION (SEE BELOW)</td>
<td></td>
</tr>
<tr>
<td>76 C04</td>
<td>FNCBL</td>
<td>7,36 FUNCTION TABLE CONTAINING DATA REQUIREMENTS FOR EACH FUNCTION</td>
<td></td>
</tr>
<tr>
<td>77 C04</td>
<td>MRBUFF</td>
<td>64 2220 XPREF BUFFER</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: STACKS USED IN THE ASSGN PROCESSOR ARE SIZED FOR THE MAXIMUM POSSIBLE AND OVERFLOW IS NOT TESTED.

TABLE DEFINITIONS:
<table>
<thead>
<tr>
<th>CDA</th>
<th>Syntax - 7x40 Table Containing Information for Processing Each Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>CDA</td>
</tr>
<tr>
<td>108</td>
<td>CDA</td>
</tr>
<tr>
<td>109</td>
<td>CDA</td>
</tr>
<tr>
<td>110</td>
<td>CDA</td>
</tr>
<tr>
<td>111</td>
<td>CDA</td>
</tr>
<tr>
<td>112</td>
<td>CDA</td>
</tr>
<tr>
<td>113</td>
<td>CDA (EOS)</td>
</tr>
<tr>
<td>114</td>
<td>CDA INTEGER</td>
</tr>
<tr>
<td>115</td>
<td>CDA REAL</td>
</tr>
<tr>
<td>116</td>
<td>CDA DOUBLE</td>
</tr>
<tr>
<td>117</td>
<td>CDA</td>
</tr>
<tr>
<td>118</td>
<td>CDA NAME</td>
</tr>
<tr>
<td>119</td>
<td>CDA</td>
</tr>
<tr>
<td>120</td>
<td>CDA</td>
</tr>
<tr>
<td>121</td>
<td>CDA</td>
</tr>
<tr>
<td>122</td>
<td>CDA</td>
</tr>
<tr>
<td>123</td>
<td>CDA</td>
</tr>
<tr>
<td>124</td>
<td>CDA</td>
</tr>
<tr>
<td>125</td>
<td>CDA</td>
</tr>
<tr>
<td>126</td>
<td>CDA</td>
</tr>
<tr>
<td>127</td>
<td>CDA</td>
</tr>
<tr>
<td>128</td>
<td>CDA</td>
</tr>
<tr>
<td>129</td>
<td>CDA</td>
</tr>
<tr>
<td>130</td>
<td>CDA</td>
</tr>
<tr>
<td>131</td>
<td>CDA</td>
</tr>
<tr>
<td>132</td>
<td>CDA</td>
</tr>
<tr>
<td>133</td>
<td>CDA</td>
</tr>
<tr>
<td>134</td>
<td>CDA</td>
</tr>
<tr>
<td>135</td>
<td>CDA</td>
</tr>
<tr>
<td>136</td>
<td>CDA</td>
</tr>
<tr>
<td>137</td>
<td>CDA</td>
</tr>
<tr>
<td>138</td>
<td>CDA</td>
</tr>
<tr>
<td>139</td>
<td>CDA (BKSLASH)</td>
</tr>
<tr>
<td>140</td>
<td>CDA</td>
</tr>
<tr>
<td>141</td>
<td>CDA</td>
</tr>
<tr>
<td>142</td>
<td>CDA</td>
</tr>
<tr>
<td>143</td>
<td>CDA</td>
</tr>
<tr>
<td>144</td>
<td>CDA</td>
</tr>
<tr>
<td>145</td>
<td>CDA</td>
</tr>
<tr>
<td>146</td>
<td>CDA</td>
</tr>
<tr>
<td>147</td>
<td>CDA</td>
</tr>
<tr>
<td>148</td>
<td>CDA</td>
</tr>
<tr>
<td>149</td>
<td>CDA</td>
</tr>
<tr>
<td>150</td>
<td>CDA</td>
</tr>
<tr>
<td>151</td>
<td>CDA UNARY</td>
</tr>
<tr>
<td>152</td>
<td>CDA FUNCTION</td>
</tr>
<tr>
<td>153</td>
<td>CDA</td>
</tr>
<tr>
<td>154</td>
<td>CDA VR = VARIABLE</td>
</tr>
<tr>
<td>155</td>
<td>CDA RB = RIGHT BRACKET</td>
</tr>
</tbody>
</table>

- **Token**
- **Input Priority**
- **Output Priority**
- **Token #Pnnds-1**
- **Data Type**
- **Precedent Type**
- **Valid Precedents**

**Columns:**
- **OP** (Operator)
- **LB** (Left Bracket)
- **RB** (Right Bracket)
- **Variable**
- **Constant**

**Example Rows:**
- CDA INTEGER 4 OP(LBRT),= 4 OP(LBRT),=
- CDA REAL 4 OP(LBRT),=
- CDA DOUBLE 4 OP(LBRT),=
- CDA NAME 2 OP(LBRT),=MUL
- CDA VR = VARIABLE 1 OP = OPERATOR LB = LEFT BRACKET CN = CONSTANT
<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>ASIN</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></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>REAL</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>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DCM</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>DOUBLE</td>
</tr>
<tr>
<td>DSIGN</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DSGN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DSGNT</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>DTHN</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>INTEGER</td>
</tr>
<tr>
<td>IABS</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IDINT</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IFIX</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</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>REAL</td>
</tr>
<tr>
<td>SIN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SML</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SORT</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></td>
</tr>
</tbody>
</table>

**FMCTRL - 7X36 ABLE CONTAINING INFORMATION FOR PROCESSING FUNCTIONS**
Routines referenced - rmpar, xpgat, xrmov, xi?si, xips2, xpmx

Backus-Naur Language Definition

```
<assignment> ::= <replacement> <range> //
<evaluation> ::= <range> //
<replacement> ::= <non-numeric def> = <non-numeric def> //
<non-numeric def> ::= "character string" //
<replacement> ::= <variable> = free data element //
<non-numeric def> ::= free data element //
<expression> ::= free data element <subscript list> //
<subscript list> ::= <expression> //
<expression> ::= <expression> <additive operator> <term> //
<term> ::= <term> <multiplicative operator> <factor> //
<factor> ::= <power>**<power> //
<factor> ::= <power> //
<power> ::= <expression> //
<operator> ::= <additive operator> //
```

- The reproduction of the original page is poor.
260 1 CD5  <CONSTANT> ::= INTEGER  //      ASSIGN
261 1 CD5  SINGLE PRECISION REAL  //      ASSIGN
262 1 CD5  DOUBLE PRECISION REAL   //      ASSIGN
263 1 CD5  <RANGE> ::= <RANGE> <LIMITS>  //      ASSIGN
264 1 CD5  <RANGE> <LIMITS>       //      ASSIGN
265 1 CD5  <LIMITS> ::= <INDEX>=INTEGER,INTEGER     //      ASSIGN
266 1 CD5  <LIMITS> ::= <INDEX>=INTEGER,INTEGER     //      ASSIGN
267 1 CD5  <LIMITS> ::= <INDEX>=INTEGER,INTEGER     //      ASSIGN
268 1 CD5  <LIMITS> ::= <INDEX>=INTEGER,INTEGER     //      ASSIGN
269 1 CD5  <LIMITS> ::= <INDEX>=INTEGER,INTEGER     //      ASSIGN
270 1 CD*******
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:

XIDIN - READS AND EDITS INTERFACE TABLE

XIDP1 - READS ORDE A, VALIDATES NAMES IN INTERFACE TABLE AGAINST NAMES IN DATABOX SCAN. WITH SUBROUTINE XIDM, IT DEVELOPS CONSTRAINT Masks FOR ARRAYS

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

DBDSP MERELY CALLS THESE OVERLAYS IN THE PROPER ORDER AND EXITS

INPUTS TO DBDSP FROM INTERFACE TABLE:

DATABOX - DATA BOX FILE NAME
NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER
KEEP - DISPLAY VARIABLE SCALE LIST SET UP BY USER
VIDEF - CONSTRAINT VARIABLE DEFINITION LIST

INPUTS TO DBDSP FROM ORDE FILE:

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

RECORD 2
(1M) NO OF ENTRIES IN SUMMARY TABLE
(2) X SCAN VARIABLE (6 CHAR)
(12) - X INCREMENT (REAL)
(14) - X NUMBER OF STEPS (INTEGER 1-5)
(15) - YSCAN VARIABLE (6 CHAR)
(18) - Y FIRST SUBSCRIPT (INT OR ZERO)
(19) - Y SECOND SUBSCRIPT (INT OR ZERO)
(20) - Y UNITS (6 CHAR)
(23) - Y CENTROID (REAL)
(25) - Y(INCREMENT (REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPENDANT VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

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

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

CD  1  |  COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPRM
CD  1  |  DATA - NAME OF DATA BOX TO BE DISPLAYED BY BBOSP
CD  1  |  DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
CD  1  |  IDV1 - POINTER TO DEP DISPLAY VARIABLE NAME LIST FOR FIRST VAR
CD  1  |  IDV2 - POINTER TO DEP DISPLAY VARIABLE NAME LIST FOR 2ND VAR
CD  1  |  JSAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)
CD  1  |  JSMNT1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD  1  |  JSMNT2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD  1  |  JTMNT1 - NAME OF COMMON AREA USED FOR INTERFACE TABLE
CD  1  |  JTMNT2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
CD  1  |  LBUF - LENGTH OF COMBUF IN 16-BIT WORDS - XPRM
CD  1  |  LENST - LENGTH IN WDS OF CHAR STRING USED FOR USER PROMPT -XPRM
CD  1  |  MASK - ARRAY CONTAINING MASKS FOR UP TO 8 CONSTR (4 X 121 SIZE)
CD  1  |  MXVAR - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCM
CD  1  |  MC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
CD  1  |  MCERR - LIST OF CONSTRAINT RELATIONS INPUT BY USER
CD  1  |  NCMVAR - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
CD  1  |  NCVAR - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
CD  1  |  MCVAR - NUMBER OF DEP DISP VAR IN NCVAR LIST (INTEGER)
CD  1  |  NCVAR - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P
CD  1  |  PRED - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT = 1
CD  1  |  MASEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (1 TO 5)
CD  1  |  NYSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (1 TO 5)
CD  1  |  RETC - RTM TOLERANCE PROMPT - XPRM, PRIOR TO EXECUTING ERR/ERR
CD  1  |  RE - STRING CONTAINING USER PROMPT MESSAGE
CD  1  |  XCOORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
CD  1  |  XSCNNM - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
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 YSCANN - 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 SUNITB VARIABLE NAMES AND UNITS
402 1 CD 4 NAMVUL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCH
403 1 CD 4 SUMTAB - VALUES FOR SCANVARIABLE(S) 1 TO 32 VALUES/RECORD
404 1 CD 4 PARGS - COMMUNICATION BUFFER FOR RMPAR = Lu, USER ID, FLAGS
405 1 CD 4 LU - LOGICAL UNIT # FOR PRDM CALLING SEQUENCE - USER LOCATM
406 1 CD 4 LDBG - 0DSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
407 1 CD 4 PRBLP - TABLE IN COMMON TO COMMUNICATE WITH PRDM
408 1 CD 4 DEBUG
409 1 CD 4 SELECT - SELECT =0 PROMPT; SELECT =0 RUN ALL DISPLAYS TO O/P
410 1 CD 4 CARTS - CARTAGE USED TO LOCATE DATA BOX
411 1 CD 4 CARTG - CARTAGE USED TO LOCATE DATA BOX
412 1 CD 4 CVLUE -
413 1 CD 5*******
414 1 CD 5*******
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

XPRDM, XELBS, XPXIT, RMPAR
431 1 CO************
432 1 CD0             DEFIN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER
433 1 CD0
434 1 CD0
435 1 CD0************
436 1 CD1             DEFIN ALLOCATES DATA ELEMENTS IN THE ANA THAT WERE SPECIFIED
437 1 CD1             BY THE PARAMETER KEYWORD DEFINE. IF THE DATA ELEMENT ALREADY
438 1 CD1             EXISTS, IT IS DELETED AND REALLOCATED. DATA ELEMENTS ARE
439 1 CD1             INITIALIZED TO ZERO (CHARACTER STRINGS TO BLANKS).
440 1 CD1************
441 1 CD2             INPUTS FROM THE MANAGER:
442 1 CD2
443 1 CD2             LU - LOGICAL UNIT OF THE USER'S TERMINAL
444 1 CD2             DEBUG - FLAGS FOR DEBUG
445 1 CD2************
446 1 CD2             INPUTS FROM THE INTERFACE TABLE:
447 1 CD2
448 1 CD2             DEFINE - SYMBOLIC STRING CONTAINING DATA ELEMENT NAME(S),
449 1 CD2             OPTIONAL I AND J DIMENSIONS AND A REQUIRED TYPE
450 1 CD2************
451 1 CD3             OUTPUTS TO THE ANA:
452 1 CD3
453 1 CD3             SET OF DATA ELEMENT(S) REQUESTED
454 1 CD3************
455 1 CD4             INTERNAL VARIABLES:
456 1 CD4
457 1 CD4             INDBUF - INTERFACE TABLE HEADER
458 1 CD4             ISLENG - LENGTH OF SYMBOLIC STRING
459 1 CD4             ITOKEN - POSITION WITHIN THE SYMBOLIC STRING
460 1 CD4             NNAME - BUFFER FOR VALID NAMES TO BE ALLOCATED
461 1 CD4             NBPUFF - BUFFER AREA FOR XPCT AND XPRE USE
462 1 CD4             NBENT - NUMBER OF ENTRIES IN THIS ANA REQUEST
463 1 CD4             NNAME - NUMBER OF NAMES IN SYMBOLIC STRING
464 1 CD4             NEXTHM - TOKEN POSITION FOR NEXT NAME
465 1 CD4             STRING - SYMBOLIC STRING INPUT TO DEFINE
466 1 CD4************
467 1 CD5             EXTERNAL ROUTINES USED:
468 1 CD5
469 1 CD5             EXEC, IAMC, KCVT, XMPAR, XPSET,
470 1 CD5             XPGEN, XPFIT, XUONG, XIDFT, XIMS6
471 1 CD5************
472 1 CD5************
483 1 BEGIN DEFIN
484 2 CALL RPMAR TO GET LU AND DEBUG FLAGS
485 3 CALL APGET TO RETRIEVE SYMBOLIC STRING
486 4 GET STRING LENGTH FROM INTERFACE TABLE HEADER
487 5 SEARCH WHILE TOKEN-POSITION .LT. STRING-LENGTH, OR
488 6 WHILE CURRENT-DELEM .NE. END-OF-STRING
489 7 PERFORM ZIDPM TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
490 8 PERFORM ZIDPM TO PROCESS THE CURRENT NAME
491 9 EXIT IF THERE WAS AN ERROR RETURN FROM ZIDPM
492 10 CALL ZINSK TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
493 11 SET PROCESSOR RETURN CODE TO ABEND
494 12 ORELSE INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
495 13 ENDLOOP
496 14 SET PROCESSOR RETURN FOR NORMAL EXIT
497 15 ENDS
498 16 END SEARCH
499 17 SET OPTION SO XPRER WILL DO A QUEUE REQUEST
500 18 DO FOR NUMBER OF REQUESTS IN REQUEST BUFFER (MAXARR)
501 19 IF THIS IS LAST REQUEST, THEN
502 20 SET OPTION TO CLOSE REQUEST BUFFER
503 21 ENDF
504 22 COMPUTE INDEX TO THIS REQUEST
505 23 CALL XPRER TO QUEUE THIS REQUEST
506 24 ENDDO
507 25 CALL XPRIT TO RETURN TO THE MANAGER
508 26 1 END DEFIN
527 1 BEGIN XIDPM
528 2 SET IDIM AND JOIM TO 1
529 3 IF TOKEN IS NOT A NAME THEN
530 4 SET ERROR CODE
531 ELSE
532 5 MOVE NAME INTO REQUEST
533 6 INCREMENT TO NEXT TOKEN
534 7 IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREM) THEN
535 8 INCREMENT TO NEXT TOKEN
536 9 IF TOKEN IS NOT AN INTEGER OR
537 10 TOKEN IS NOT ZERO THEN
538 11 CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
539 12 EXIT TO :PNERR1:
540 13 ENDF
541 14 SET IDIM TO THIS TOKEN
542 15 INCREMENT TO NEXT TOKEN
543 16 IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
544 17 INCREMENT TO NEXT TOKEN
545 18 IF TOKEN IS NOT AN INTEGER OR
546 19 TOKEN IS NOT ZERO THEN
547 20 CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
548 21 EXIT TO :PNERR1:
549 22 ENDF
550 23 INCREMENT TO NEXT TOKEN
551 24 IF TOKEN IS NOT A RIGHT PAREM THEN
552 25 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
553 26 EXIT TO :PNERR1:
554 27 ENDF
555 28 INCREMENT TO NEXT TOKEN
556 29 IF TOKEN IS NOT A NAME THEN
557 30 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
558 31 EXIT TO :PNERR1:
559 32 ENDF
560 33 INCREMENT TO NEXT TOKEN
561 34 STARTSEARCH FOR ALL VALID TYPES
562 35 EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
563 36 SET TYPE AND CLASS IN REQUEST
564 37 COMPUTE SIZE AS IDIM * JOIM * LENGTH OF TYPE
565 38 IF SIZE IS TOO LARGE (>1200 WORDS) THEN
566 39 CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
567 40 EXIT TO :PNERR1:
568 41 ENDF
569 42 ENDOLOOP
570 43 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
571 44 EXIT TO :PNERR1:
572 45 ENDOLOOP
573 46 EXIT TO :PNERR1:
574 47 ENDOSEARCH
575 48 INCREMENT TO NEXT TOKEN
576 49 IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
577 50 NEXT TOKEN IS NOT A COMMA THEN
578 51 CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
BUILD REQUEST TO DELETE/VERIFY ABSENT THIS DATA ELEMENT
BUILD REQUEST TO ALLOCATE THIS DATA ELEMENT
EXIT XZOPN
:PMEPRI:
CALL XIMSG TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
EXIT XZOPN
ENSDC IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

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

IT HAS NO INTERFACE TABLE.

INPUTS FROM THE MANAGER:
- LU - LOGICAL UNIT OF THE USER'S TERMINAL
- UID - USER ID CODE
- FLAGS - FLAGS FOR DEBUG

INPUTS FROM AWA:
- &SCNTB - SCAN CONTROL TABLE CREATED BY SCAN

OUTPUTS TO MANAGER:
- XZRET - RETURN CODE TO MANAGER

OUTPUTS TO AWA:
- XSCAN - NEW VALUE FOR X SCAN VARIABLE
- YSCAN - NEW VALUE FOR Y SCAN VARIABLE

RTE ROUTINES USED:
- CLOSE, EXEC, FLOAT, KCVT, POINT,
- READ, RMPAR, WRIT

FDS ROUTINES USED:
- XPXIT, XRXT, XRMV, XDBG, XVPAM,
- XIFCL, XIMSG
BEGIN ENDS
CALL RMPAR TO GET INPUTS FROM MANAGER
CALL XVPAM TO REQUEST AWA MANAGEMENT
ERREXIT IF &SCNTB NOT FOUND TO :ERR4:
CALL EXEC TO READ IN &SCNTB
SAVE REQUEST TO RETRIEVE SUMTAB IN REGBUF
CALL XVPAM TO REQUEST AWA MANAGEMENT
ERREXIT IF NOT FOUND TO :ERR4:
CALL EXEC TO READ IN SUMTAB
IF SUMMARY TABLE IS LARGER THAN 32 ENTRIES THEN
SET SIZE OF SUMTAB TO 32 ENTRIES
ENDIF
CALL WRITF TO WRITE SUMTAB TO DATBOX
ERREXIT IF WRITF ERROR TO :ERR4
IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
CALL READ TO READ HEADER RECORD
ERREXIT IF READF ERROR TO :ERR4.
UPDATE NUMBER OF SUMMARY TABLE ENTRIES
CALL WRITF TO WRITE UPDATED HEADER
ERREXIT IF WRITF ERROR TO :ERR4:
CALL CLOSE TO CLOSE DATBOX
SAVE REQUEST TO DE/LIVER ABS &SCNTB IN REGBUF
IF THERE ARE REMAINING SCANS IN &SCNTB THEN
CALL EXEC TO WRITE REMAINING &SCNTB
SAVE REQUEST TO_ALLOC AND STORE VALUES FOR NEW &SCNTB
ENDIF
SET RETURN PARAMETER TO NORMAL RETURN
ELSE
PERFORM SETXY
ENDIF
CALL XVPAM TO REQUEST AWA MANAGEMENT
CALL XPIIT TO TERMINATE WITH RETURN PARAMETERS
EXIT ENDS
BEGIN SETXY
IF XCUR IS END STEP THEN
   IF THERE IS 1 VARIABLE THEN
      SET X TO CENTROID
      CALL POSNT TO POSITION FILE TO CENTROID RECORD
      ERREXIT IF POSNT ERROR TO :ERR4:
   ELSE
      IF YCUR IS END STEP THEN
         SET XCUR TO ZERO
         SET X TO CENTROID
         SET YCUR TO ZERO
         SET Y TO CENTROID
         CALL POSNT TO POSITION TO CENTROID RECORD
         ERREXIT IF POSNT ERROR TO :ERR4:
      ELSE
         SET XCUR TO (-XSTEP)
         INCREMENT YCUR BY 1
         IF XSTEP IS ZERO AND Y IS CENTROID THEN
            INCREMENT YCUR BY 1
            CALL WRITF TO WRITE DUMMY AS CENTROID RECORD
            ERREXIT IF WRITE ERROR TO :ERR4:
         ENDIF
         COMPUTE X AS (XCENT + XINCR + FLOAT (XCUR))
         COMPUTE Y AS (YCENT + YINCR + FLOAT (YCUR))
      ENDIF
   ENDIF
ELSE
   INCREMENT XCUR BY 1
   IF THERE IS 1 VARIABLE AND X IS THE CENTROID OR
   THERE ARE 2 VARIABLES AND X IS THE CENTROID AND Y IS THE CENTROID THEN
   INCREMENT XCUR BY 1
   CALL WRITF TO WRITE DUMMY AS CENTROID RECORD
   ERREXIT IF WRITE ERROR TO :ERR4:
   ENDIF
   COMPUTE X AS (XCENT + XINCR + FLOAT (XCUR))
   IF THERE ARE 2 VARIABLES THEN
      COMPUTE Y AS (YCENT + YINCR + FLOAT (YCUR))
   ENDIF
ENDIF
END IF # SCAN VARIABLES
CALL EXEC TO WRITE VARIABLE
SAVE REQUEST TO STORE VALUES FOR VARIABLE IN RERBUF
ENDDEF
CALL EXEC TO WRITE $SCNTB
SAVE REQUEST TO STORE NEW $SCNTB
SET RETURN PARAMETERS TO RESET SEQUENCE NUMBER
END SETXY
**CD**********

**CD**

SCAN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

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

**CD**

SCAN PROCESSOR EXECUTES A SERIES OF PROCESSES ITERATING ON XSCAN AND 

**CD**

TSCAN (IF ENTERED) VALUES COMPUTED USING THE CENTROID, THE INCREMENT, 

**CD**

AND THE CURRENT STEP NUMBER, CREATING A DATA BOX FILE.

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

**CD**

INPUTS FROM MANAGER:

**CD**

LU - LOGICAL UNIT OF THE USER'S TERMINAL

**CD**

USERID - USER ID CODE

**CD**

FLAGS - FLAGS FOR DEBUG

**CD**

ENTSUB-DISPLACEMENT OF THIS SCAN ENTRY IN BSEGTH

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

**CD**

INPUTS FROM INTERFACE TABLE:

**CD**

PROCSCARTRIDGE # FOR DATA BOX FILE

**CD**

SUMTAB-SUMMARY TABLE

**CD**

*DATABOX-NAME OF DATA BOX ENTERED

**CD**

NOVAR - NUMBER OF SCAN VARIABLES

**CD**

*SCAN - NAME OF X SCAN VARIABLE

**CD**

UNIT - UNIT OF X VARIABLE

**CD**

XCENTR-CENTROID OF X VARIABLE

**CD**

ZINC - INCREMENT FOR X

**CD**

XSIES - NUMBER OF STEPS FOR X

**CD**

*TSCAN - NAME OF Y SCAN VARIABLE

**CD**

UNIT - UNIT OF Y VARIABLE

**CD**

TCENTR-CENTROID OF Y VARIABLE

**CD**

TINC - INCREMENT FOR Y

**CD**

YSTEPS - NUMBER OF STEPS FOR Y

**CD**

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

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

**CD**

OUTPUTS TO MANAGER

**CD**

XZRET - RETURN CODE TO MANAGER

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

**CD**

OUTPUTS TO WORK AREA (CAMA):

**CD**

DATABOX-FILE WHERE SUMMARY TABLE IS WRITTEN

**CD**

XSCAN - X SCAN VARIABLE

**CD**

TSCAN - Y SCAN VARIABLE

**CD**

BSCNTB - SCAN CONTROL TABLE (173 WORDS PER ACTIVE SCAN)

**CD**

(1) SUMTAB & CHAR NAME OF SUMMARY TABLE

**CD**

(2) DISPL DisPLACEmenT FOR SUMMARY TABLE

**CD**

(3) DATABOX & CHAR QUALIFIED NAME OF DATA BOX FILE

**CD**

(4) RESET - RESET SEQUENCE NUMBER

**CD**

(5) CENTRE CENTROID RECORD NUMBER

**CD**

(6) XSCAN NAME OF X VARIABLE

**CD**

(7) XDISPL DisPLACEmenT FOR X

**CD**

(8) XCEXT CENTROID FOR X

**CD**

(9) XZINC INCREMENT FOR X

**CD**

(10) XSCAN NAME OF X VARIABLE

**CD**

(11) XDISPL DisPLACEmenT FOR X

**CD**

(12) XCEXT CENTROID FOR X

**CD**

(13) XZINC INCREMENT FOR X
BEGIN SCAN
CALL RMAP TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XPSET TO GET PROCON AND # SCAN VARIABLES
EREREX IF # SCAN VARIABLES < 1 OR > 2 TO :ERR3;
GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
EREREX IF SMTAB IS A LITERAL TO :ERR3;
EREREX IF DISPLACEMENT IS NOT AN ELEMENT BOUNDARY TO :ERR3;
GET DATA BOX NAME FROM INTERFACE TABLE
DO FOR # SCAN VARIABLES
CALL XPSET TO GET NAME AND DISPLACEMENT
COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND DOTH
CALL XPGET TO GET UNITS, CENTROID, INCR, # STEPS
EREREX IF STEPS < 0 OR > 5 TO :ERR3;
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR $EVENT AND $SCNTB
CALL XPAM TO REQUEST ANA MANAGEMENT
CALL EXEC TO READ IN $EVENT
IF $SCNTB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF $SCNTB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN $SCNTB
EREREX IF # SCANS = MAXIMUM ALLOWED (4) TO :ERR3;
EREREX IF THIS DATBOX NAME IS ALREADY IN USE TO :ERR3;
ENDIF
EREREX IF THIS IS THE LAST ENTRY IN $EVENT TO :ERR3;
GET THE SEQUENCE NUMBER OF THIS SCAN FROM $EVENT
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH $EVENT FOR THE SEQUENCE NUMBER
EREREX IF THIS SCAN IS THE LAST ENTRY IN $EVENT TO :ERR3;
IF THIS IS A SEMI OVERLAP (2 PROCESSORS NAMES NOT EQUAL) THEN
SET RESET NUMBER TO THIS ENTRY SEQUENCE NUMBER
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
COMPUTE SIZE OF DATBOX FILE = (2+((2 * XSTEPS + 1)+(2 * YSTEPS + 1)+1)) / 2
COMPUTE CENTROID RECORD NUMBER = SIZE + 3
DO FOR # SCAN VARIABLES
COMPUTE BEGINNING VALUE = (CENT + INCR * FLOAT (CUR STEP))
ENDDO
CALL XPPUT TO STORE DATBOX AND SCAN VARIABLES
CALL CREAT TO CREATE DATBOX FILE
IF FILE ALREADY EXISTS THEN
CALL PURGE TO PURGE FILE
EREREX IF PURGE ERROR TO :ERR2;
CALL CREAT TO CREATE FILE
ENDIF
EREREX IF CREAT ERROR TO :ERR2;
CALL WRIT TO WRITE HEADER RECORD TO DATBOX
EREREX IF WRIT ERROR TO :ERR2;
CALL POSMT TO POSITION FIELD TO FIRST DATA RECORD
EREREX IF POSMT ERROR TO :ERR2;
CALL EXEC TO WRITE $SCNTB
SAVE REQUEST TO DELETE/VERIFY ABSENT $SCNTB IN REQBUF
SAVE REQUESTS TO ALLOC AND STORE VALUES FOR NEW $SCNTB
CALL XPAM TO REQUEST ANA MANAGEMENT
873 2 EXETYPE IF NO AREA SPACE TO ERR1:
874      CALL EXIT TO EXIT NORMALLY
875      IF THERE ARE MORE ACTIVE SCANS (# SCAMS > 0) THEN
876      CALL EXEC TO READ NEW SCANS
877      SAVE REQUEST TO ALL SCANS STORE VALUES FOR ORIGINAL SCANS
879      ENDIF
882     ERR2: CALL CLOSE TO CLOSE DATABASE
883     CALL CLOSE TO CLOSE DATABASE FOR XINPUT
884     SET VALUE FOR XINPUT
886     SAVE REQUEST TO REQUEST VIA MANAGEMENT
888     CALL CLOSE TO DISPLAY ERROR
890     END SCAN TO ABEND SCAN
891                  5-346
893 1 CD0  FORTRAN CALLING PROCEDURE:
894 1 CD0
895 1 CD0
896 1 CD0
897 1 CD0
898 1 CD0
899 1 CD1  XCHR IS USED BY THE ASSGN ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS
900 1 CD1  FOR CHARACTER-TYPE OBJECT DATA ELEMENTS
901 1 CD1
902 1 CD1
903 1 CD0
904 1 CD2
905 1 CD2  INPUTS FROM ASSCOM
906 1 CD2
907 1 CD2  LU, SYMTAB, ESTRNG, DATYPS, RSLTPT, CLSTRT, MAPMDS, RESULT
908 1 CD2
909 1 CD0
910 1 CD3
911 1 CD3  OUTPUTS TO ASSCOM
912 1 CD3
913 1 CD3  RESIT, NUMMD, RSLTPT, OFRMD, OFRMD
914 1 CD3
915 1 CD0
916 1 CD5
917 1 CD5  EXTERNAL REFERENCES
918 1 CD5
919 1 CD5  FDS - XPREG, XPAR, XMMOV, XILSS, XIMSG
920 1 CD5
921 1 CD5
922 1 CD5  RTE - IANB
923 1 CD0
1 BEGIN XICHR
2 SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
3 CASE RESULT DATA TYPE:
5 CASE:
6 :CHRSTR:
7 DETERMINE # WORDS IN CHARACTER STRING
8 MOVE CHARACTER STRING TO RESULT LOCATION
9 IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
10 BLANK FILL AFTER CHARACTER STRING
11 ENDIF
12 :CHRRE:
13 IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
14 POP DISPLACEMENT FROM RESULT STACK
15 ELSE
16 SET DISPLACEMENT = 0
17 ENDIF
18 :CHRFS:
19 POP RESULT OPERAND FROM STACK
20 IF RESULT OPERAND IS FREE, THEN
21 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
22 ELSE CHARACTER = CHARACTER
23 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
24 ENDIF
25 :CHRFS:
26 CALL XPERG TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
27 IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
28 BLANK FILL REMAINING DATA
29 ENDIF
30 END CASE
31 EXIT XICHR
32 :FIXERR:
33 SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO"
34 "NUMERICAL DATA"
35 CALL XINC5 TO OUTPUT MESSAGE TO USER
36 CALL XLIST5 TO LIST SYMBOLIC STRING
37 CALL XEXIT TO EXIT PROCESSOR
38 EXIT XICHR
XZDIM - DDBSP INPUT PROCESSOR

XZDIM IS CALLED TO INTERPRET THE VARIOUS INTERFACE TABLE INPUTS
(MOSTLY SYMBOLIC STRINGS) AND BUILD DATA ARRAYS FROM THEM.

ALL INPUT COMES FROM THE 26 INTERFACE TABLE ARGUMENTS

OUTPUT

COMMON

MCVARL, MCRELL, CVVALUE, MDCVARL, MDCVRUL, NC

NOTES

USES ROUTINES

EXEC

XPUG

XPMOV

XZLIS

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0

CD0
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'.

INPUTS FROM CALLING SEQUENCE:

ARRAY - SYMBOIC STRING TO BE SEARCHED
END    - ENDING INDEX INTO 'ARRAY'
START  - BEGINNING INDEX INTO 'ARRAY'
TOKEN  - TOKEN TO BE SEARCHED FOR

OUTPUTS TO CALLING SEQUENCE:

INDEX - POSITION OF 'TOKEN' IN SYMBOIC STRING IF FOUND,
        OR 'END' IF NOT FOUND
1083 1 BEGIN XZDFT
1084 2 SET INDEX TO START
1085 2 DO UNTIL INDEX .GE. END
1086 3 IF ARRAY(INDEX) .EQ. TOKEN, THEN
1087 4 SET END TO INDEX
1088 3 ELSE
1089 4 CASE ARRAY(INDEX)+1 ( XZDFT
1090 5 :ADD1:. 0 END OF STRING
1091 5 :ADD2:. 1 INTEGER
1092 5 :ADD3:. 2 REAL
1093 5 :ADD4:. 3 DOUBLE
1094 5 :ADD1:. 4 UNKNOWN
1095 5 :ADD4:. 5 NAME
1096 5 :ADD1:. 6 UNKNOWN
1097 5 :ADD1:. 7 UNKNOWN
1098 5 :CALST:. 8 CHARACTER STRING
1099 5 :ADD1:. 9 UNKNOWN
1100 5 :ADD1:. 10 +
1101 5 :ADD1:. 11 -
1102 5 :ADD1:. 12 *
1103 5 :ADD1:. 13 /
1104 5 :ADD1:. 14 <
1105 5 :ADD1:. 15 >
1106 5 :ADD1:. 16 #
1107 5 :ADD1:. 17 @
1108 5 :ADD1:. 18 =
1109 5 :ADD1:. 19 %
1110 5 :ADD1:. 20 ?
1111 5 :ADD1:. 21 !
1112 5 :ADD1:. 22 )
1113 5 :ADD2:. 23 START OF SYMBOLIC STRING
1114 5 :ADD1:. 24 END OF STRING
1115 5 :ADD1:. 25 %
XIDP1 - DATA BOX DISPLAY OVERLAY - PREPARES DATA FOR DISPLAY
SCHEDULED BY DBDSP

XIDP1 READS IN THE SCAN CONTROL INFORMATION AND THE SUMMARY
TABLES PRODUCED BY SCAN/ENDSCAN FOR FURTHER PROCESSING.
IN ADDITION, XIDP1 VERIFIES THE DISPLAY DEPENDANT VARIABLE
NAMES AND THE CONSTRAINT VARIABLE NAMES ARE CONTAINED IN THE
DEPENDANT VARIABLE NAME LIST GENERATED DURING THE SCAN.
XIDP1 ALSO GENERATES CONSTRAINT MASKS WITH SUBROUTINE XIDMK
FOR SUBSEQUENT DISPLAY BY XIDP2 AND XIDP3.

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) 3 ASCII WORDS OF BLANKS

RECORD 2
(6) X SCAN VARIABLE (6 CHAR)
(7) X SECOND SUBSCRIPT (INT OR ZERO)
(8) X UNITS (6 CHAR)
(9) X CENTROID (REAL)
(10) X INCREMENT (REAL)
(11) Y NUMBER OF STEPS (INTEGER 1-5)
(12) Y SCAN VARIABLE (6 CHAR)
(13) Y SECOND SUBSCRIPT (INT OR ZERO)
(14) Y UNITS (6 CHAR)
(15) Y CENTROID (REAL)
(16) Y INCREMENT (REAL)
(17) Y NUMBER OF STEPS (INTEGER 1-5)

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

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
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

DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DDOSP

NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN

NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)

LIST OF CONSTRAINT RELATIONS INPUT BY USER

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

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

NUMBER OF DEP DISP VAR IN NDTARV LIST (INTEGER)

NUMBER OF DEP DISP VARIABLE PPAIR *CALC FACTORS FOR O/P

SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/MOT=0

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

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

X VAR VALUES FOR X COORDINATES (1 - 11 REAL)

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

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

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

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

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

TABLE IN COMMON FOR SMTAB VARIABLE NAMES AND UNITS

UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN

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

COMMUNICATION BUFFER FOR RMAP = LU, USER ID, FLAG

LOGICAL UNIT # FOR XPROM CALLING SEQUENCE = USER LOCATIN

DBDSP = DDOSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU

PROMPT = TABLE IN COMMON TO COMMUNICATE WITH XPROM

SELECT = SELECT = 0 PROMPT ; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
**Input 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 dependent 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.

---

**XZDP2**

- Data box display overlay - prompts user for display desired, then formats and displays data accordingly.
- Scheduled by DDSP.

---

**Functionality**

- XZDP2 issues prompts to the user requesting page number, starting row number, number of rows, and desired Lu. 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.
CD**

1592  CD0     FORTRAN CALLING PROCEDURE
1594  CD0     CALL XZ1SP (STRING, LEN)
1596  CD1     ********
1597  CD1     XZ1SP REMOVED DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1598  CD1     A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1600  CD1     WITH BLANKS
1601  CD1     ********
1602  CD2     INPUT
1604  CD2     ********
1605  CD2     CALLING SEQUENCE
1607  CD2     STRING - INPUT CHARACTER STRING
1608  CD2     LEN - NUMBER OF WORDS IN STRING
1609  CD2     ********
1610  CD2     OUTPUT
1611  CD3     ********
1613  CD3     CALLING SEQUENCE
1615  CD3     STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
1616  CD3     BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILLED
1617  CD3     LEN - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILLED
1618  CD3     ********
1620  CD3     NOTES
1621  CD4     ********
1622  CD4     USES Routines
1624  CD4     XR1SP
1626  CD4     ********
1628  CD4     BEGIN XZ1SP
1630  CD4     CALL XR1SP TO REMOVE DUPLICATE BLANKS FROM STRING
1632  CD4     DO WHILE THERE ARE TRAILING WORDS IN STRING
1633  CD4     SET THIS TRAILING WORD TO BLANKS
1634  CD4     ENDDO
1635  CD4     END XZ1SP
**FORTRAN CALLING SEQUENCE:**

- **CALL XIIFCL (LU)**

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

**INPUTS FROM CALLING SEQUENCE:**

- **LU - LOGICAL UNIT OF TERMINAL BEING USED.**

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

**BEGIN XIIFCL**

- **CALL .ENTR TO RESOLVE PARAMETER ADDRESS**
- **STARTSEARCH FOR ALL XVSTB ENTRIES**
- **EXIT IF THIS ENTRY'S LU IS MINE**
- **ENDSEARCH**
- **GET CLASS NUMBER FROM XVSTB**
- **SAVE CLASS NUMBER IN XPCLS**

**END XIIFCL**
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1674</td>
<td>C00</td>
<td>FORTRAN CALLING PROCEDURE:</td>
</tr>
<tr>
<td>1675</td>
<td>C00</td>
<td>CALL XIFNC(ENTRY)</td>
</tr>
<tr>
<td>1676</td>
<td>C00</td>
<td>CALL XIFNC(ENTRY)</td>
</tr>
<tr>
<td>1677</td>
<td>C00</td>
<td>CALL XIFNC(ENTRY)</td>
</tr>
<tr>
<td>1678</td>
<td>C00</td>
<td>CALL XIFNC(ENTRY)</td>
</tr>
<tr>
<td>1679</td>
<td>C00</td>
<td>CALL XIFNC(ENTRY)</td>
</tr>
<tr>
<td>1680</td>
<td>C01</td>
<td>XIFNC IS USED BY THE ASSGN ROUTINE XZPS2 TO EVALUATE FUNCTION</td>
</tr>
<tr>
<td>1681</td>
<td>C01</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1682</td>
<td>C01</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1683</td>
<td>C01</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1684</td>
<td>C01</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1685</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1686</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1687</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1688</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1689</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1690</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1691</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1692</td>
<td>C02</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1693</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1694</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1695</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1696</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1697</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1698</td>
<td>C03</td>
<td>OPERATIONS</td>
</tr>
<tr>
<td>1699</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1700</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1701</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1702</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1703</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1704</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1705</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1706</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1707</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1708</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1709</td>
<td>C05</td>
<td>EXTERNAL REFERENCES</td>
</tr>
</tbody>
</table>
1711 1 BEGIN XIFNC
1712 2 PERFORM FUNCTION INDICATED BY ENTRY
1713 2 EREXIT TO :OVER: IF OVERFLOW OR UNDERFLOW IS INDICATED
1714 2 PUSH RESULT AND TYPE ONTO RESULT STACK
1715 1 EXIT XIFNC
1716 2 :OVER:
1717 2 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
1718 2 CALL XIMSG TO OUTPUT MESSAGE TO USER
1719 2 CALL XLIST TO LIST SYMBOLIC STRING
1720 2 CALL XPRT TO EXIT PROCESSOR
1721 1 END XIFNC
FORTRAN CALLING PROCEDURE:

CALL XIIFRE

XIIFRE IS USED BY THE ASGN ROUTINE XIIPS2 TO PROCESS DATA ASSIGNMENTS
FOR FREE-TYPE OBJECT DATA ELEMENTS

INPUTS FROM ASGCOM

SYMTAB, SSTRNG, RESULT, RSLTPF, LSCTAM, HAPWDS

OUTPUTS TO ASGCOM

CPRNDS, RERST, NI'MHDS, RSLTPF

EXTERNAL REFERENCES

FD5 - XPREQ, XRMO, XIPCS

RTC - IAMD
1755   1 BEGIN XZFRE
1756   2 * CASE (RESULT DATA TYPE ) :FRESTR:, :FREFRE:, :FREFIX:, :FREFX:
1757   3 1
1758   4 * :FRESTR:
1759   5 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1760   6 MOVE CHARACTER STRING TO RESULT LOCATION
1761   7 :FREFRE:
1762   8 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1763   9 POP DISPLACEMENT FROM RESULT STACK
1764  10 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPED
1765  11 SET DISPLACEMENT = 0
1766  12 ENDIF
1767  13 :FREFIX:
1768  14 POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1769  15 GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
1770  16 IF RESULT OPERAND IS FREE, THEN
1771  17 CALL XPRG: TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1772  18 ELSE FREE = CHARACTER DATA ELEMENT
1773  19 CALL XPREQ TO RETRIEVE LOGICAL ELEMENT OF CHARACTER DATA FROM RESULT OPERAND
1774  20 AT DISPLACEMENT DETERMINED
1775  21 SET # WORDS TO BE STORED : IN OBJECT = # WORDS/ELEMENT FOR RESULT OPERAND
1776  22 ENDIF
1777  23 :FREFX:
1778  24 CALL XZPCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1779  25 SET # WORDS TO BE STORED = RESULT DATA TYPE
1780  26 END CASE
1781  27 END XZFRE
1785    1 CDO  FORTRAN CALLING PROCEDURE:
1786    1 CDO
1787    1 CDO
1788    1 CDO  CALL XIFXD
1789    1 CDO
1790    1 CDO************
1791    1 CDO
1792    1 CDO  XIFXD IS USED BY THE ASGW ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS
1793    1 CDO  FOR FIXED-TYPE OBJECT DATA ELEMENTS
1794    1 CDO
1795    1 CDO************
1796    1 CDO
1797    1 CDO  INPUTS FROM ASGCOM
1798    1 CDO
1799    1 CDO  LU,SINTAB,SSTRTG,RESULT,RSLTP,DATYPES,CLSTAN
1800    1 CDO
1801    1 CDO************
1802    1 CDO
1803    1 CDO  OUTPUTS TO ASGCOM
1804    1 CDO
1805    1 CDO  RSLTP,REGST,NUMWDS,OPRnds
1806    1 CDO
1807    1 CDO************
1808    1 CDO
1809    1 CDO  FD$ - XPREG,XPXIT,XRMOV,XZLSS,XZMSG,XZPCS
1810    1 CDO
1811    1 CDO
1812    1 CDO  RTE - IAND
1813    1 CDO************
1815 1 BEGIN XIFXD
1816 2   SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1818 4   SET DISPLACEMENT = 0
1819 3   # WORDS RETRIEVED = OBJECT DATA TYPE
1820 1821 3   ERREXIT TO :NOCHAR: IF TYPE IN SYMBOL TABLE FOR RESULT OPERAND IS NOT FREE
1822 3   IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1823 4   POP DISPLACEMENT FROM RESULT STACK
1824 3   ELSE FREE ELEMENT HAS NOT BEEN SUBSCRIPTED
1825 4   SET DISPLACEMENT = 0
1826 3   ENDF
1827 3   POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1828 3   CALL XPREQ TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1829 3   CALL XPPCS TO POP RESULT VALUE, CONVERT IF NECESSARY, AND SET UP FOR STORE
1830 3   ENDCASE
1831 3   EXIT XIFXD
1832 1833 2   :NOCHAR:
1834 3   SET MESSAGE TO BE OUTPUT TO "NUMERICAL DATA ELEMENT CANNOT BE SET EQUAL TO CHARACTER DATA"
1835 3   CALL XMSSG TO OUTPUT MESSAGE TO USER
1836 3   CALL XILSS TO LIST SYMBOLIC STRING
1837 3   CALL XPXIT TO EXIT PROCESSOR
1838 1842 1 END XIFXD
1844 1 CDO FORTRAN CALLING PROCEDURE
1845 1 CDO
1846 1 CDO CALL XILSS (LU, STRING, INDEX)
1847 1 CDO
1848 1 CDO
1849 1 C*****
1850 1 C1
1851 1 C1 XILSS IS CALLED TO LIST A SYMBOLIC STRING AND AN INDICATOR TO A
1852 1 C1 PARTICULAR TOKEN IN THAT STRING
1853 1 C1
1854 1 C*****
1855 1 C2
1856 1 C2 INPUT
1857 1 C2
1858 1 C2 LU - LOGICAL UNIT NO. FOR OUTPUT OF STRING
1859 1 C2
1860 1 C2 STRING - SYMBOLIC STRING TO BE LISTED
1861 1 C2
1862 1 C*****
1863 1 C3
1864 1 C3 OUTPUT
1865 1 C3
1866 1 C3 THE SYMBOLIC STRING IS OUTPUT TO THE LU FOLLOWED BY A LINE CONTAINING
1867 1 C3 AN INDICATOR (UP ARROW) TO THE DESIGNATED TOKEK.
1868 1 C3
1869 1 C*****
<table>
<thead>
<tr>
<th>CD4</th>
<th>Internal Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INTERNAL VARIABLES</td>
</tr>
<tr>
<td></td>
<td>CONTROL = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING</td>
</tr>
<tr>
<td></td>
<td>FOR EACH OF THE TOKEN VALUES 1-32. EACH CONTROL TABLE</td>
</tr>
<tr>
<td></td>
<td>ENTRY IS 3 WORDS:</td>
</tr>
<tr>
<td></td>
<td>WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER</td>
</tr>
<tr>
<td></td>
<td>WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER</td>
</tr>
<tr>
<td></td>
<td>OR FLAG DESCRIBING HOW TO COMPUTE</td>
</tr>
<tr>
<td></td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>WORD 3 (TOKSZ) = NO. OF WORDS IN SYMBOLIC STRINGS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CD4</th>
<th>Entry</th>
<th>Token</th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD4</td>
<td>1</td>
<td>INTEGER</td>
<td>3</td>
<td>-1&gt;&gt;CALL XRI6</td>
<td>2</td>
</tr>
<tr>
<td>CD4</td>
<td>2</td>
<td>REAL</td>
<td>7</td>
<td>-2&gt;&gt;CALL XRI6</td>
<td>3</td>
</tr>
<tr>
<td>CD4</td>
<td>3</td>
<td>DOUBLE</td>
<td>9</td>
<td>-3&gt;&gt;CALL XRI6</td>
<td>4</td>
</tr>
<tr>
<td>CD4</td>
<td>4</td>
<td>NAME</td>
<td>3</td>
<td>-4&gt;&gt;USE 3 WORDS</td>
<td>4</td>
</tr>
<tr>
<td>CD4</td>
<td>5</td>
<td>FOLLOWING TOKEN</td>
<td>0</td>
<td>Q&gt;&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>CD4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>Q&gt;&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>CD4</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>Q&gt;&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>CD4</td>
<td>8</td>
<td>CHAR. STR.</td>
<td>-1&gt;&gt;USE VALUE</td>
<td>-5&gt;&gt;USE SIZE WORDS</td>
<td>-1&gt;&gt;USE SIZE+2 WORDS</td>
</tr>
<tr>
<td>CD4</td>
<td>9</td>
<td>TOKEN</td>
<td>TOOKEN</td>
<td>0</td>
<td>Q&gt;&gt;ERROR (INVALID)</td>
</tr>
<tr>
<td>CD4</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>19</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>23</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>24</td>
<td>END SYM. STR.</td>
<td>0</td>
<td>7&gt;&gt;END SYM. STR.</td>
<td>0</td>
</tr>
<tr>
<td>CD4</td>
<td>25</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>26</td>
<td>BACKSLASH</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>27</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>28</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>29</td>
<td>LEFT BRACKET</td>
<td>1</td>
<td>LEFT BRACKET</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>30</td>
<td>RIGHT BRACKET</td>
<td>1</td>
<td>RIGHT BRACKET</td>
<td>1</td>
</tr>
<tr>
<td>CD4</td>
<td>31</td>
<td>REPEAT</td>
<td>4</td>
<td>-6&gt;&gt;CALL XRI6 AND</td>
<td>2</td>
</tr>
<tr>
<td>CD4</td>
<td>32</td>
<td>,</td>
<td>1</td>
<td>APPEND &quot;A&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>
**Lines 1-60**

```
1  C*******
2  CD5     
3  CD5     
4  CD5     
5  CD5     
6  CD5     
7  CD5     
8  CD5     
9  CD5     
10 CD5    
11 CD5    
12 CD5    
13 CD5    
14 CD5    
15 CD5    
16 CD5    
17 CD5    
18 CD5    
19 CD5    
20 CD5    
21 CD5    
22 CD5    
23 CD5    
24 CD5    
25 CD5    
26 CD5    
27 CD5    
28 CD5    
29 CD5    
30 CD5    
31 CD5    
32 CD5    
33 CD5    
34 CD5    
35 CD5    
36 CD5    
37 CD5    
38 CD5    
39 CD5    
40 CD5    
41 CD5    
42 CD5    
43 CD5    
44 CD5    
45 CD5    
46 CD5    
47 CD5    
48 CD5    
49 CD5    
50 CD5    
51 CD5    
52 CD5    
53 CD5    
54 CD5    
55 CD5    
56 CD5    
57 CD5    
58 CD5    
59 CD5    
60 CD5    
```

**Lines 1-20**

```
1  BEGIN XILSS
```

**Lines 21-60**

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

1. **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, XUDBR WILL BE CALLED AFTER THE MESSAGE IS OUTPUT.

**NUMBER** - INTEGER MESSAGE NUMBER OF THE FORM 'AMN' WHERE A - AREA INDICATOR AS FOLLOWS

- 1 - AS
- 2 - XB
- 3 - XG
- 4 - XI
- 5 - XS
- 6 - XT
- 7 - XX
- 8 - XL
- 9 - DF
- 10 - SC

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

**LOCATE** - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEDE '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

**NOTES**

- USES FOS SYSTEM MESSAGE FILE JXMSG
- USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XROV, XUDBR
BEGIN XZMSG
SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
SET NUMBER IN PREFIX
READ MESSAGE DIRECTORY RECORD
IF AREA VALID
THEN
SET AREA CODE IN PREFIX
IF MESSAGE NUMBER > 0
THEN
IF VALID MESSAGE NUMBER
THEN
COMPUTE MESSAGE RECORD NUMBER
READ RECORD
CALL XRMV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
ELSE
EXIT TO :ERROR:
ENDIF
ENDIF
CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
IF MESSAGE NUMBER > 0
THEN
CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
ENDIF
ELSE
SET AREA IN PREFIX
ERROR: CALL XRMV TO MOVE 'XZMSG ERROR' INTO BUFFER
CALL XRMV 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 XUDBG
ENDIF
END XZMSG
2070   1 CD0  FORTRAN CALLING PROCEDURE:
2071   1 CD0  CALL XIOPR(ENTRY)
2072   1 CD0  -------------------
2073   1 CD0  XIOPR IS USED BY THE ASGCM ROUTINE XIPS2 TO EVALUATE MATH OPERATIONS
2074   1 CD0  -----------
2075   1 CD1  INPUTS
2076   1 CD2  ENTRY - OPERATOR TOKEN CURRENTLY BEING PROCESSED
2077   1 CD2  FROM ASGCM - LU,SSTNG,SYNTAX,DATYPS,RSLTPT,CLSTRM,MAPDOS,RESULT
2078   1 CD2  -----------
2079   1 CD3  OUTPUTS TO ASGCM
2080   1 CD3  RESULT,OPRDOS,REGST,RSLTPT
2081   1 CD3  -----------
2082   1 CD4  INTERNAL VARIABLES
2083   1 CD4  -----------
2084   1 CD5  MAPOP - MAPS OPERATOR TOKENS FOR EXECUTION
2085   1 CD5  -----------
2086   1 CD6  EXTERNAL REFERENCES
2087   1 CD6  FD SPXPR,XPXIT,XXMOP,XXLSS,XXMSG
2088   1 CD6  RTE - IANC,OWF
2089   1 CD6  -----------
BEGIN XOPR

CASE OPERATOR :ADD:, :SUBTR:, :MULT:, :DIVIDE:, :INDEX:, :SUBSCR:, UNARY -

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

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

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

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

:INDEX:
IF RESULT STACK IS NOT EMPTY, THEN
GET 1-DIM FOR TOP OPERAND IN RESULT STACK
SET # WORDS PER ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK
ELSE OBJECT SUBSCRIPTOR FOR OBJECT IS BEING EVALUATED
GET 1-DIM FOR FIRST ENTRY IN SYMBOL TABLE
SET # WORDS PER ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
ENDIF
CALCULATE RESULT (I.E., INDEX = IDIM*(SECOND OPERAND-1)+FIRST OPERAND)

:SUBSCR:
IF RESULT STACK IS NOT EMPTY, THEN
SET # WORDS/ELEMENT BASED ON TOP ENTRY OF RESULT STACK ENTRY
ELSE OBJECT IS BEING SUBSCRIBED
SET # WORDS/ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
ENDIF
CALCULATE RESULT (I.E., DISPLACEMENT = (OPERAND-1)*#WORDS PER ENTRY)
IF RESULT STACK IS NOT EMPTY AND TOP ENTRY IS NOT FREE OR CHARACTER, THEN
CALL XPREP TO RETRIEVE A LOGICAL VALUE FOR TOP OPERAND AT DISPLACEMENT
DECREMENT RESULT STACK POINTER TO DISCARD ARRAY NAME
ELSE OBJECT IS BEING SUBSCRIBED
SET TYPE = -2
ENDIF

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

:UNIMIN:
CHANGE SIGN OF OPERAND FOR RESULT
ENDCASE

EXCEPT IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
PUSH RESULT AND TYPE ONTO RESULT STACK
EXIT XOPR

:OVER:
SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
BEGIN XPICS

POP OPERAND AND DATA TYPE FROM TOP ENTRY OF RESULT STACK

IF TARGET TYPE AND DATA TYPE ARE NOT EQUAL, THEN

CASE (TARGET TYPE) :INTG:, :REAL:, :DBLE:

CONVERT OPERAND TO INTEGER

CONVERT OPERAND TO SINGLE PRECISION REAL

CONVERT OPERAND TO DOUBLE PRECISION REAL

ENDCASE

ENDIF

EXIT XPICS

OVER:

SET MESSAGE TO BE OUTPUT TO “OVERFLOW OR UNDERFLOW DETECTED”

CALL XMISG TO OUTPUT MESSAGE TO USER

CALL XLIST TO LIST SYMBOLIC STRING

CALL XPXIT TO EXIT PROCESSOR

END XPICS
FORTRAN CALLING PROCEDURE:
CALL XIPS1
XIPS1 IS USED BY ASSGN TO TRANSLATE THE SYMBOLIC STRING EXPRESSION
TO A POST-FIX NOTATION STRING
INPUTS FROM ASGCOR
LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTBL
OUTPUTS TO ASGCOR
EXPTR, RNGSTK, POLISH, SYNTAX
INTERNAL VARIABLES
GRPSK - 2X120 ARRAY USED TO TRACK FUNCTIONS, SUBSCRIPTS, AND
PARENTHEtical GROUPINGS
OPSTK - 2X120 ARRAY; EACH ENTRY CONTAINS OPERATOR TOKEN AND ITS OUTPUT
PRIORITY
PRCNT - PRECEDENT TYPE FOR PRECEDING CHARACTER; USED FOR SYNTAX CHECK
TKNTR - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING
TOKEN - TOKEN CURRENTLY BEING PROCESSED
EXTERNAL REFERENCES
RTE - IAMO, MIMO
FDS - XPSIT, XILSS, XIMSG, XISYM
BEGIN SETUP
CASE TOKEN
  :INVLD; :SYMBOL; :SYMBOL; :SYMBOL;
  NAME CHARACTER + - *
  :SYMBOL; :CHAR; :PLUS; :MINUS; :ASTER;
  < > < >
  :ENDCAS; :INVLD; :INVLD; :INVLD; :INVLD;
  = ( )
  :ENDCAS; :INVLD; :INVLD; :Lparen; :ENDCAS;
  BSS ESS X BK $
  :INVLD; :ENDCAS; :INVLD; :INVLD; :INVLD;
  = . LFT BRKT BY BRKT RPT ,
  :INVLD; :ENDCAS; :ENDCAS; :ENDCAS; :ENDCAS;
  /

SYMBOL:
  IF TOKEN IS A NAME AND NEXT TOKEN IS A LEFT BRKT, THEN
  START SEARCH UNTIL FUNCTION TABLE IS CHECKED
  EXIT IF TOKEN NAME MATCHES FUNCTION NAME
  CHANGE TOKEN TO FUNCTION'S INDEX IN TABLE + 128
  PUSH COUNT-=BINARY/UNARY FLAG AND LIMIT=0 ONTO GROUPING STACK
  ENDFUNCTION
  ERREXIT TO :BADFUN:IF NO MATCH IS FOUND
  END SEARCH
  ELSE TOKEN IS AN OPERAND
  IF TOKEN IS A NAME AND NEXT TOKEN IS (, THEN
  SET SUBSCRIPT FLAG
  CALL XSYM TO STORE SYMBOL AND SUBSCRIPT FLAG
  PUSH SYMBOL INDEX ONTO EXPRESSION STACK
  ENDFUNCTION
  :CHAR:
  PUSH NEGATIVE POINTER TO CHARACTER STRING ONTO EXPRESSION STACK
  :PLUS:
  IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
  INCREMENT TO NEXT TOKEN AND EXIT TO :ENDF:
  ENDFUNCTION
  :MINUS:
  IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
  CHANGE TOKEN TO UNARY MINUS
  ENDFUNCTION
  :ASTER:
  IF NEXT TOKEN IS *, THEN
  CHANGE TOKEN TO EXPONENTIATION
  ENDFUNCTION
  :Lparen:
  IF PRECEDING TOKEN WAS A VARIABLE (DATA ELEMENT NAME), THEN
  PUSH COUNT AND LIMIT ALLWHEN DOUBLE SUBSCRIPTING ONTO GROUPING STACK
  ELSE EXPRESSION IS BEING PROCESSED
  PUSH COUNT AND LIMIT ALLOWING NO SUBSCRIPTING ONTO GROUPING STACK
  ENDFUNCTION
ENDCASE
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 XINSG TO OUTPUT MESSAGE TO USER
2332 2 CALL XILSS TO LIST SYMBOLIC STRING
2333 2 CALL XPXIT TO EXIT PROCESSOR
2334 1 END SETUP
BEGIN STRING
SET INDEX INTO SYNTAX TABLE TO MIN(TOKEN,40)
ERREXIT TO :SYNTAX1: IF THIS TOKEN IS NOT VALID ACCORDING TO SYNTAX TABLE
IF TOKEN IS NOT AN OPERAND, THEN
DO UNTIL TOKEN IS PUSHED ONTO OPERATOR STACK OR DISCARDED
IF INPUT PRIORITY OF THIS TOKEN > OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR
STACK, THEN
IF TOKEN IS , THEN
INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK
ERREXIT TO :EXECEX: IF COUNT > COMMA LIMIT FOR (TOP ENTRY IN GROUPING STK
IF COMMA LIMIT SHOWS SUBSCRIPTING IN F..GRESS (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 :SYNTAX1: 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 :BADEQL:
ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTAX2:
DISCARD TOKEN (; OR ESS)
ENDIF
POP OPERATOR STACK
POP OPERATOR ONTO EXPRESSION STACK
ELSE BRACKETS OR PARENCHESES HAVE BEEN MATCHED
IF TOP ENTRY OF OPERATOR STACK IS ( THEN
ERREXIT IF CURRENT TOKEN IS NOT ) OR :SYNTAX2:
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 :SYNTAX2:
ERREXIT TO :IFSTER: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GRPING STK COUNT=0)
ENDIF
POP OPERATOR STACK
POP GROUPING STACK
DISCARD CURRENT TOKEN
ENDIF
ENDIF
:ENDO:
INCREMENT TO NEXT TOKEN USING TOKEN LENGTH FROM SYNTAX TABLE
EXIT STRING
:SYNTAX1:
SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"
:SYNTAX2:
SET MESSAGE TO BE OUTPUT TO "PARENTHOSES OR BRACKETS DO NOT MATCH PROPERLY"
:COMERR:
SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
2400 1 BEGIN RANGE
2401 2 INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
2402 3 IF LAST TOKEN INPUT WAS 7, THEN
2403 4 DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
2404 5 ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
2405 6 CALL XITSYM TO INSERT NAME IN SYMBOL TABLE, IF NECESSARY, FLAG ENTRY AS
2406 7 AN INDEX, AND RETURN SYMBOL TABLE INDEX
2407 8 ERREXIT IF NEXT TOKENS ARE NOT "=INTEGER,INTEGER" TO :RNGSYM:
2408 9 IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
2409 10 SET INCREMENT TO -1
2410 11 ENDFI
2411 12 PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
2412 13 END DO
2413 14 ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
2414 15 ENDFI
2415 1 EXIT RANGE
2416 2 :RNGSYM:
2417 3 SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ":NAME=
2418 4 * INTEGER,INTEGER"
2419 5 CALL XIMSG TO OUTPUT MESSAGE TO USER
2420 6 CALL XVLIST TO LIST SYMBOLIC STRING
2421 7 CALL XPRINT TO EXIT PROCESSOR
2422 1 END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

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

INPUTS FROM ASSCM

LU, TOKENS, LASTY, SYMTAB, SSTRING, RESULT, OPRNDS

OPINFO, NUMWDS, DATTPS, RSLTPY, EXPRX, CLSREQ, CLSTRM, DECLAS,

RNGSTR, POLISH, SYNTAX, FNCTYL

OUTPUTS TO ASSCM

SYMTAB, RESULT, REQST, RSLTPY, OPINFO

INTERNAL VARIABLES

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

EXTERNAL REFERENCES

FDS - XPREQ, XPRET, XMOV, XICMR, XIFMC, XIFRE, XIFXD, XILSS, XIMSG,

XIFPH, XIPS, XIREF

RTE - IABS, IAND, MAXD
EXPRESS

BEGIN IPS2 TO RETRIEVE RCZ ENTRIES FOR ALL DATA ELEMENTS ON SCROLL TABLE.
INITIALIZE RANGE VALUES FOR NON-SCROLLED-FIXED-TYPE DATA ELEMENTS.
DO UNTIL ALL POLISH STRING VALUES EVALUATED (STARTING WITH SECOND ENTRY OF STRING).
IF ENTRY IS Operator, OR POLISH STRING VALID, PERFORM POLISH STRING.
ELSE IF ENTRY IS AN OPERAND, PERFORM POLISH STRING.
ELSE IF ENTRY IS NOT OPERAND, PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT.
END IF
END DO
END IPS2

REPRODUCIBILITY 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 = 0 (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   ENDF
2494 9   ENDF
2495 2  END DO
2496 2  CALL XPREQ WITH A CLOSE BUFFER REQUEST
2497 1  END TOC
1 BEGIN EVAL
2 IF OPERATOR IS A FUNCTION, THEN
3 CALL XIPCS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
4 CALL XIPMC TO PERFORM FUNCTION FOR RESULTS
5 END IF
6 ELSE THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "="
7 DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
8 CASE (TYPE REQUIREMENT+1) :SAME:, :INT:
9 :SAME:
10 IF # OPERANDS = 1, THEN
11 SET TARGET TYPE TO DATA TYPE
12 ELSE # OPERANDS = 1
13 SET TARGET TYPE TO OPERAND'S DATA TYPE
14 ENDIF
15 :INT:
16 SET TARGET TYPE TO INTEGER
17 ENDCASE
18 DO FOR EACH OPERAND
19 CALL XIPCS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
20 END DO
21 CALL XZOPR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
22 END IF
23 EXIT EVAL
24 MISERR:
25 SET MESSAGE TO BE OUTPUT TO "FREE OR CHARACTER DATA ELEMENT FOUND IN AN
26 EXPRESSION"
27 CALL XZPSG TO OUTPUT MESSAGE TO USER
28 CALL XZLSS TO LIST SYMBOLIC STRING
29 CALL XPRMT TO EXIT PROCESSOR
30 END EVAL
BEGIN REPLAC
IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
ENDIF
CASE (OBJECT DATA TYPE )
BEGIN 
:FREE::FIXED::FIXED::CHAR::CHAR::
:CHAR::CHAR::CHAR:
FINISH:
CALL XFREE TO RETRIEVE DATA AND SET UP FOR STORE
CALL FIXOBJ TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
CALL CHOBJ TO RETRIEVE DATA AND SET UP FOR STORE
ENDIF
IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT, THEN
POP OBJECT'S DISPLACEMENT FROM RESULT STACK
ELSE OBJECT HAS NOT BEEN SUBSCRIPTED
SET OBJECT'S DISPLACEMENT TO 0
ENDIF
BUILD XPRER REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S DISPLACEMENT
CALL XPREQ TO STORE DATA IN OBJECT
END REPLAC
1 BEGIN RNGSET
2 \* DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
3 \* RANGES ARE PROCESSED
4 IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
5 INCREMENT RANGE VALUE
6 ELSE
7 SET RANGE INDEX VALUE TO BEGIN VALUE
8 ENDIF
9 END BD
10 END RNGSET
2604  1 CD0  FORTRAN CALLING PROCEDURE:
2605  1 CD0  CALL XIRET(ENTRY)
2606  1 CD0  **************
2607  1 CD1  XIRET IS USED BY THE ASSGN ROUTINE XIPS2 TO RETRIEVE THE DATA VALUE
2608  1 CD1  FOR A GIVEN SYMBOL INDEX FROM THE SYMBOL TABLE AND PUSH IT ONTO THE
2609  1 CD1  RESULT STACK ALONG WITH ITS DATA TYPE
2610  1 CD1  **************
2611  1 CD2  INPUTS
2612  1 CD2  ENTRY - SYMBOL TABLE INDEX CURRENTLY BEING PROCESSED
2613  1 CD2  FROM ASSGN - SYMTAB,RSLTP
2614  1 CD2  **************
2615  1 CD3  OUTPUTS TO ASSGN
2616  1 CD3  RESULT,RSLTP
2617  1 CD3  **************
2618  1 CD5  EXTERNAL REFERENCES
2619  1 CD5  FDS - XRM0V
2620  1 CD5  RTE - IAMD
2621  1 CD5  **************
2687 1 BEGIN XSYM
2688 2  CALL XSYM TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY
2689 3  IF TOKEN IS A NAME, THEN
2690 4  IF SPECIAL PROCESSING FLAG IS SET (1=RANGE INDEX, 2=SUBSCRIPTED), THEN
2691 5  IF FLAG INDICATES RANGE INDEX, THEN
2692 6  ERRENTRY TO :BADRNG: WITH ERROR AS01 IF OBJECT (FIRST ENTRY IN TABLE)
2693 7  ERRRENTRY TO :BADRNG: WITH ERROR AS02 ENTRY IS ALREADY SUBSCRIPTED
2694 8  ERRRENTRY TO :BADRNG: WITH ERROR AS03 ENTRY IS ALREADY A DEFINED RANGE
2695 9  ENDIF
2696 10  SET ENTRY FLAG WORD TO FLAG VALUE
2697 11  ENDIF
2698 12  ENDIF
2699 13  ADD BIAS OF 256 TO SYMBOL INDEX
2700 14  EXIT XSYM

2701 2  :BADRNG:
2702 2  CALL XIMSG TO OUTPUT ERROR DESCRIPTION
2703 2  CALL XILSS TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR
2704 2  CALL XEXIT TO TERMINATE PROCESSOR
2705 1 END XSYM
FORTRAN CALLING PROCEDURE

CALL XISY (TNKPTR, SYIND)

PROVIDED KEY VALUES ARE SUITABLY LOCATED IN COMMON, XISY PROVIDES A GENERAL CAPABILITY FOR SYMBOL TABLE ACCESS FOR TOKENS INPUT VIA A SYMBOLIC STRING.

INPUT

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

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

OUTPUT

SYIND - INDEX INTO SYMBOL TABLE (SYMTAB) WHERE SYMBOL IS LOCATED.

VALUE OF SYIND INDICATES ENTRY NUMBER, I.E., 1, 2, ...

ASGCOM COMMON - LASTST, SYMTAB

INTERNAL

DISP - DISPLACEMENT INTO SYMBOL TABLE ENTRY FOR FIELD TO BE USED AS KEY. NAMES USE A VALUE OF 1, CONSTANTS A VALUE OF 0.

EDOOP - INDEX OF LAST ALLOCATED WORD IN SYMTAB. VALUE IS EQUAL TO LASTST*STWIDE

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

EXTERNAL REFERENCES

XRCPR, XRMOV

SPECIAL REMARKS

THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS

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

I TYPE (1 WO) I NAME I UNUSED (4 WORDS) I VALUE I OTHER I

CD******

CD******

CD******
BEGIN LIST
$BEGIN
EXIT
IF TOKEN IS A NAME THEN
BEGIN
SET COMPARISON DISPLACEMENT IN TABLE 1 (NAME FIELD)
DUMP TOKEN INTO BUFFER
STORE BUFFER CONTENTS INTO ALLOCATED ENTRY
EXIT IF ENTRY MATCHES TOKEN CODE
ENDIF
STORE TOKEN CODE IN TYPE FIELD OF NEXT ENTRY
END SET STANDING TO ENTRY NUMBER
END IF TOKEN IS NAME
EXIT
<table>
<thead>
<tr>
<th>SYMBOL Definition Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD : 2114</td>
</tr>
<tr>
<td>ADD1 : 1106</td>
</tr>
<tr>
<td>ADD2 : 1105</td>
</tr>
<tr>
<td>ADD3 : 1103</td>
</tr>
<tr>
<td>ADD4 : 1124</td>
</tr>
<tr>
<td>ADD5 : 1112</td>
</tr>
<tr>
<td>ADD6 : 1123</td>
</tr>
<tr>
<td>ADD7 : 1122</td>
</tr>
<tr>
<td>ADD8 : 1102</td>
</tr>
<tr>
<td>ADD9 : 1120</td>
</tr>
<tr>
<td>ADD10 : 1119</td>
</tr>
<tr>
<td>ADD11 : 1118</td>
</tr>
<tr>
<td>ADD12 : 1117</td>
</tr>
<tr>
<td>ADD13 : 1116</td>
</tr>
<tr>
<td>ADD14 : 1115</td>
</tr>
<tr>
<td>ADD15 : 1114</td>
</tr>
<tr>
<td>ADD16 : 1101</td>
</tr>
<tr>
<td>ADD17 : 1100</td>
</tr>
<tr>
<td>ADD18 : 1111</td>
</tr>
<tr>
<td>ADD19 : 1110</td>
</tr>
<tr>
<td>ADD20 : 1109</td>
</tr>
<tr>
<td>ADD21 : 1108</td>
</tr>
<tr>
<td>ADD22 : 1090</td>
</tr>
<tr>
<td>ADD23 : 1094</td>
</tr>
<tr>
<td>ADD24 : 1099</td>
</tr>
<tr>
<td>ADD25 : 1097</td>
</tr>
<tr>
<td>ADD26 : 1096</td>
</tr>
<tr>
<td>ADD27 : 1126</td>
</tr>
<tr>
<td>ADD28 : 1091</td>
</tr>
<tr>
<td>ADD29 : 1113</td>
</tr>
<tr>
<td>ADD30 : 1121</td>
</tr>
<tr>
<td>ADD31 : 1092</td>
</tr>
<tr>
<td>ADD32 : 1128</td>
</tr>
<tr>
<td>ADD33 : 1095</td>
</tr>
<tr>
<td>ADD34 : 1093</td>
</tr>
<tr>
<td>ADD35 : 1139</td>
</tr>
<tr>
<td>ASGN : 272</td>
</tr>
<tr>
<td>ASTEN : 2314</td>
</tr>
<tr>
<td>BADEDL : 2393</td>
</tr>
<tr>
<td>BADFUM : 2329</td>
</tr>
<tr>
<td>BADRNG : 2701</td>
</tr>
<tr>
<td>CALSTY : 1098</td>
</tr>
<tr>
<td>SCALSTY : 1132</td>
</tr>
<tr>
<td>CHAR : 2304</td>
</tr>
<tr>
<td>CHAR : 2590</td>
</tr>
<tr>
<td>CHRFB : 937</td>
</tr>
<tr>
<td>CHRSTR : 931</td>
</tr>
<tr>
<td>COMERR : 2389</td>
</tr>
<tr>
<td>DATA1 : 2499</td>
</tr>
<tr>
<td>DSP : 423</td>
</tr>
<tr>
<td>DREF : 2208</td>
</tr>
<tr>
<td>DEF : 483</td>
</tr>
<tr>
<td>DIV : 2125</td>
</tr>
<tr>
<td>ENCA : 2325</td>
</tr>
<tr>
<td>ENDC : 2382</td>
</tr>
<tr>
<td>ENDS : 641</td>
</tr>
</tbody>
</table>
DO - CONDITIONAL ITERATION (LOOPING) PROCESSOR

SCHEDULED BY FBX

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FBX 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 ENTER 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.
CHARACTERISTIC PASSED TO ENDDO)
RELATION - 2CH CHARACTER STRING CONTAINING RELATIONAL SYMBOL
# - NOT EQUAL
< - LESS THAN
<= OR =< - LESS THAN OR EQUAL
= - EQUAL
>= OR => - GREATER THAN OR EQUAL
> - GREATER THAN

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
BODSTK - SEE OUTPUT DEFINITION
BSERTB - EXEC COPY OF CURRENTLY EXECUTING SEQUENCE TABLE

OUTPUT TO THE MANAGER VIA RETURN PARAMETERS
RETURN - RETURN CODE TO MANAG'TR
0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL EXECUTION
3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE NUMBER FOR CONTINUED EXECUTION
8 - ABNORMAL TERMINATION. ABORT SEQUENCE EXECUTION

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

INPUT/OUTPUT FROM/TO AWA
BODSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27.8) 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 509, volume IV, figure 1.2-17)

Words 8-14 - Interface table entry for OPLMD1 (same as

in original table except for possible new

values pointing into literal area)

Words 15-21 - Interface table entry for OPLMD2 (see

above)

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

OPLMD1 & 2

Word 26 - relation ID code as follows

0 - 

1 - >

2 - >= or >

3 - =

4 - <= or <=

5 - <

Word 27 - sequence number of top of loop

Internal variables

code - array of eight acceptable relation mnemonics and

corresponding internal codes

doent - index into dostk for new 27 word entry

pruff - 64 word manager communications buffer. eight word

entries are of the form

renst - AWA management request code

class - class and type of data

name - six character data name

desc - size of data

disp - displacement into data of transaction origin

classno - class i/o number transmitted through which data is

top = index into 8e6th of top of loop

xpels = class i/o number for manager communications (set by

xpget)

Notes

Referenced routines

exec, iand, std, xpar, xpget, xparexpget, xpxl, xrccph, xrenov,

xreset, xudcg, xupaw, xupsg

Do and endo must be used in pairs

Endsk is allocated in the AWA for interprocessor communications

xesch cleans up any residual endsk after execution controller

termination

The maximum number of nested loops is 4

all nesting (structure) errors are left for execution time

detection
QUALITY OF THE ORIGINAL PAGE IS POOR.
124 1 BEGIN DO
125 2 CALL XPER TO INITIALIZE ACCESS TO XPER AND TO RETRIEVE DTYPE AND RELATN
126 2 CALL XPAM TO RETRIEVE BOOST INTO BUFFER
127 2 IF RETRIEVAL FAILED, THEN
128 2 INITIALIZE BUFFER FOR BUILDING FIRST BOOST ENTRY
129 2 ENDIF
130 2 IF BOOST IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
131 2 IF XPER INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXIST, THEN
132 2 CALL XPER TO RETRIEVE LITERALS
133 2 ENDIF
134 2 IF RELATN IS A VALID RELATION OPERATOR, THEN
135 2 SET RELATION CODE IN NEW ENTRY IN BUFFER
136 2 SET INTERFACE TABLE HEADER WITH NAME OF SINTAB AND NUMBER OF PARAMETER OF 2
137 2 DO FOR EACH OPND
138 3 MOVE OPND ENTRY INTO NEW INTERFACE TABLE BUFFER
139 3 IF OPND HAS LITERAL VALUE OR DOUBLE SUBSCRIPTS, THEN
140 4 MOVE LITERAL DATA
141 4 ADJUST LITERAL POINTERS
142 4 ENDIF
143 4 ENDDO
144 4 CALL XPER TO RETRIEVE SSBETB (EXECUTING SEQUENCE TABLE)
145 4 IF SSBETB DISPLACEMENT (SEDSP) > 0, THEN
146 5 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (0 IF END OF TABLE)
147 4 ELSE INSERTED COMMAND
148 4 EXIT TO :ERROR: IF SEQUENCE NUMBER IS ZERO (MANUAL)
149 4 LOCATE ORIGINAL SEQUENCE ENTRY
150 5 IF ORIGINAL ENTRY WAS ALSO A DO (OVERRIDE CONDITION), THEN
151 6 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (OR ZERO)
152 5 ELSE (INSERT)
153 6 SET TOP OF LOOP TO CURRENT NUMBER
154 5 ENDIF
155 4 ENDIF
156 4 CASE (:WHILE:, :UNTIL:, :OTHER:) DTYPE
157 5 :WHILE:
158 5 INVERT RELATION CODE
159 5 INITIALIZE NEST COUNTER TO 1
160 5 START SEARCH FROM TOP OF LOOP ENTRY UNTIL ALL ENTRIES HAVE BEEN EXAMINED
161 6 IF COMMAND IS ENDDO, THEN
162 7 DECREMENT NEST COUNTER
163 6 ELSE
164 7 IF COMMAND IS ANOTHER DO, THEN
165 8 INCREMENT NEST COUNTER
166 7 ENDIF
167 6 ENDIF
168 5 EXIT IF NEST COUNTER IS ZERO
169 6 SET RESET NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
170 5 END LOOP
171 5 EXIT TO :ERROR: FOR NO MATCHING ENDDO
172 5 END SEARCH
173 5 :UNTIL:
174 5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
175 5 :OTHER:
176 5 TERMINATE WITH ERROR FOR UNRECOGNIZED DTYPE
177 5 END CASE
178 4 CALL XPER TO OUTPUT NEW EXPANDED BOOST
179 3 ELSE INVALID RELATION
TERMINATE WITH ERR05 FOR INVALID RELATION
181 IF
182 ELSE BOSTK OVERFLOW
183 THEN
184 ENDIF
185 END
186 2: ERR02: TERMINATE FOR EXECUTING IN MANUAL MODE
187 2: ERR04: TERMINATE FOR NO MATCHING ENDDO FOUND DURING WHILE PROCESSING
188 1 END DO
1 CD******  
2 CD0  
3 CD0  
4 CD0  
5 CD******  
6 CD1  
7 CD1  
8 CD1  
9 CD1  
10 CD1  
11 CD1  
12 CD1  
13 CD1  
14 CD1  
15 CD1  
16 CD1  
17 CD1  
18 CD1  
19 CD1  
20 CD1  
21 CD1  
22 CD1  
23 CD1  
24 CD1  
25 CD1  
26 CD1  
27 CD1  
28 CD1  
29 CD1  
30 CD1  
31 CD1  
32 CD1  
33 CD1  
34 CD1  
35 CD1  
36 CD1  
37 CD1  
38 CD1  
39 CD1  
40 CD1  
41 CD1  
42 CD1  
43 CD1  
44 CD1  
45 CD1  
46 CD1  
47 CD1  
48 CD1  
49 CD1  
50 CD1  
51 CD1  
52 CD1  
53 CD1  
54 CD1  
55 CD1  
56 CD1  
57 CD1  
58 CD1  
59 CD1  
60 CD1  
61 CD1  
62 CD1  
63 CD1  
64 CD1  
65 CD1  
66 CD1  
67 CD1  
68 CD1  
69 CD1  
70 CD1  
71 CD1  
72 CD1  
73 CD1  
74 CD1  
75 CD1  
76 CD1  
77 CD1  
78 CD1  
79 CD1  
80 CD1  
81 CD1  
82 CD1  
83 CD1  
84 CD1  
85 CD1  
86 CD1  
87 CD1  
88 CD1  
89 CD1  
90 CD1  
91 CD1  
92 CD1  
93 CD1  
94 CD1  
95 CD1  
96 CD1  
97 CD1  
98 CD1  
99 CD1  
100 CD1  
101 CD1  
102 CD1  
103 CD1  
104 CD1  
105 CD1  
106 CD1  
107 CD1  
108 CD1  
109 CD1  
110 CD1  
111 CD1  
112 CD1  
113 CD1  
114 CD1  
115 CD1  
116 CD1  
117 CD1  
118 CD1  
119 CD1  
120 CD1  
121 CD1  
122 CD1  
123 CD1  
124 CD1  
125 CD1  
126 CD1  
127 CD1  
128 CD1  
129 CD1  
130 CD1  
131 CD1  
132 CD1  
133 CD1  
134 CD1  
135 CD1  
136 CD1  
137 CD1  
138 CD1  
139 CD1  
140 CD1  
141 CD1  
142 CD1  
143 CD1  
144 CD1  
145 CD1  
146 CD1  
147 CD1  
148 CD1  
149 CD1  
150 CD1  
151 CD1  
152 CD1  
153 CD1  
154 CD1  
155 CD1  
156 CD1  
157 CD1  
158 CD1  
159 CD1  
160 CD1  
161 CD1  
162 CD1  
163 CD1  
164 CD1  
165 CD1  
166 CD1  
167 CD1  
168 CD1  
169 CD1  
170 CD1  
171 CD1  
172 CD1  
173 CD1  
174 CD1  
175 CD1  
176 CD1  
177 CD1  
178 CD1  
179 CD1  
180 CD1  
181 CD1  
182 CD1  
183 CD1  
184 CD1  
185 CD1  
186 CD1  
187 CD1  
188 CD1  
189 CD1  
190 CD1  
191 CD1  
192 CD1  
193 CD1  
194 CD1  
195 CD1  
196 CD1  
197 CD1  
198 CD1  
199 CD1  
200 CD1  
201 CD1  
202 CD1  
203 CD1  
204 CD1  
205 CD1  
206 CD1  
207 CD1  
208 CD1  
209 CD1  
210 CD1  
211 CD1  
212 CD1  
213 CD1  
214 CD1  
215 CD1  
216 CD1  
217 CD1  
218 CD1  
219 CD1  
220 CD1  
221 CD1  
222 CD1  
223 CD1  
224 CD1  
225 CD1  
226 CD1  
227 CD1  
228 CD1******
BEGIN ELSE
   CALL RMPAR TO GET SCHEDULING PARAMETERS
   CALL ZXFCL TO ESTABLISH FDS MANAGER'S CLASS NO. (XPCLS)
   CALL ZISCN TO SEARCH FOR MATCHING ENDEF COMMAND
   CALL ZPXIT TO RETURN PARAMETERS TO FDS MANAGER
END ELSE
CD0************
CD0          ENDDO - LOOP TERMINATION PROCESSOR
CD0          SCHEDULED BY FDS
CD0************
CD1          THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FDS USERS WITH THE
CD1          CAPABILITY OF LOOPING THROUGH A BLOCK OF SEQUENCE TABLE ENTRIES
CD1          EITHER UNTIL A GIVEN CONDITION IS SATISFIED OR WHILE THE CONDITION
CD1          IS TRUE.
CD1************
CD2          INPUT FROM MANAGER VIA SCHEDULING PARAMETERS
CD2          LU - LOGICAL UNIT OF USER'S TERMINAL
CD2          FLAGS - DEBUG FLAGS FROM USER SIGN-ON
CD2************
CD3          OUTPUT TO THE MANAGER VIA RETURN PARAMETERS
CD3          RETURN - RETURN CODE TO MANAGER
CD3          0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL
CD3          EXECUTION
CD3          3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE
CD3          NUMBER FOR CONTINUED EXECUTION
CD3          8 - ABNORMAL TERMINATION. ABORT SEQUENCE EXECUTION
CD3          RESET - SEQUENCE RESET NUMBER OF TOP OF LOOP IF RETURN = 3
CD3************
CD4          INTERNAL VARIABLES
CD4          DOENT - INDEX INTO EDOSTK FOR LAST 27 WORD ENTRY
CD4          MRBUFF - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD
CD4          ENTRIES ARE OF THE FORM
CD4          RESGT - AWA MANAGEMENT REQUEST CODE
CD4          CLASS - CLASS AND TYPE OF DATA
CD4          NAME - SIX CHARACTER DATA NAME
CD4          SIZE - SIZE OF DATA
CD4          DISP - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN
CD4          CLASN0 - CLASS I/O NUMBER THROUGH WHICH DATA IS
CD4          TRANSMITTED
CD4          XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS
CD4************
CD5          REFERENCED ROUTINES
CD5          EXEC, IAND, IMPAR, XPRER, XPRER(XPRER), XPRX, XRLOC, XRMOV,
CD5          XUDRG, XPFW, XIMSG
CD0************
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 GINTAB FROM FIRST CLASS BUFFER INTO AWA
7 IF RESTORE SUCCESSFUL, THEN
8 CALL XPGET TO RETRIEVE OPRMD
9 CALL XEVL 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 ENDF
17 ELSE
18 SET RESET NUMBER TO TOP OF LOOP
19 ENDF
20 ELSE
21 TERMINATE WITH ERR06 FOR AWA OVERFLOW
22 ENDF
23 ELSE
24 TERMINATE WITH ERR03 FOR ENDDO WITH OUT DO
25 ENDF
26 END ENDDO
FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

THE ENDIF UTILITY PROCESSOR RETURNS TO THE FDS MANAGER (VIA XPX1T) SO
THAT NORMAL PROCESSOR EXECUTION SEQUENCE MAY CONTINUE

INPUTS

SCHEDULING PARAMETERS

LU = LOGICAL UNIT NO. OF FDS USER
FLAGS = DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT

OUTPUTS

RMPAMS - RETURN PARAMETERS FOR FDS MANAGER VIA XPX1T
(C1) = 0 => CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE

Routines Used

RMPAR
XREXT
XPX1T
XUDBG
364 1 C6********
365 1 C60   FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
371 1 C61   IF UTILTY PROCESSOR COMPARES TWO INPUT QUANTITIES ACCORDING TO A
372 1 C61   SPECIFIED RELATION. WHEN THE COMPARISON IS FALSE, THE SEQUENCE RESET
373 1 C61   NUMBER IS SET (VIA XPSET) TO THE PROCESSOR FOLLOWING THE CORRESPONDING
374 1 C61   ELSE IF CRITICISM IS TRUE THE NORMAL PROCESSOR
375 1 C61   EXECUTION SEQUENCE IS FOLLOWED.
377 1 C61   C62********
378 1 C62   INPUT
379 1 C62   C62   SCHEDULING PARAMETERS
380 1 C62   L0 -LOGICAL UNIT NO. OF THIS FDS USER
381 1 C62   C62   FLGS - DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT
384 1 C62   C62   INTERFACE TABLE PARAMETERS
386 1 C62   C62   OPRND1 - FIRST REAL VALUE TO COMPARE
387 1 C62   C62   OPRHID2 - SECOND REAL VALUE TO COMPARE
388 1 C62   C62   C62********
389 1 C62   OUTPUT
391 1 C62   C62   RPAPMS - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPXIT
392 1 C62   C62   (1) = 0 => CONTINUE WITH NORMAL PROCESSOR EXECUTION SEQUENCE
395 1 C62   C62   = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE
396 1 C62   C62   NO. GIVEN IN RPAPMS(2)
397 1 C62   C62   = 8 => ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE
398 1 C62   C62   (2) = SEQUENCE NO. TO BE EXECUTED NEXT WHEN RPAPMS(1) = 3
400 1 C62   C62********
401 1 C62   REFERENCES Routines
404 1 C62   C62********
405 1 C62   RMPAR
406 1 C62   XPSET
407 1 C62   XPXIT
408 1 C62   XIEY
409 1 C62   XZSN
410 1 C62   XNEXT
411 1 C62   XRDG
412 1 C62   C63********
1 BEGIN IF
2 CALL HIPAR TO GET INPUT (SCHEDULING) PARAMETERS
3 CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
4 VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSLATE IT TO A CODE
5 ERREXIT IF RELATIONAL OPERATOR INVALID :ERROR3:
6 CALL XIVAL TO EVALUATE THE RELATIONAL EXPRESSION
7 IF THE EXPRESSION IS FALSE, THEN
8 CALL XIXCH TO LOCATE 'THE ELSE OR ENL:' CORRESPONDING TO THIS IF
9 AND SET SEQUENCE RESET NUMBER
10 ELSE
11 CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE
12 ENDIF
13 EXIT IF
14 IF :ERROR3: CALL XIXMSG 'INVALID RELATIONAL OPERATOR - MUST BE #,>,>=,<,<=,OR =>
15 END IF
FORTRAN CALLING PROCEDURE

LVALUE = XIEVL (OPRND1, RELATN, OPRND2)

XIEVL 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
OPRND2 - SECOND REAL VALUE TO COMPARE
RELATN - RELATION CODE AS FOLLOWS
0 - NOT EQUAL
1 - GREATER THAN
2 - GREATER THAN OR EQUAL
3 - EQUAL
4 - LESS THAN OR EQUAL
5 - LESS THAN

OUTPUT

XIEVL - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

INTERNAL VARIABLES

TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS
FOLLOWS
OPRND1 - OPRND2
OPRND1 - OPRND2
CODE RELATION + 0 - RELATION + 0 -

0 $ T F T $ NOT= T F T
1 $ T F T
2 > T F T
3 = F T F
4 < F T T

(THE COMPLEMENTING CODE > 2)

THEORETICAL FOR OPRND1 - OPRND2 POSITIVE (BITS 0-2) 1, 1, 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 RELATN IS 0, 1, OR 2 ON THE
COMPLEMENT IF RELATN IS 3, 4, OR 5

REFERRED ROUTINES
XREX

NO CHECKS FOR VALID RELATION CODES OR OVERFLOW/UNDERFLOW ARE MADE
1 BEGIN X:EV
2 CASE (+:0, =:0): DIFFERENCE OF OPRAND1 AND OPRAND2
3 1+: SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)
3 0: SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
4 END CASE
5 IF RELATH > 2 (BOTTOM OF TRUTH TABLE), THEN
6 COMPLEMENT TTABLE
7 DECREMENT RELATH BY 3
8 ENDIF
9 ADD RELATH TO FIELD OFFSET (INDEXES TO CORRECT 4TH FOR RELATH AND DIFFERENCE)
10 SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
11 END X:EV.
**FORTRAN CALLING PROCEDURE**

**CALL ZISCH (SRCFG, RPARMS)**

**ZISCH IS CALLED TO LOCATE THE CORRESPONDING ELSE/ENDIF IN THE CURRENT SEQUENCE TABLE. THE RETURN PARAMETERS FOR THE FBS MANAGER ARE OUTPUT**

**INPUTS**

**SRCFG - FLAG INDICATING ORIGINATING 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

**RPARMS - SCHEDULED PARAMETERS FROM THE FBS MANAGER**

- (1) - LOGICAL UNIT NO. OF THE FBS USER
- (5) - SERSIP, INDEX INTO SEQUENCE TABLE (&SESEQ) OF THE CURRENT COMMAND

**OUTPUTS**

**RPARMS - PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XPAR**

- (1) - 0 => CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
- 3 => EXECUTE SEQUENCE NO. GIVEN BY RPARMS(2) NEXT
- 6 => ABNORMALLY TERMINATE PROCESSOR EXECUTION SEQUENCE
- (2) - SEQUENCE NO. TO BE EXECUTED NEXT IF RPARMS(1) = 3

**Routines used**

- EXEC
- RMPAR
- XVPAM
- XUXDG
- XRCPR
- XVMSG
- VXSG
560 1 BEGIN XISCH
561 2  RETRIEVE $SEGTAB FROM THE AWA USING XVPAM
562 3  STARTSEARCH UNTIL ALL COMMANDS IN $SEGTAB
563 4  EXIT IF CURRENT COMMAND IS FOUND
564 5  SET NUMBER OF IF NESTS TO 1
565 6  STARTSEARCH FROM NEXT COMMAND IN $SEGTAB UNTIL ALL FOLLOWING COMMANDS
566 7  IF COMMAND IS FOR ENDIF PROCESSOR, THEN
567 8  DECREMENT NUMBER OF IF NESTS BY 1
568 9  ELSE
570 10  IF COMMAND IS FOR IF PROCESSOR, THEN
571 11  INCREMENT NUMBER OF IF NESTS BY 1
572 12  ELSE
573 13  IF CALLED BY IF PROCESSOR, AND
574 14  COMMAND IS FOR ELSE PROCESSOR, THEN
575 15  ERREXIT IF THIS IS THE END OF $SEGTAB :ERROR1:
576 16  IF NUMBER OF IF NESTS IS 1, THEN
577 17  DECREMENT NUMBER OF IF NESTS TO 0
578 18  ENDIF
579 19  ENDIF
580 20  ENDIF
581 21  EXIT IF NUMBER OF IF NESTS IS 0
582 22  SET SEQUENCE RESET NUMBER (RPARMS(2)) TO BE SEQUENCE NUMBER OF THE
583 23  NEXT COMMAND IN THE TABLE
584 24  ENDLOOP
585 25  ERREXIT :ERROR1:
586 26  ENDSWITCH
587 27  ENDSWITCH
588 28  ERREXIT :ERROR4:
589 29  ENDSWITCH
590 30  1 EXIT XISCH
591 31  :ERROR1: CALL X2MSG - 'IF CANNOT "E EXECUTED WITHOUT MATCHING ENDIF''
592 32  :ERROR4: CALL X.MSG - 'SYSTEM ERROR - NO $SEGTAB'
593 33  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.
BLOCK F.
FURPUR 27R1  RL72-8  10/05/78  08:39:23
END PACK.  TEXT=144,TOC=1,SYM=24,REL=5,ABS=1

QPRT JL F.
FNM-191697/FDSIPDL(1) ELEMENT TABLE

<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>PSRMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td></td>
<td>ERT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>55</td>
<td>0</td>
<td>1</td>
<td>17912</td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ERT SYMB</td>
<td>15 APR 77</td>
<td>12:54:07</td>
<td>2</td>
<td>30</td>
<td>0</td>
<td>1</td>
<td>1843</td>
</tr>
<tr>
<td>SORT2</td>
<td></td>
<td>RELOCATABLE</td>
<td>15 APR 77</td>
<td>12:59:32</td>
<td>3</td>
<td>55</td>
<td>0</td>
<td>1</td>
<td>1875</td>
</tr>
<tr>
<td>XUPDL</td>
<td></td>
<td>ERT SYMB</td>
<td>22 MAR 77</td>
<td>03:10:22</td>
<td>4</td>
<td>84</td>
<td>0</td>
<td>1</td>
<td>1932</td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ERT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>0</td>
<td>2</td>
<td>2016</td>
</tr>
<tr>
<td>RAP</td>
<td></td>
<td>ERT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2103</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></td>
<td>2105</td>
</tr>
<tr>
<td>XC</td>
<td></td>
<td>ERT SYMB</td>
<td>08 NOV 77</td>
<td>05:47:05</td>
<td>8</td>
<td>109</td>
<td>5</td>
<td>6</td>
<td>2385</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>10 NOV 77</td>
<td>06:19:22</td>
<td>9</td>
<td>394</td>
<td>5</td>
<td>2</td>
<td>2694</td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td>ERT SYMB</td>
<td>11 JAN 78</td>
<td>23:22:28</td>
<td>10</td>
<td>85</td>
<td>5</td>
<td>12</td>
<td>2888</td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ERT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>373</td>
<td>5</td>
<td>8</td>
<td>2923</td>
</tr>
<tr>
<td>XL</td>
<td></td>
<td>ERT SYMB</td>
<td>13 FEB 78</td>
<td>19:17:22</td>
<td>12</td>
<td>824</td>
<td>5</td>
<td>9</td>
<td>3296</td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ERT SYMB</td>
<td>18 FEB 78</td>
<td>12:48:36</td>
<td>13</td>
<td>627</td>
<td>5</td>
<td>14</td>
<td>4120</td>
</tr>
<tr>
<td>XV</td>
<td></td>
<td>ERT SYMB</td>
<td>22 FEB 78</td>
<td>23:50:36</td>
<td>14</td>
<td>320</td>
<td>5</td>
<td>20</td>
<td>4747</td>
</tr>
<tr>
<td>COMM:N</td>
<td></td>
<td>ERT SYMB</td>
<td>03 MAR 78</td>
<td>22:20:04</td>
<td>15</td>
<td>200</td>
<td>5</td>
<td>20</td>
<td>4857</td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>RELOCATABLE</td>
<td>11 MAR 76</td>
<td>03:16:51</td>
<td>16</td>
<td>2</td>
<td>19</td>
<td></td>
<td>5007</td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>RELOCATABLE</td>
<td>11 MAR 76</td>
<td>03:17:56</td>
<td>17</td>
<td>1</td>
<td></td>
<td></td>
<td>5088</td>
</tr>
<tr>
<td>XP</td>
<td></td>
<td>ERT SYMB</td>
<td>17 APR 76</td>
<td>10:54:29</td>
<td>18</td>
<td>101</td>
<td>5</td>
<td>10</td>
<td>5094</td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>ERT SYMB</td>
<td>17 APR 76</td>
<td>10:54:52</td>
<td>19</td>
<td>534</td>
<td>5</td>
<td>22</td>
<td>5193</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>27 APR 76</td>
<td>11:43:48</td>
<td>20</td>
<td>384</td>
<td>5</td>
<td>6</td>
<td>5729</td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td>ERT SYMB</td>
<td>18 MAY 78</td>
<td>09:12:58</td>
<td>21</td>
<td>1548</td>
<td>5</td>
<td>4</td>
<td>6113</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>28 AUG 76</td>
<td>12:52:31</td>
<td>22</td>
<td>403</td>
<td>5</td>
<td>20</td>
<td>7661</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>28 AUG 76</td>
<td>12:52:39</td>
<td>23</td>
<td>504</td>
<td>5</td>
<td>20</td>
<td>8064</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>28 AUG 76</td>
<td>12:52:42</td>
<td>24</td>
<td>151</td>
<td>5</td>
<td>5</td>
<td>8372</td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ERT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>25</td>
<td>118</td>
<td>5</td>
<td>13</td>
<td>8723</td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td>ERT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>26</td>
<td>1201</td>
<td>5</td>
<td>21</td>
<td>8841</td>
</tr>
<tr>
<td>XN</td>
<td></td>
<td>ERT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:39</td>
<td>27</td>
<td>800</td>
<td>5</td>
<td>29</td>
<td>10042</td>
</tr>
<tr>
<td>XZ</td>
<td></td>
<td>ERT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:57</td>
<td>28</td>
<td>328</td>
<td>5</td>
<td>2</td>
<td>10642</td>
</tr>
</tbody>
</table>

NEXT AVAILABLE LOCATION:
ASSEMBLER PROCEDURE TABLE EMPTY
COBOL PROCEDURE TABLE EMPTY
FORTRAN PROCEDURE TABLE EMPTY
ENTRY POINT TABLE EMPTY

BKPT F.POLIST