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
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.
FDS-1 SYSTEM DESIGN DOCUMENT

Volume I  Introduction, Overview, and User Interface

Volume II  Utility Processor Library

Volume III  Processor Library

Volume IV  System Architecture and Executive

Volume V  Data Management and Data Base Documentation Support System

Volume VI  Standards

Volume VII  Utility Support Software

Volume VIII  Build and Delivery Procedures, Software Development, Debug, and System Build Aids

Volume IX  Executive Logic Flow - Program Design Language

Volume X  Document Change Request Procedure and Submittal Form

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

<table>
<thead>
<tr>
<th>Section</th>
<th>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)

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

********

XE CONTENTS

1. INTEGER
   2. CARTPG
   3. CLASMO
   4. COMPTR
   5. EEND
   6. EXIT
   7. OOLDIND
   8. QUAL
   9. REQBUF(64)
   10. RENPT
   11. SEREND
   12. SERNAME
   13. SUBSTA
   14. TABEND
   15. TKMLNG
   16. TOKEMS(32)
   17. XE

DIRECTION

1. INTNAM(3)

EQUIVALENCE

1. (XE(1)) = (CLASMO)
   2. (XE(3)) = (QUAL)
   3. (XE(5)) = (MASSTA)
   4. (XE(7)) = (SEGNAME(1))
   5. (XE(11)) = (SEREND)
   6. (XE(13)) = (INTNAM(3))
   7. (XE(19)) = (REPT)
   8. (XE(80)) = (TKMLNG)
   9. (XE(159)) = (EXTEND)
   10. (XE(141)) = (EOLDIND)
   11. (XE(142)) = (CARTRG)
   12. (XE(143)) = (EPROC)
   13. (XE(144)) = (COMPTR)

.. CONTINUE

CARTDG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

CLASMO - EXECUTIVE/MANAGER REQUEST BLOCK CLASS I/O NUMBER

COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER

1. NUMBER OF TOKENS IN BUFFER
   2. NUMBER OF USED WORDS IN BUFFER

COMPTR - TOKENS REPRESENTING USER'S RESPONSE

EXTEND - SEQUENCE # WHERE EXECUTION IS TO END (RETURN TO %)

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

BETS 0-10 UNUSED

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

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

**MASA** — 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 NODE 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
- INITIALIZATION FROM DIRECTIVE
- INITIALIZATION FOR REENTRY

**BITS 12-13** — EXECUTION CONTROL NODE 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

**HPROC** — NUMBER OF PROCESSORS IN LIBRARY

**OLDIND** — OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

**PROCMA** — 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)

**REBUF** — BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SDO 6.2.7)

**REPT** — POINTER TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUF (0 INDICATES REBUF EMPTY) OR RETURN CODE FLOW XRD

**SEG** — 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

**SERT** — POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SEMI OR AUTO MODE

**SERST** — INITIAL SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN SEMI OR AUTO MODE

**SUBSTA** — EXECUTIVE SUB-STATE FLAG (LEVEL IN COMMUNICATION WITH USER TERMINAL). SET TO LEVEL TO BE INITIALIZED NEXT OR ZERO IF LEVEL INITIALIZATION FAILS.
147 1 C*  XN CONTENTS (DIRECTIVE LEVEL)
148 1 CD  INTEGER,
150 1 CD  * DIRECT(50)
152 1 CD  * EQUIVALENCE
153 1 CD  * (XN(1),NUMDIR)
154 1 CD  * (XN(2),DIRECT(1))
155 1 CD  * (XN(151),BEGINNING OF DIRECTIVE DEPENDENT ALLOCATION)
156 1 CD  * DIRECT - FIXED ORDER LIST OF FDS DIRECTIVES (FOUR CHARACTERS PER
157 1 CD  DIRECTIVE)
158 1 CD  NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
160 1 CD********
<table>
<thead>
<tr>
<th>Line No.</th>
<th>Line Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>162</td>
<td>1  CD</td>
</tr>
<tr>
<td>163</td>
<td>1  CD</td>
</tr>
<tr>
<td>164</td>
<td>INTEGER</td>
</tr>
<tr>
<td>165</td>
<td>1  CD</td>
</tr>
<tr>
<td>166</td>
<td>* ASCENT</td>
</tr>
<tr>
<td>167</td>
<td>1  CD</td>
</tr>
<tr>
<td>168</td>
<td>* SEINO</td>
</tr>
<tr>
<td>169</td>
<td>1  CD</td>
</tr>
<tr>
<td>170</td>
<td>* DIMENSION</td>
</tr>
<tr>
<td>171</td>
<td>1  CD</td>
</tr>
<tr>
<td>172</td>
<td>* SEXTAB(150)</td>
</tr>
<tr>
<td>173</td>
<td>1  CD</td>
</tr>
<tr>
<td>174</td>
<td>EQUIVALENCE</td>
</tr>
<tr>
<td>175</td>
<td>1  CD</td>
</tr>
<tr>
<td>176</td>
<td>* (XB1, NPROC2)</td>
</tr>
<tr>
<td>177</td>
<td>1  CD</td>
</tr>
<tr>
<td>178</td>
<td>* (XB(235), RESIND)</td>
</tr>
<tr>
<td>179</td>
<td>1  CD</td>
</tr>
<tr>
<td>180</td>
<td>* (XB(2,6), CURIND)</td>
</tr>
<tr>
<td>181</td>
<td>1  CD</td>
</tr>
<tr>
<td>182</td>
<td>* (XB(250), SELEN)</td>
</tr>
<tr>
<td>183</td>
<td>1  CD</td>
</tr>
<tr>
<td>184</td>
<td>* (XB(251), SEXTAB(150))</td>
</tr>
<tr>
<td>185</td>
<td>1  CD</td>
</tr>
<tr>
<td>186</td>
<td>ASCENT - SEQUENCE TABLE ENTRY IN ASCII TO PROMPT USER IN SEMI MODE ONLY</td>
</tr>
<tr>
<td>187</td>
<td>CURIND - CURRENT INDEX TO EXECUTING SEQUENCE ENTRY</td>
</tr>
<tr>
<td>188</td>
<td>LIBR - LIBRARY DIRECTORY PROCESSOR NAME TABLE</td>
</tr>
<tr>
<td>189</td>
<td>NPROC2 - NUMBER OF PROCESSORS IN LIBR (SAME AS XE(143))</td>
</tr>
<tr>
<td>190</td>
<td>RESIND - INDEX OF RESET ENTRY WHEN RESET SEQUENCE 0 IS REQUESTED</td>
</tr>
<tr>
<td>191</td>
<td>SELEN - LENGTH OF SEQUENCE TABLE</td>
</tr>
<tr>
<td>192</td>
<td>SENO - NUMBER OF ENTRIES IN SEQUENCE TABLE</td>
</tr>
<tr>
<td>193</td>
<td>SEXTAB - SEQUENCE TABLE CURRENTLY BEING EXECUTED</td>
</tr>
<tr>
<td>194</td>
<td>**</td>
</tr>
</tbody>
</table>
EQUIVALENCES

* (XEB(27), DEBUG)  =  (XEB(28), OLDTAB)
  (XEB(31), REPUT)  =  (XEB(32), PRGFND)
  (XEB(35), MUMENT) =  (XEB(36), PRLMENT)
  (XEB(37), PROMPT) =  (XEB(49), INC.)
  (XEB(50), ISNMO)  =  (XEB(51), PRCMAM)
  (XEB(54), INTDOWNLOAD) =  (XEB(57), TADBORDER)
  (XEB(58), TABSIZE) =  (XEB(59), INSERT)
  (XEB(60), LICE-SIZE) =  (XEB(61), BEGMO)
  (XEB(62), EMENG)  =  (XEB(100), UKBUF)
  (XEB(101), UKBUF) =  (XEB(120), UKBUF)

DIMENSIONS

*INITN(3)  =  NEWTAB(3)
*OLDUNIT(3) =  PRECNUM(3)
*PROMPT(12) =  WRKBUF(1050)
*XLIBL(192)

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

BEGNO = STRING IN UKBUF TO FIRST TABLE ENTRY TO BE LISTED OR DELETED
ENMO = INDEX IN UKBUF TO LAST TABLE ENTRY TO BE LISTED OR DELETED

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

INTMAN = INTERFACE TABLE NAME INPUT OR 0

LINTC = RETURN CODE SET TO -1 WHEN THERE IS TO TERMINATE

LDOSZ = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBD

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

OLDSZ = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

PRLLEN = LENGTH IN WORDS OF PROMPT CREATED BY XSPRN

PRINTM = CURRENT PROMPTING MODE:

= 1 => UPDATE MODE ( # : )
= 2 => CREATE OR READING MODE ( # 200 : )
= 3 => ALL MODE ( # 200=PROC,TAKE )

PROMPT = PROMPT BUILT BY XSPRN

SESNMO = CURRENT SEQUENCE NUMBER TO BE PROMPTED

TABDIN = SIZE IN WORDS OF TABLE IN UKBUF

TABDIN = INDEX IN UKBUF TO CURRENT TABLE ENTRY

UKBUF = MAXIMUM SIZE IN WORDS OF UKBUF

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

1 CD INTEGER BITNO
2 CD * ARGNO BITNUM
3 CD * DEBUG DIRECT(6)
4 CD * DFLAG EQUIAT(3)
5 CD * PROMPT(30) PRMTHD
6 CD * VERSION WBUFF(1300)
7 CD * DMLNG
8 CD * ISIZE(IS) NEWTAG(3)
9 CD EQUIVALENCE
10 CD * ,XB(23) COMFLG
11 CD * ,XB(24) ICCLASS
12 CD * ,XB(26) VERSION
13 CD * ,XB(28) LISTLU
14 CD * ,XB(29) NODSAY
15 CD * ,XB(31) IDIM
16 CD * ,XB(33) SFLAG
17 CD * ,XB(35) CFLAG
18 CD * ,XB(37) ARGNO
19 CD * ,XB(39) ISUB
20 CD * ,XB(41) PRMTHD
21 CD * ,XB(43) PROMPT(1)
22 CD * ,XB(45) ISIZE(IS)
23 CD * ,XB(47) CTYPE
24 CD * ,XB(49) BITNO
25 CD * ,XB(55) BITNUM
26 CD * ,XB(57) NODSAY
27 CD * ,XB(59) LITDMN
28 CD * ,XB(61) LITLMN
29 CD * ,XB(63) LITIDM
30 CD * ,XB(65) LITIDM
31 CD * ,XB(67) LITLMN
32 CD * ,XB(69) LITDMN
33 CD * ,XB(71) EDIT
34 CD * ,XB(73) EDIT
35 CD * ,XB(75) ARG
36 CD * ,XB(77) NEWTAG(3)
37 CD * ,XB(100) WBUFF
38 CD * ,XB(101) WBUFF
39 CD ARGO - CURRENT ARGUMENT'S NO. (I.E. 1 TO 64)
40 CD ARGPR - INDEX IN WBUFF TO ARGUMENT PROMPT FOR CURRENT ARGUMENT
41 CD BITNO - BIT NO. IN BIT MASK CURRENTLY BEING PROCESSED
42 CD BITNUM - BIT NO. IN BIT MASK OF NEXT DIFFERING BIT
43 CD COMFLG - FLAG SET TO 1 IF A CONTINUATION PROMPT OF CURRENT ARGUMENT
44 CD DIRECT - ASCII ARRAY OF VALID INTERFACE TABLE EDIT DIRECTIVES
45 CD DEBUG - FLAG SET TO VALUE OF DEBUG FLAG (BITS 13-15 OF XE(4))
46 CD DFLAG - FLAG SET TO VALUE OF DATA EDIT FOR THIS ARGUMENT
47 CD SFLAG - FLAG SET TO I/O BITS FOR THIS ARGUMENT
48 CD IRET - GENERAL RETURN FLAG (0=OK, -1=ERROR, 5='EXIT')
49 CD SIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
50 CD SIZET - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
51 CD ISIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
52 CD ICLASS - CLASS OF THIS ARGUMENT
53 CD IOFLAG - FLAG SET TO I/O BITS FOR THIS ARGUMENT
54 CD ISUB - CURRENT EFFECTIVE SUBSCRIPT FOR THIS ARGUMENT
55 CD ITYPE - TYPE FLAG FOR THIS ARGUMENT
1 i:
JSUB - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
1 CD
LENFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
1 CD
LISTLU - LU TO WHICH PRINT SHOULD GO (USED WHEN
1 CC
LST DIRECTIVE CALLS XILSD OR XICHX)
1 CD
LITDSP - DISPL. TO LITERAL DATA FOR THIS ARGUMENT
1 CD
LITBUF - INDEX IN WKBUF TO END OF LITERAL DATA
1 CD
LITLEN - LENGTH OF LITERAL DATA AREA OF WKBUF
1 CD
LITPTR - INDEX IN WKBUF TO START OF LITERAL DATA
1 CD
LITSIZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
1 CD
LSTFLG - FLAG USED TO DETERMINE ORIGIN OF A
1 CD
CALL TO XILSD OR XICHX.
1 CD
= 0, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
1 CD
AN INTERFACE TABLE
1 CD
= 3, CALLED FROM INTERFACE TABLE EDITOR TO LIST ARGUMENT
1 CD
DATA, PARAMEER OR INCOMPLETE INDICATORS
1 CD
= 4, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
1 CD
A DATA ELEMENT'S VALUE(S)
1 CD
MODAV - PREVIOUS VALUE OF PRMTOK WHILE PRMTOK = 4 (CONTINUE)
1 CD
NARG - INDEX IN WKBUF TO START OF SHORT PROMPTS
1 CD
NOKAVE - INDEX IN WKBUF TO BIT MASK WORD(S) FOR THIS ARGUMENT
1 CD
WENTS - ASCII NAME OF TABLE BEING GENERATED
1 CD
NODIM - NO. OF BIT MASKS ASSOCIATED WITH THIS ARGUMENT
1 CD
NUMARG - NO. OF ARGUMENTS IN THIS INTE. TABLE
1 CD
NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
1 CD
PRMLN - LENGTH (IN WORDS) OF PROMPT BUILT
1 CD
PRMTOK - CURRENT PROMPTING MODE
1 CD
= 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
1 CD
= 3, PROMPT WITH A LIST OF CURRENT VALUES
1 CD
= 4, CONTINUATION OF SAME ARGUMENT
1 CD
= 5, PROMPT WITH "/" (CREATE MODE)
1 CD
PROMPT - ASCII ARRAY FOR PROMPT BUILT
1 CD
SFLAG - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
1 CD
VERSN - VERSION NO. OF THIS INTERFACE TABLE
1 CD
WKBLNG - LENGTH OF WKBUF
1 CD
WKBUF - WORKING BUFFER FOR INTERFACE TABLE BEING EDITED
1 CD
ORGANIZED AS :

HEADER (7 WORDS)
1 CD
ARGUMENT CHARACTERISTICS (NUMARG+2 WORDS)
1 CD
ARGUMENT PROMPTS (NUMARG+3 WORDS)
1 CD
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.

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
FD5 ERROR MESSAGES

**ATTENTION FUNCTION**

**XAD0** MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION

**XAD0** USER INITIATED INTERRUPT ENTER REQUEST - KILL(S), STATUS(S), RETURN( )

**XAD5** ERROR, LU IS NOT SIGNED ON TO FD5

**XAD4** FD5 ATTENTION FUNCTION TERMINATING

**XAD5** FD5 MANAGER SIGNALS TO TERMINATE BOTTOM ASSOC TASK

**XAD6** FD5 EXECUTIVE ACTIVE - NO ACTION TAKEN

**XAD7** FD5 PROCESSOR 'NAME' SCHEDULED TO ABORT

**XAD3** MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN

**XB** BATCH JOB CREATION

**XC** CONFIGURATION PROGRAMS

**XCO** LU 'NN' SIGNED ON TO FD5

**XCO** LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED

**XCO** -LU 'LU' IS AN INVALID LU

**XCO** FD5 CURRENTLY AT MAX USER'S. CANNOT SIGN ON

**XCO** 'LU' IS ALREADY SIGNED ON TO FD5

**XCO** CANNOT FIND 'NAME' ID - SIGN ON TERMINATED

**XCO** ENTER VALID USER ID (A-Z)

**XM** MANAGER

**XM0** INVALID REQUEST II FROM 'NAME'

**XM0** FD5 SIGNS OFF FOR LU 'LU'

**XM0** INVALID BACK CHAIN FOR CURRENT PROCESSOR

**XM0** 'M' TRACKS NOT AVAILABLE FOR DWA

**XM0** NO DWA SPACE FOR 'NAME'

**XM0** NO AWA FOR DWA DIRECTORY

**XM0** I/O ERROR FOR DWA, STATUS = 'NNNNNN'

**XM0** TOC SEARCH ERROR, PHASE 3 COMPRESS

**XP** PROCESSOR SERVICES

**XP0** TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED

**XP0** DATA AREA OVERFLOW

**XP0** SUBSCRIPT OUT OF RANGE

**XP0** INVALID ENTRY AFTER SUBSCRIPT

**XP0** INVALID REPEAT ENTRY

**XP0** TOO MANY NESTED REPEATS

**XP0** INVALID SUBSCRIPT

**XP0** RESPONSE IS TOO LONG FOR BUFFER

**XP0** INVALID RESPONSE ON OR AFTER COLUMN NN

**XP0** PROCESSOR INITIALIZATION ERROR

**XP1** AWA ACCESS FAILURE FOR .......

**XP1** PARAMETER I/O INCONSISTENCY

**XP1** INVALID PARAMETER REQUEST

**XP1** INVALID ORDE FILE ATTRIBUTE(S)

**XP15** RETRIEVAL OF TOO MUCH DATA REQUESTED

**XV** SYSTEM SERVICES

**XV0** 'ID-NAME' NAMED PROGRAM MADE A PAM REQUEST RESERVED FOR MANAGER

**XV0** 'ID-NAME' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER REPLY

**XV0** '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 CD1
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

**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
- COLUMN FROM THE PREVIOUS. RECORD CONTINUES TO APPEAR IN THE
- IDENTIFICATION COLUMNS OF THE OUTPUT
- * IN COLUMN 1 INDICATES A COMMENT TO BE Copied TO THE OUTPUT LISTING.
- OTHER IN COLUMN 1 INDICATES PDL RECORD TO BE STRUCTURED BASED ON KEY
- WORDS APPEARING AS FIRST NON-BLANK CHARACTERS (SEE LOGIC).

**OUTPUT**
132 COLUMN LISTING AS FOLLOWS

- 2-6 SEQUENCE NUMBER
- 8-11 STRUCTURE LEVEL NUMBER
- 13-125 STRUCTURED LISTING OF 80 COLUMN INPUT RECORDS
- 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

PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
PDLIST
1 BEGIN PDLIST
2 BEGIN SEQUENCE NUMBER, LEVEL AND DEFINITION TABLE COUNTER
3 DO UNTIL END-OF-FILE INPUT
4 READ RECORD
5 INCREMENT SEQUENCE NUMBER
6 IF COLUMN 1 = " 
7 THEN
8 SET PAGE EJECT IN IMAGE
9 IF REMAINDER OF IMAGE IS NOT BLANK
10 THEN
11 SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
12 ENDIF
13 ELSE
14 CLEAR LEVEL INCREMENT
15 IF COLUMN 1 NOT = " 
16 THEN
17 IF FIRST CHARACTER = :: - INDICATES :LABEL:
18 THEN
19 SET LINE SKIP IN IMAGE
20 GENERATE DEFINITION TABLE ENTRY FOR LABEL
21 ELSE
22 CALL FSTVND TO GET FIRST WORD OF PDL
23 LOOKUP FIRST WORD IN KEY WORD TABLE
24 KEY WORD TABLE CONTAINS
25
26 1 BEGIN - BEGIN SECTION INDICATOR
27 2 IF - SECTION INDICATOR
28 3 DO - SECTION INDICATOR
29 4 DOWFR - SECTION INDICATOR
30 5 DOWTI - SECTION INDICATOR
31 6 DOWMIL - SECTION INDICATOR
32 7 CASE - SECTION INDICATOR
33 8 START - SECTION INDICATOR
34 9 STARTS - SECTION INDICATOR
35 10 ELSE - SECTION SEPARATOR
36 11 THRU - SECTION SEPARATOR
37 12 EXIT - SECTION SEPARATOR
38 13 EXITIF - SECTION SEPARATOR
39 14 OR - SECTION SEPARATOR
40 15 ORELSE - SECTION SEPARATOR
41 16 ENDOLO - SECTION SEPARATOR
42 17 END - END OR END LOOP?
43 18 ENDF - SECTION TERMINATOR
44 19 ENDO - SECTION TERMINATOR
45 20 ENDC - SECTION TERMINATOR
46 21 ENDOSE - SECTION TERMINATOR
47
48 IF KEY WORD LOCATED
49 THEN
50 CASE LOCATION (<BEGIN>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECON>, <SECO

BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
2 SET LEVEL INCREMENT = 1
3 :SECTION: SET LEVEL INCREMENT = 1
80     8       :SEP: SET LEVEL INCREMENT = 1
81     8       DECRENIENT LEVEL
82     8       :END?: CALL FSTWDRD TO GET NEXT WORD OF PDL
83     8       DECREMENI LEVEL
84     8       IF WORD = LOOP
85     9       THEN
86     9       SET LEVEL INCREMENT = 1
87     8       ENDIF
88     8
89     7       :TERM: DECRENENT LEVEL
90     8       ENDCASE
91     6       ENDF
92      5      ENDIF
93      5      COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
94      5      ELSE
95      5      SET INDENTATION FACTOR = 1
96      4      ENDF
97      4      CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
98      4      INPUT RECORD AND IDC FIELD
99      4      OUTPUT IMAGE
100     4      APPLY LEVEL INCREMENT
101     4      CLEAR PAGE EJECT
102     3      ENDF
103     2      ENDOO
104     2      CALL SORT1 TO ORDER DEFINITION TABLE
105     2      OUTPUT DEFINITION TABLE
106     1      END POLIST
116 1 CD1
117 1 CD1
118 1 CD2
119 1 CD2
120 1 CD2
121 1 CD2
122 1 CD2
123 1 CD2
124 1 CD2
125 1 CD2
126 1 CD2
127 1 CD2
128 1 CD2
129 1 CD2
130 1 CD2
131 1 CD2
132 1 CD2
133 1 BEGIN FHSTWRD
134 2 BLANK OUTPUT WORD
135 3 DO UNITAL NON-BLANK CHARACTER
136 3 DO UNTIL SIX CHARACTERS STORED OR END-OF-RECORD
137 3 IF CHARACTER IS NON-BLANK AND NON-
138 3 THEN
139 4 STORE CHARACTER
140 4 ELSE
141 4 EXIT DO
142 3 ENDFHSTWRD
143 2 ENDFHSTWRD
144 2 RETURN LOCATION
145 1 ENDFHSTWRD
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.
<table>
<thead>
<tr>
<th>Line</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>179</td>
<td><strong>XXSMART</strong></td>
<td>SEMI - AUTOMATIC MODE</td>
</tr>
<tr>
<td>180</td>
<td><strong>XXSTO</strong></td>
<td>STORE SEQUENCE TABLE IN ASERTAB</td>
</tr>
<tr>
<td>181</td>
<td><strong>XXTRP</strong></td>
<td>TEMPORARY EXECUTION OF ONE ENTRY WITH BINTAB</td>
</tr>
<tr>
<td>182</td>
<td><strong>XZ</strong></td>
<td>UTILITY PROCESSORS</td>
</tr>
<tr>
<td>184</td>
<td><strong>ASSGN</strong></td>
<td>ASSIGN PROCESSOR</td>
</tr>
<tr>
<td>185</td>
<td><strong>DBDSP</strong></td>
<td>DATA BOX DISPLAY PROCESSOR</td>
</tr>
<tr>
<td>186</td>
<td><strong>DEFIN</strong></td>
<td>DEFINE PROCESSOR</td>
</tr>
<tr>
<td>187</td>
<td><strong>DO</strong></td>
<td>CONDITIONAL LOOP IN SEQUENCE TABLE</td>
</tr>
<tr>
<td>188</td>
<td><strong>ELSE</strong></td>
<td>EXECUTION POINT FOR FALSE IF CONDITION</td>
</tr>
<tr>
<td>189</td>
<td><strong>ENDIF</strong></td>
<td>TERMINATES AN IF STRUCTURE</td>
</tr>
<tr>
<td>190</td>
<td><strong>ENDDO</strong></td>
<td>TERMINATES A DO LOOP STRUCTURE</td>
</tr>
<tr>
<td>191</td>
<td><strong>ENDS</strong></td>
<td>END SCAN PROCESSOR</td>
</tr>
<tr>
<td>192</td>
<td><strong>SCAN</strong></td>
<td>CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES</td>
</tr>
<tr>
<td>193</td>
<td><strong>SCAN</strong></td>
<td>SCAN PROCESSOR</td>
</tr>
<tr>
<td>194</td>
<td><strong>XICHR</strong></td>
<td>CHARACTER OBJECT STORE FOR ASSGN</td>
</tr>
<tr>
<td>196</td>
<td><strong>ZDFT</strong></td>
<td>FIND ANY TOKEN IN A SYMBOLIC STRING</td>
</tr>
<tr>
<td>197</td>
<td><strong>ZDIN</strong></td>
<td>DATA BOX DISPLAY INPUT PROCESSOR</td>
</tr>
<tr>
<td>199</td>
<td><strong>Z zoning</strong></td>
<td>DATA BOX DISPLAY CONSTRAINT MASKER</td>
</tr>
<tr>
<td>200</td>
<td><strong>ZDLOT</strong></td>
<td>DATA BOX DISPLAY OUTPUT ROUTINE</td>
</tr>
<tr>
<td>202</td>
<td><strong>ZDOSIS</strong></td>
<td>DATA BOX DISPLAY PASS 1 PROCESSOR</td>
</tr>
<tr>
<td>201</td>
<td><strong>ZDOPZ</strong></td>
<td>DATA BOX DISPLAY PASS 2 PROCESSOR</td>
</tr>
<tr>
<td>202</td>
<td><strong>ZDOPZ</strong></td>
<td>PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS</td>
</tr>
<tr>
<td>203</td>
<td><strong>ZDOPZ</strong></td>
<td>FIND PROCESSOR CLASS NUMBER</td>
</tr>
<tr>
<td>204</td>
<td><strong>ZDOPZ</strong></td>
<td>FUNCTIONAL OPERATIONS FOR ASSGN</td>
</tr>
<tr>
<td>205</td>
<td><strong>ZDFRE</strong></td>
<td>FREE OBJECT STORE FOR ASSGN</td>
</tr>
<tr>
<td>206</td>
<td><strong>ZDFRD</strong></td>
<td>FIXED OBJECT STORE FOR ASSGN</td>
</tr>
<tr>
<td>207</td>
<td><strong>ZDZSP</strong></td>
<td>REMOVE DUPL. BLANKS &amp; BLANK FILL</td>
</tr>
<tr>
<td>208</td>
<td><strong>ZDJS</strong></td>
<td>SYMBIOIC STRING SYNTAX ERROR LISTER</td>
</tr>
<tr>
<td>209</td>
<td><strong>ZDMSG</strong></td>
<td>FPS PROCESSOR MESSAGE ROUTINE</td>
</tr>
<tr>
<td>210</td>
<td><strong>ZXOPR</strong></td>
<td>MATH OPERATIONS FOR ASSGN</td>
</tr>
<tr>
<td>211</td>
<td><strong>ZXOPES</strong></td>
<td>DATA CONVERSION AND STORAGE FOR ASSGN</td>
</tr>
<tr>
<td>212</td>
<td><strong>ZXOPT</strong></td>
<td>PASS 1 SUBROUTINE FOR ASSGN PROCESSOR</td>
</tr>
<tr>
<td>213</td>
<td><strong>ZXOPT</strong></td>
<td>PASS 2 SUBROUTINE FOR ASSGN PROCESSOR</td>
</tr>
<tr>
<td>214</td>
<td><strong>ZXSSN</strong></td>
<td>SEARCHES SEQUENCE TABLE FOR IF STRUCTURES</td>
</tr>
<tr>
<td>215</td>
<td><strong>ZXSYM</strong></td>
<td>SYMBOL TABLE INTERFACE FOR ASSGN</td>
</tr>
<tr>
<td>216</td>
<td><strong>ZXSYT</strong></td>
<td>SYMBOL TABLE MAINTENANCE</td>
</tr>
</tbody>
</table>
SAVE EAT ADDRESS(IN BREG ON ENTRY)
CALL ENLU(BREG) GET LU IN ASCII & BINARY
STARTSEARCH UNTIL LAST STATUS TABLE ENTRY
EXIT IF STBU EN LU
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STBNG)
IF MANAGER IS DORMANT, THEN
WRITE "***XAO- MANAGER HAS TERMINATED;"
REPLY TO CONTINUE TERMINATION:"
READ(LU) ** WAIT FOR REPLY **
LOCK ON THE FDS TABLE RESOURCE
CALL SLIRN DISABLE
IF STBEX(EXECUTIVE ADDRESS) .NE. 0, THEN
IF STAT(CURRENT) .NE. STBEX, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XMCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STN(MANGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARN OME(P1)
CALL SLIRX ENABLE
CALL MESS 'OFF,BOTTOM'
CALL SLIRX DISABLE
SET BOTTOM TO NEXT-AT
ENDO
ENDIF
ENDIF
CALL SLIRX MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBEX
ENDIF
DECREMENT NUMBER ACTIVE(STBAC)
GET EAT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBER)
CLEAR STBER
CLEAR MANAGER'S ID, STNG, & STBEX
ENABLE.....(VIA A JMP TO EE=DISPATCHER))
RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS
CLEAR LOCK ON FDS TABLE
ELSE ** MANAGER IS STILL ALIVE **
WRITE "***XAO- USER INITIATED INTERRUPT;"
WRITE 'ENTER REQUEST- KILL(X). STATUS(S),OR RETURN(Blank)'
READ (LU) REQUEST
IF REQUEST IS KILL OR X, THEN
PERFORM XAKILL
ELSE
IF REQUEST IS STATUS OR S, THEN
PERFORM XAKAT
ENDIF
ENDIF
ELSE
WRITE '***XAO- ERROR LU IS NOT SIGNED-ON TO FDS'
ENDIF
WRITE '***XAO FDS ATTENTION FUNCTION TERMINATING'
END
PRODUCE A FDS STATUS REPORT

CALL SORDER DISABLE
GET CURRENT-TIME FROM STIME
GET MANAGER'S ADDRESS FROM STMBG
MOVE NAME, STATUS, PARTITION, & PRIORITY
GET EXECUTIVE'S ADDRESS FROM STBEX
MOVE NAME, STATUS, PARTITION, & PRIORITY
GET CURRENT AT FROM STBAT
MOVE NAME, STATUS, PARTITION, & PRIORITY
PERFORM XABTM(CURRENT) FIND BOTTOM AT
SET BOTTOM TO CURRENT
GO WHILE FATHER-ID NE ZERO
SET FATHER-ID FROM CURRENT
CALCULATE NEXT
IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
MOVE NAME, STATUS, PARTITION, & PRIORITY
ENDIF
IF NEXT IS THE MANAGER, THEN
SET CURRENT AS TOP
ENDIF
SET CURRENT TO NEXT
ENDO
IF TOP EQ ZERO, THEN
GOTO TOP TO CURRENT
USE TOP TO MOVE NAME, PARTITION, & PRIORITY
SET STATUS TO 'IN USE' OCTAL 17
ENDIF
CALL SORDER ENABLE
WRITE FIRST SET OF HEADERS
SET TOP AS REPORT DATA
WRITE REPORT LINE
SET MANAGER AS REPORT DATA
WRITE REPORT LINE
SET EXECUTIVE AS REPORT DATA
WRITE REPORT LINE
SET CURRENT AS REPORT DATA
WRITE REPORT LINE
WRITE INTERMEDIATE HEADERS
DO UNTIL MAX ENTRIES OR NO MORE DATA
WRITE REPORT LINE
SET NEXT REPORT DATA
ENDO
END XSTAT
SAMPLE REPORT

FDS STATUS FOR LU 10 HH:MN:SS 360
NAME PRIOR PART# STATUS
TOP AT- PROC 922 3 GENERAL WAIT
MANAGER- XGNN 40 3 GENERAL WAIT
EXECUTIVE- XEXN 50 3 GENERAL WAIT
CURRENT AT- PROC 11311 4 GENERAL WAIT
BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
PROC 32767 6 DISK ALLOCATE SUSPEND
PROC 11311 4 GENERAL WAIT
PROC 2040D 3 GENERAL WAIT
PROC 845 6 GENERAL WAIT
140 1 BEGIN XAKILL
141 2 * TERMINATE CURRENT FDS FUNCTION
142 3 * IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
143 4 * ON NEXT RETURN VIA A PAW.
144 5 * IF THE EXEC IS ACTIVE-DO NOTHING
145 6 * IF A PROCESSOR IS ACTIVE- USE RTE MESS TO OFF THE PROCESSOR
146 7 * GET MANAGER'S ID ADDRESS(STRM)
147 8 * IF STATUS OF MANAGER IS NOT WAIT, THEN
148 9 * SET TERMINATE FLAG IN STR-ENTRY
150 3 ELSE
151 3 * IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
152 4 * WRITE "***XAO5 FDS MANAGER SIGNAL TO TERMINATE SEQUENCE'
153 3 ELSE
154 4 PERFORM XAHM(CURRENT) FIND BOTTOM AT
155 4 IF BOTTOM AT IS D.RTR OR SNP THEN
156 4 WRITE "**XAO5 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 "**XAO5 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN.'
162 6 EXIT XAKIL
163 6 ELSE
164 7 SET RETURN PARAMETER TO PROCESSOR ABENDED
165 7 INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
166 7 CALL BLIST TO REACTIVATE MANAGER
167 6 ENDIF
168 5 ENDIF
169 5 WRITE "**XAO7 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT.
170 5 IF RETURNED BOTTOM WAS NOT MANAGER THEN
171 6 SET NAME IN 'OFF' COMMAND
172 6 CALL MESS TO 'OFF' THE PROCESSOR
173 5 ENDIF
174 4 ENDIF
175 3 ENDIF
176 2 ENDIF
177 1 END XAKILL
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
        XATTM
179 1 BEGIN XABTM FIND BOTTOM AT
180 2 DO WHILE CURRENT IS IN GENERAL WAIT,
181 3 AND WAIT POINTER(P) HAS A SON ADDRESS,
182 4 AND SONS FATHER ID POINTS TO CURRENT
183 5 SET SON AS CURRENT
184 6 ENDPO
185 7 SET BOTTOM AS CURRENT
186 8 END XABTM
1 BEGIX XCONFG
2 a
3 *D1 FDS CONFIGURATION MANAGER
4 *D1 INITIATES AN FDS SYSTEM FOR A REQUESTED TERMINAL OR
5 *D1 TERMINATES AN FDS SYSTEM FOR A REQUESTED TERMINAL
6 *D1 INITIATED VIA
7 *D1 RU,FDS,LU,ID,NAME SIZE,PARM,OPTIONS
8 *D2 INPUTS
9 *D2 LOGICAL UNIT(LU) FOR THE REQUESTED TERMINAL,
10 *D2 A PARM TO DENOTE INITIATION OR TERMINATION
11 *D2 A ONE CHARACTER USERID
12 *D2 A DEBUG OPTION INDICATOR
13 *D2 THE NUMBER OF DNA TRACKS
14 *D3 OUTPUTS
15 *D3 INITIATION-
16 *D3 A BLANK ID-SEGMENT WILL BE CONSTRUCTED FOR THE FDS MANAGER,
17 *D3 AND FDS EXECUTIVE
18 *D3 THE ENT FOR THE REQUESTED LU WILL BE CONNECTED
19 *D3 TO THE FDS ATTENTION TASK
20 *D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
21 *D3 WILL BE UPDATED TO REFLECT THE INITIATION
22 *D3 TERMINATION-
23 *D3 THE FD SEGMENTS WILL BE RETURNED TO BLANK STATUS
24 *D3 THE ENT FOR THE LU WILL BE REINSTATED
25 *D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
26 *D3 WILL BE UPDATED FOR THE TERMINATION
27
1 BEGIN XCON
2 * SIGN ON A USER TO FDS
3  DO UNTIL VALID USER ID (P3)
4    IF ID NOT A - 2 THEN
5      WRITE 'XCON ENTER VALID ID (A - 2)' PATH RESPONSE
6    ENDIF
7  ENDDO
8  IF USER ID IS BEING USED, THEN
9    WRITE '***XCO5 LU 'LU' IS CURRENTLY USING ID 'ID'- SIGN ON REJECTED'
10  EXIT :XCETA
11  ENDIF
12  IF FDS RESOURCE NUMBER NOT DEFINED, THEN
13    CALL RMAQ (GLOBAL ALLOCATE, LOCAL SET)
14  ELSE
15    CALL RMAQ (LOCAL SET)
16  ENDDIF
17  IF NUMBER SIGNED DM (STBNK) .EQ. MAXIMUM USERS (STBNK), THEN
18    ISSUE MESSAGE '***XCO5 FDS CURRENTLY AT MAX USER'S.
19  ELSE
20    DO FOR STBNK (NUMBER OF FDS ENTRIES)
21      IF ENTRY'S LUCSTBU .EQ. REQUESTING LU(P3) THEN
22        ISSUE MESSAGE '***XCO5 'LU' IS CURRENTLY SIGN ON TO FDS'
23      EXIT :XCETA
24    ELSE
25      IF THIS ENTRY IS AVAILABLE, THEN
26        SET AS CURRENT-ENTRY-ADDRESS
27      ENDFL
28    ENDDO
29    BECOME PRIVILEGED & DISABLED
30    CALL SLUR
31    START SEARCH WHILE NUMBER-FOUND .LT. NUMBER-NEEDED
32    SEARCH ID-SEGMENTS USING KEYW((1657)
33      IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
34        SET ID ADDRESS OF XEXEC
35      INCRCENT NUMBER-FOUND
36    ELSE
37      IF XNGR NOT FOUND AND THIS ID .EQ. XNGR, THEN
38        SET ID ADDRESS OF XNGR
39      INCRNENT NUMBER-FOUND
40    ELSE
41      IF XATH NOT FOUND AND THIS ID .EQ. XATH, THEN
42        SET ID ADDRESS OF XATH
43      INCRNENT NUMBER-FOUND
44    ELSE
45      IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
46        SET ID ADDRESS OF FIRST-BLANK
47      INCRNENT NUMBER-FOUND
48    ELSE
49      IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
50        SET ID ADDRESS OF SECOND-BLANK
51      INCRNENT COUNT
52    ENDFL
53    ENDFL
54    ENDFL
55  EXIT THERE ARE NO MORE IDS
CALL BLRX    ENABLE
ISSUE MESSAGES "**XCOI / cannot find 'NAME' ID-SIGNON TERMINATED"
ENDLOOP
BUILD FTRY IN XSTD
*    SET LU IN'S STBLU
SET LU IN ASCII INTO STBLA
SET USER'S ID INTO STBD
SET ADDRESS OF FIRST-BLANK INTO STBE
SET ADDRESS OF SECOND-BLANK INTO STBX
INCREMENT ACTIVE COUNT(STBAC)
BUILD XPHGN & XEXHN
MOVE PRIORITY THRU DISC ADDRESS FROM XMSR TO FIRST-BLANK
TURN ON TN BIT
SET NAME TO XPHGN
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
TURN ON TN BIT
SET NAME TO XEXHN
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE ERT OVERLAY
SAVE ERT VALUE IN STBD
SET ID ADDRESS OF XATTN INTO ERT
ENDIF
SET INPUT PARTS INTO ID OF XPHGN
SCHEDULE XPHGN VIA BLIST
CALL BLIST
CALL BLRX    ENABLE
IF FDS HAS A FATHER, THEN
CALL MESS 'OFF,FATHER'
ENDIF
ISSUE MESSAGE "**XCOI LU "MN" SIGNED ON TO FDS"
ENDSEARCH
:XCETA
CALL RMNR (LOCAL CLEAR)
ENDIF
CALL EXEC TERMINATE
END XCON
148 1 DEF XCOFF
149 2 * SIGN OFF A USER 1/ FDS
150 3 " * WRITE 'FDS SIGN OFF FUNCTION NOT SUPPORTED'
152 1 END XCOFF
FORTRAN CALLING PROCEDURE
CALL XELDS (XOCLD)

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

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

INTERNAL VARIABLES
LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

NOTES
USES .ENTR, XDLST, XERTM
XOCLD IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
CONTAINING THE REFERENCED DIRECTIVES

BEGIN XOCLD
EXTRACT DIRECTIVE INDEX FROM MASSTA
CASE (LIST:) INDEX
LIST: CALL XDLST
ENDCASE
CALL XERTM TO RETURN FROM SEGMENT
END XOCLD
FORTAN CALLING PROCEDURE

**XDCLU** DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS

- STORE
- RESTORE
- UNLOAD
- LOAD
- BATCH

**INPUT**

- XE COMMON - MASSTA (BITS 10-13 CONTAIN A 9-13 INDEX INTO A LIST OF
  DIRECTIVES)

**INTERNAL VARIABLES**

- LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

**NOTES**

- USES .ENTR, XDSTO, XDRES, XDUNL, XDLOA, XDBAT, XERTH

**XDCLU IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT

**CONTAINING THE REFERENCED DIRECTIVES**

1 BEGIN XDCLU
2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7
3 CASE (:STOR; :REST; :UNLO; :LOAD; :BATCH) INDEX

:STOR: CALL XDSTO
:REST: CALL XDRES
:UNLO: CALL XDUNL
:LOAD: CALL XDLOA
:BATCH: CALL XDBAT

ENDCASE
2 CALL XERTH TO RETURN FROM SEGMENT
1 END XDCLU
138  1 C00  FORTRAN CALLING PROCEDURE
139   1 C00
140   1 C00  CALL XDCLE
141   1 C00
142   1 C00  C********
143   1 C01  XDCLE PURGES ALL DATA FROM THE AWA. THE ONLY ELEMENTS REMAINING ARE PERMANENT SYSTEM TABLES AND DATA BASE FILES (CHAINS 1 & 8).
144   1 C01
145   1 C01  C********
146   1 C02  INPUT
147   1 C02  XE COMMON - CARTAG, FLAG, LU
148   1 C02  MANAGER - ANA HEADER AND TOC (SEE XMAMA)
149   1 C02
150   1 C02  C********
151   1 C03  OUTPUT
152   1 C03  XE COMMON - REBUF, REOPTR, COMBUF-SCRATCH
153   1 C03  XB COMMON - SCRATCH BEYOND X8(200)
154   1 C03  XS COMMON - SCRATCH
155   1 C03  MANAGER - COMMANDS TO PURGE AWA AND RESTORE CHAINS 0 AND 8
156   1 C03
157   1 C03  C********
158   1 C04  LOCAL VARIABLES
159   1 C04  AWA - ANA HEADER AND TOC RECEIVED FROM MANAGER
160   1 C04  HD1 - TOC CHAIN 1 HEAD
161   1 C04  HD2 - TOC CHAIN 2 HEAD
162   1 C04  HD3 - TOC CHAIN 3 HEAD
163   1 C04  HD4 - TOC CHAIN 4 HEAD
164   1 C04  NEXT - INDEX TO NEXT ALLOCATABLE EIGHT WED ENTRY IN QUEUE
165   1 C04  NMAX - MAXIMUM SIZE OF QUEUE
166   1 C04  ORG - BASE REFERENCE ADDRESS FOR AWA ADDRESSES
167   1 C04  QUEUE - BUFFER FOR CONSTRUCTING AWA RESTORE REQUESTS FOR NON-DELETED ITEMS
168   1 C04  TOCMAX - MAXIMUM SIZE OF TOC ACCOMODABLE BY XDCLE
169   1 C04
170   1 C04  C********
171   1 C05  NOTES
172   1 C05  USES EXEC, IAND, PRTN, PURGE, XREQ, XRIG, XRMOS, XR6G, XR6H, XR6M, XVRG, XVAB
173   1 C05
174   1 C05  C********
175   1 C05  IN THE EVENT THAT A COMPACTED AWA TOC WILL NOT FIT IN THE ALLOCATABLE BUFFER SPACE AND ELEMENTS OF CHAIN 1 OR 8 EXTEND BEYOND IT, THE CLEAR FUNCTION WILL NOT BE PERFORMED.
176   1 C05
177   1 C05  C********
178   1 C05  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.
1 BEGIN XDCL
  2 RETRIEVE TOC
  3 BUILD REQUEST TO CLEAR AW
  4 DO UNTIL END OF PERMANENT SYSTEM TABLES CHAIN (CHAIN 1)
  5 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
  6 BUILD REQUEST TO REALLOCATE TABLE
  7 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
  8 BUILD REQUEST TO STORE TABLE INTO AW FROM SAM
  9 ENDDO
 10 DO UNTIL END OF DATA BASE FILES CHAIN (CHAIN 8)
 11 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
 12 BUILD REQUEST TO REALLOCATE TOC ENTRY
 13 ENDDO
 14 BUILD REQUEST TO TERMINATE LIST
 15 DO UNTIL END OF DRDE CHAIN (CHAIN 3)
 16 THEN
 17 OUTPUT XD13 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
 18 EXIT PURGE LOOP
 19 ENDF
 20 ENDF
 21 ENDO
 22 DO FOR EACH BLOCK OF EIGHT REQUESTS
 23 TRANSIT BLOCK TO MANAGER
 24 EXIT TO :ERR23: IF REQUESTS FAILED
 25 ENDDO
 26 EXIT XDCL
 27 :ERR23:
 28 DO FROM FAILING REQUEST TO END OF LIST
 29 IF REQUEST TO STORE
 30 THEN
 31 READ SAM TO FREE BUFFER AND CLASS NUMBER
 32 ENDF
 33 ENDDO
 34 DO UNTIL END OF DATA BASE FILE CHAIN (CHAIN 8)
 35 IF FILE IS UTD (TYPE 1)
 36 THEN
 37 CALL PURGE TO DELETE FILE
 38 ENDF
 39 ENDDO
 40 PURGE ALL UTD FILES
 41 TERMINATE FDS WITH CLEAR FAILURE MESSAGE
 42 :ERR48:
 43 DO FOR ALL STORE REQUESTS BUILT
 44 READ SAM TO FREE BUFFER AND CLASS NUMBER
 45 ENDDO
 46 EXIT XDCL WITH CLEAR FAILURE MESSAGE
 47 END XDCL
**Fortran Calling Procedure**

**CALL XDCOP**

**XCOP PROCESS A USER REQUEST TO COPY A SEQUENCE TABLE, INTERFACE TABLE, DATA ELEMENT, OR FOR. ONLY THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S WA CAN BE COPIED.**

**COMMON XE - CARTAG, COMBUF, QUAL, REQPTR, TOKENS**

**COMMON XE - COMPRTR, RERBUF**

**INTERNAL VARIABLES**

**COMMON XE - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE) (2) DBTYPE: TYPE CODE FOR DATA BASE FILES (3) I: INDEX (4) IDC: EXEC BUFFER (148) IERR: FILE MANAGER ERROR RETURN (149) IMSG: ERROR MESSAGE NUMBERS (150) NNAME: NEW FMGR FILE NAME (153) WNAME: NEW FMGR FILE NAME (154) OFNAME: OLD FMGR FILE NAME (157) XNAME: OLD NAME IN COBMAIN (158) TNCENT: TOC ENTRY FOR DATA BASE RENAME (166) EOF: EOF RETURN FROM FMGR**

**COMMON XB - (204) IBUFF: USER BUFFER FOR EXEC I/O (1024)**

**Routines Used -**

CLOSE, CREAT, EXEC, OPEN, PURGE, READF, WRITF, XDOBA, XDOOD, XREQ, XRMV, XRNSG, XRFP, XRIS
1 BEGIN XDCOP
2 1 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
3 2 SAVE POINTER TO CURRENT NAME
4 3 IF NEXT TOKEN IS HYPHEN, THEN
5 4 DECODE CLASS NAME
6 5 EXIT TO :CLASS: IF CLASS SPECIFIED IS NOT VALID (S,I,O,F,B)
7 6 ELSE
8 7 SET CLASS TO DATA ELEMENT
9 8 ENDIF
10 1 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
11 2 SAVE POINTER TO NEW NAME
12 3 EXIT TO :SYNTAX: IF NEXT TOKEN IS END-OF-MESSAGE
13 4 IF CLASS IS DATA BASE OR ORDE, THEN
14 5 EXIT IF NEW NAME IS MORE THAN 4 CHARACTERS :NAMERR:
15 6 IF CLASS IS DATA BASE, THEN
16 7 EXIT TO :NAMERR: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
17 8 ENDIF
18 9 BUILD AUA MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
19 10 BUILD AUA MANAGER REQUEST TO VERIFY NEW NAME
20 11 CALL XRN
21 2 EXIT TO :TSCERR: IF RETURN INDICATES ERROR ON FIRST REQUEST
22 3 EXIT TO :TSCERR: IF NO ERROR RETURNED ON SECOND REQUEST
23 4 CALL EXEC TO GET TOC ENTRY
24 5 IF CLASS IS DATA BASE THEN
25 6 IF TYPE IS POB, THEN
26 7 CALL XBDBA TO ADD NEW POB TO LOG FILE (XPDB)
27 8 EXIT TO :FILERR: IF FMGR ERROR RETURNED
28 9 EXIT TO :TSCERR: IF NEW NAME IS DUPLICATE
29 10 EXIT TO :MAXERR: IF POB MAX IS EXCEEDED
30 11 SET FILE PREFIX TO RIGHT BRACKET
31 12 ELSE
32 13 EXIT TO :INVLD: IF FILE IS NOB
33 14 SET FILE PREFIX TO *
34 15 ENDIF
35 16 SET FMGR FILE TYPE TO 1
36 17 SET SECURITY CODE TO 55
38 3 ELSE
39 4 SET FILE PREFIX TO /
40 5 SET SECURITY CODE = 0
41 6 SET FMGR FILE TYPE FROM TOC ENTRY
42 7 IF FILE IS TYPE 2, THEN
43 8 STORE RECORD LENGTH FROM TOC ENTRY
44 9 ENDIF
45 10 CALL XRN TO FORMAT NEW FILE NAME
46 11 CALL CREATE FOR NEW FILE
47 12 EXIT TO :FILERR: IF FMGR ERROR RETURNED
48 13 CALL XRN TO FORMAT OLD FILE NAME
49 14 CALL OPEN FOR OLD FILE
50 15 EXIT TO :FILERR: IF FMGR ERROR RETURNED
51 16 EXIT TO :TYPERR: IF TYPE IS NOT SAME AS IN TOC
52 17 DO UNTIL END-OF-FILE IS READ ON OLD FILE
53 18 CALL READ TO READ RECORD FROM OLD FILE
54 19 EXIT IF ERROR DETECTED TO :FILERR:
55 20 CALL WRITE TO WRITE RECORD TO NEW FILE
56 21 EXIT IF ERROR DETECTED TO :FILERR:
57 22 END DO
58 23 CALL CLOSE FOR OLD FILE
59 24 CALL CLOSE FOR NEW FILE
60 25 END EXEC
61 26 END XDCOP
BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY
EXIT TO :TODES: IF ERROR IS INDICATED
ELSE
BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
BUILD MANAGER REQUEST FOR DATA RETRIEVAL
CALL XRMS
EXIT TO :TODES: 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 XRMS
EXIT TO :TODES: IF ERROR IS INDICATED
ENDIF
EXIT XRMS
376 2 :SYNTAX: CALL XRMS "SYNTAX ERROR ..." AND EXIT
377 2 :CLASS: CALL XRMS "INVALID CLASS DESIGNATOR ..." AND EXIT
378 2 :NAMEERR: CALL XRMS "NEW NAME IS INVALID ..." AND EXIT
379 2 :MAXERR: CALL XRMS "AUTHORIZED LIMIT ..." AND EXIT
380 2 :INVAL: CALL XRMS "NOB CANNOT BE ..." AND EXIT
381 2 :FILEERR: CALL XRMS "FILE ACCESS ERROR ... ON ......." AND EXIT TO :END:
382 2 :TODES: CALL XRMS TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
383 2 :TYPEERR: CALL XRMS "INCONSISTENT FILE TYPE ..."
384 2 :END:
385 2/2 IF PDB HAS BEEN LOGGED IN XPDG, THEN
386 3 CALL XDROD TO DELETE PDB FROM XPDG
387 2/2 ENDIF
388 2/2 IF A NEW FILE HAS BEEN BUILT, THEN
389 3 PURGE NEW FILE
390 3 CLOSE OLD FILE
391 2/2 ENDIF
392 1 END XRMS
FORTRAN CALLING PROCEDURES

CALL XDDBA(NAME, IERR, ISIZE)
CALL XDDBV(NAME, IERR, ISIZE)
CALL XDDBD(NAME, IERR)

********

XDDBA, XDBBV, AND XDDBD 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
XDDBA ONLY)

********

OUTPUT
IERR - INTEGER WORD CONTAINING RETURN CODE
0 NO ERROR
1 DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE
2 MDB/PDB LIMIT EXCEEDED
ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
XDBBV ONLY)

********

COMMON USED
XE - REAL
CARTB - CHAR
XS - DCO (WORDS 1-144)

********

NOTES
XRLK, XRLCK, .ERNG, OPEN, CLOSE, READF, AND WRITF ARE USED
WHEN WORKING WITH MDB'S, QUAL SHOULD BE SET TO 77 OCTAL
445 1 BEGIN XDBA
446 2 STORE RETURN ADDRESS
447 3 CALL :ENTR TO SET UP CALLING ARGUMENTS
448 2 CALL XLCK FOR EXCLUSIVE USE OF XPD
449 2 CALL OPEN FOR EXCLUSIVE USE OF XPD
450 1 EXIT TO :FILER: IF ERROR RETURNED
451 2 COMPUTE RECORD NUMBER FOR USER'S DIRECTORY = (QUAL-776)/2+1
452 2 CALL READ FOR RECORD COMPUTED
453 1 EXIT TO :FILER: IF ERROR RETURNED
454 2 DETERMINE PART OF RECORD TO BE USED
455 2 IF REQUEST IS FOR ADD, THEN
456 2 EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
457 2 START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
458 2 EXIT TO :NAMERR: IF ENTRY NAME MATCHES PARAMETER NAME
459 3 ENDL
460 3 ENDS
461 3 STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
462 3 INCREMENT # OF CURRENT ENTRIES
463 2 ELSE
464 2 START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
465 2 EXIT IF ENTRY NAME MATCHES PARAMETER NAME
466 3 ENDL
467 3 EXIT TO :NAMERR:
468 3 ENDS
469 3 IF REQUEST IS FOR VERIFY, THEN
470 4 STORE WORD 3 OF ENTRY IN ISSIE
471 3 ELSE
472 4 REPLACE ENTRY WITH LAST ENTRY
473 4 STORE ZEROES IN LAST ENTRY
474 4 DECREMENT # OF CURRENT ENTRIES
475 3 ENDF
476 2 ENDF
477 2 CALL WRITE TO WRITE RECORD TO XPD
478 1 EXIT TO :RETURN:
479 2 :NAMERR: SET IERR = 1 AND EXIT TO :RETURN:
480 2 :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
481 2 :FILERR: SET IERR = FMGR ERROR CODE
482 2 :RETURN: CALL CLOSE FOR XPD
483 2 CALL XLCK TO RETURN RESOURCE #
484 1 END XDBA
FORTRAN CALLING PROCEDURE

CALL XDELE

XDELE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT
SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE AWA.
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 - COMMDF, COMPRE, LU, QUAL, TOKENS

OUTPUT
COMMON XE - REGHDF

NOTES

ROUTINES USED
EXEC IAND PURGE XDDDB
XREG XREX XXIG
XHIS XRPCK XHWP
XHUK XHOG
1 BEGIN XDELE
2 DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED
3 EREREIf COMMA IS NOT NEXT LEXICAL ELEMENT :ERROR:
4 IF CLASS DESIGNATOR IS SPECIFIED, THEN
5 SET REQUESTED CLASS APPROPRIATELY (O, S, I, B, OR F)
6 ELSE
7 SET REQUESTED CLASS TO BE (O)
8 ENDIF
9 IF DATA BASE TO BE DELETED, THEN
10 BUILD AND ISSUE DATA MANAGER REQUEST FOR TOC ENTRY
11 ENDIF
12 IF ELEMENT IS NOT A MASTER DATA BASE, THEN
13 IF ELEMENT IS A PERSONAL DATA BASE, THEN
14 CALL FROEB TO DELETE THIS PDB FROM XPDO
15 IF ERROR IS RETURNED, THEN
16 CALL XRMG - "FILE ACCESS ERROR ??? XPDO"
17 SET ERROR FLAG
18 ENDF
19 ENDIF
20 BUILD AND ISSUE DATA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
21 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
22 CALL XRMG - "XXXXX NOT FOUND"
23 SET ERROR FLAG
24 ENDF
25 IF CLASS IS DATA BASE (C), OR
26 IF ERROR FLAG IS NOT SET, THEN
27 ISSUE RTE PURGE FOR THE FILE
28 IF RETURN CODE FROM PURGE, THEN
29 CALL XRMG - "FILER ERROR WWW XXXXX"
30 ENDF
31 ENDIF
32 IF CLASS IS IRDE (F), THEN
33 CALL XRMG - "... IS A MIB. NOT DELETED."
34 CALL XRMG - "SYNTAX ERROR"
35 ENDIF
36 ENDOD
37 1 EXIT TO :RETURN:
38 :ERROR: CALL XRMG - "SYNTAX ERROR"
39 :RETURN:
40 1 END XDELE
CALL XMIS (TOCLST)

XMISS PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECODING EACH ELEMENT AND FLAGGING IT IF TOCLST AS AN ELEMENT
TO BE STORED/RESTORED.

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XB(151) ABLG, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

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

OUTPUTS IN COMMON:

XB(151) ABLG, XB(157) TOTSEQ, XB(158) TOTWD

INTERNAL XB COMMON USED:

XB(151) ABLG - (INTEGER, 1 WORD) ABORT FLAG
XB(152) EBLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) WENO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TOTSEQ - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
XB(147) BE STORED/RESTORED
XB(158) TOTWD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
XB(198) RETC - (INTEGER, 1 WORD) XCOM RETURN CODE
XB(199) BATLCS - (INTEGER, 1 WORD) CLASS OF DATA BEING SEARCHED
XB(200) NAMTP - (INTEGER, 1 WORD) POINTER TO NAME IN COMBUF
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCLST

COMMON USED:

EQUIVALENCE 
* (RE(5), MASKTA), (RE(85), EOS :
* (RE(85), NAME :), (RE(92), HYPN )
* (RE(113), COMMA :), (RE(144), COMTR)
* (RE(145), COMBUF)

...
1 BEGIN XDLIS
2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
3 TURN ERFLG OFF
4 :RTN1:
5 DO UNTIL EOS IS SENSED IN COMBUS
6 ERREXIT IF TOKEN IS NOT "NAME" TO :ERR1:
7 SAVE INDEX TO NAME FIELD
8 INCREMENT TO NEXT TOKEN
9 IF TOKEN IS A HYPHEN THEN
10 ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:
11 INCREMENT TO NEXT TOKEN
12 DECODE CLASS NAME (I, S, D, F)
13 ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:
14 SET CLASS TO CLASS SPECIFIED
15 ELSE
16 ENDIF
17 IF XDLIS CALLED FROM STORE THEN
18 ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:
19 ENDIF
20 ERREXIT IF NAME/CCLASS ENTRY NOT FOUND IN TIC TO :ERR2:
21 CALL ARSET TO TURN STORE/RESTORE BIT ON
22 INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
23 ENDDO
24 :RTN2:
25 IF ERROR FLAG IS ON THEN
26 CALL XCOM TO REPROPRINT USER TO CONTINUE
27 ERREXIT IF RESPONSE IS X TO :ERR3:
28 ENDDO
29 1 EXIT XDLIS
30 :ERR1:
31 SET ERROR FLAG ON
32 CALL XRMSG TO DISPLAY SYNTAX ERROR
33 GO TO :RTN2:
34 :ERR2:
35 IF ERROR FLAG IS OFF THEN
36 TURN ERROR FLAG ON
37 CALL XRMSG TO DISPLAY NOT STORED/RESTORED MESSAGE
38 ENDIF
39 CALL EXEC TO DISPLAY ELEMENT NAME
40 GO TO :RTN1:
41 :ERR3:
42 SET ABFLG TO ABORT STORE/RESTORE OPERATION
43 1 END XDLIS
FORTRAN CALLING PROEDURE

CALL XDLST

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

INPUT

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

OUTPUT

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

INTERNAL VARIABLES

COMMON XE - BUFFER = PRINT LINE TO BE OUTPUT

BUFFPR = INDEX INTO BUFFER FOR NEXT ASCII DATA

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

USES ROUTINES - EXEC, XEINT, XILSD, XILST,

XEXEC, XREXIT, XRMV, XRMGS,

XPCK, XUPK, XILST, OPEN,

CLOSE, READY

NOTES
BEGIN XDLST

IF DEVICE ID FIELD SPECIFIED, THEN

SET LU FOR LISTING AS INDICATED ON INEECTIVE

ELSE

SET LU FOR LISTING TO BE TERMINAL LU

ENDIF

DO UNTIL END-OF-STATEMENT IS REACHED

ERREXIT IF COMMA IS NOT SPECIFIED :ERR9:

ERREXIT IF A NAME DOES NOT FOLLOW THE COMMA :ERR9:

RETAIN NAME FOR XREM CALL

IF A CLASS DESIGNATOR IS SPECIFIED, THEN

SET CLASS (1, S, OR D) FOR XREM CALL

ELSE

USE DATA (D) CLASS IN XREM CALL

ENDIF

CALL XREM TO RETRIEVE THIS TABLE OR DATA ELEMENT

ERREXIT IF NOT FOUND :ERR0:

ERREXIT IF APA SPACE NOT AVAILABLE FOR TABLE IN DMA :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 XINT TO INITIALIZE LITERAL ENTRIES

INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE ITS LIST RTN.

INITIALIZE 'LSTFLG' TO INDICATE ENTIRE TABLE TO BE LISTED

CALL XLIST TO LIST THE INTERFACE TABLE

ELSE

IF SEQUENCE TABLE TO BE LISTED, THEN

CALL XMOV TO MOVE TABLE TO TOP OF WORKING BUFFER

CALL XLIST TO LIST SEQUENCE TABLE

ELSE

INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE ITS LIST RTN.

SET 'LSTFLG' TO INDICATE ONLY 1 DATA ELEMENT BEING LISTED

INITIALIZE PRINT BUFFER WITH NAME OF ELEMENT

CALL XLIST TO LIST THE DATA

ENDIF

ENDIF

INCREMENT TO NEXT TOKEN IN OPERAND LIST OF THE DIRECTIVE IMAGE

:ERR10: CALL XRMGR -- '... NOT FOUND'

:ERR11: CALL XRMGR -- '... CANNOT BE MOVE FROM DMA TO APA'

ENDDO

EXIT TO :RETURN:

:ERROR: CALL XRMGR -- 'SYNTAX ERROR'

:RETURN:

END XDLST
783  1 C*******
784  1 CD0          FORTRAN CALLING PROCEDURE
785  1 CD0          CALL XDOFF
786  1 CD0          C*******
787  1 CD1          XDOFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
788  1 CD1          DELETES ALL DRDE AND UTDB FILES LOGGED IN THE ANA,
789  1 CD1          PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
790  1 CD1          RETURNS NORMAL PATHS TO THE FDS MANAGER AND TERMINATES
791  1 CD1          NORMALLY VIA RTE.
792  1 CD1          INPUT
793  1 CD2          COMMON XE = LU, FLAGS, QVAL, RERBUF
794  1 CD2          CD2          COMMON XB - ORG = ORIGIN ADDRESS OF ANA. USED TO CALCULATE
795  1 CD2          CD2          INDICES INTO 'ANA' FROM ADDRESS POINTERS
796  1 CD2          CD2          OF TOC ENTRIES
797  1 CD2          CD2          ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND
798  1 CD2          CD2          TOC RETRIEVED VIA XREQ
799  1 CD2          CD2          C*******
800  1 CD4          INTERNAL VARIABLES
801  1 CD4          CD4          COMMON XS = POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC
802  1 CD4          CD4          ENTRY. MOST SIGNIFICANT (BIT 15) BIT
803  1 CD4          CD4          SET TO INDICATE END-OF-CHAIN
804  1 CD4          CD4          INDEX = VALUE COMPUTED FROM POINTER TO BE
805  1 CD4          CD4          FORTRAN INDEX INTO 'ANA' FOR NEXT TOC
806  1 CD4          CD4          ENTRY
807  1 CD4          CD4          C*******
808  1 CD5          NOTES
809  1 CD5          ROUTINE USED - EXEC, PURGE, XDST, XPST, XREQ, XREST, XR16,
810  1 CD5          XRMOV, XRHS, XRFS, XRPCX, XRSET, XRPK, XICOM,
811  1 CD5          XURDG, XVABN
812  1 CD5          XDOFF
813  1 CD5          XDOFF
814  1 CD5          XDOFF
815  1 CD5          XDOFF
816  1 CD5          XDOFF
817  1 CD5          XDOFF
818  1 CD5          XDOFF
819  1 CD5          XDOFF
820  1 CD5          XDOFF
821  1 CD5          XDOFF
822  1 CD5          XDOFF
823  1 CD5          XDOFF
824  1 CD5          XDOFF
825  1 CD5          XDOFF
826  1 CD5          XDOFF
827  1 CD5          XDOFF
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL THEM TO REQUEST TOC AND CHAIN HEADS
5 CALL EXEC TO READ IN TOC AND CHAIN HEADS
6 IF CHAIN HEAD FOR DRE FILE IS NOT NEGATIVE THEN
7 DO UNTIL DRE CHAIN HEAD IS NEGATIVE
8 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
9 OUTPUT 'XDI3 TOC TOO LARGE, PURGE INCOMPLETE'
10 EXIT DO
11 ENDIF
12 CALL XEOFN TO CREATE FILE NAME 'HXXXX'
13 CALL PURGE TO SCRATCH FILE
14 SET DRE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
15 ENDDO
16 ENDIF
17 IF CHAIN HEAD FOR DATA BASES IS NOT NEGATIVE THEN
18 DO UNTIL DATA BASE CHAIN HEAD IS NEGATIVE
19 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
20 OUTPUT 'XDI3 TOC TOO LARGE, PURGE INCOMPLETE'
21 EXIT DO
22 ENDIF
23 IF TYPE OF DATA BASE IS UDB, THEN
24 CALL XEOFN TO CREATE FILE NAME 'HXXXX'
25 CALL PURGE TO SCRATCH FILE
26 ENDIF
27 SET DATA BASE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
28 ENDDO
29 ENDIF
30 CALL XDATA TO OUTPUT USAGE STATISTICS
31 IF USER REQUESTED DEBUG SNAP THEN
32 CALL XUDBG
33 ENDIF
34 IF USER REQUESTED ABEND DUMP THEN
35 CALL XVADB - NO RETURN FROM THIS CALL
36 ENDIF
37 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
38 CALL XPXIT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE EXEC
39 ENJIF
40 RETURN
41 END XDOFF
071 1 C0**************
072 1 C00 FORTRAN CALLING SEQUENCE:
073 1 C00 CALL XRDE (DATBUF, DDBCB)
074 1 C00**************
075 1 C01 XRDE HANDLES THE RESTORING OF DATA ELEMENTS, SEQUENCE TABLES
076 1 C01 AND INTERFACE TABLES
077 1 C01**************
078 1 C02 INPUTS FROM CALLING SEQUENCE:
079 1 C02 DATBUF - (INTEGER, 1480 WORDS) BUFFER USED TO READ IN RECORDS
080 1 C02 OF DATA FROM DATA BASE FILE
081 1 C02 DDBCB - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DDB
082 1 C02**************
083 1 C03 OUTPUTS IN CALLING SEQUENCE:
084 1 C03 DATBUF, DDBCB
085 1 C03**************
086 1 C03 OUTPUTS IN XB COMMON:
087 1 C03 XB(151) ADFLG, XB(152) ERFLG, XB(165) TOCHDS,
088 1 C03 XB(169) TOPLPL, XB(170) ENDBLK, XB(171) TOCHD,
089 1 C03 XB(201) NOTOC
090 1 C03**************
091 1 C04 INTERNAL XB COMMON USED:
092 1 C04 XB(157) ADFLG - ABORT FLAG
093 1 C04 XB(152) ERFLG - ERROR MESSAGE FLAG
094 1 C04 XB(154) ADFLG - RESTORE ALL UDBF FLAG
095 1 C04 XB(155) DEBUG - DEBUG FLAG
096 1 C04 XB(159) FNFLG - DB FILE NAME
097 1 C04 XB(162) DATBFLK - BLOCK # WHERE DATA ITEM BEGINS
098 1 C04 XB(163) DATINDX - WORD INDEX INTO DATBUF WHERE DATA BEGINS
099 1 C04 XB(164) IERR - ERROR FLAG FOR FMGR CALLS
100 1 C04 XB(165) TOCHDS - WORDS OF TOC IN TOCBUF (NOTOC *8) + 8
101 1 C04 XB(166) NOBLKS - # BLOCKS TO READ
102 1 C04 XB(167) LEN - # WORDS OF DATA TO MOVE
103 1 C04 XB(168) DATEND - BLOCK # WHERE DATA ITEM ENDS
104 1 C04 XB(169) TOPLPLK - BLOCK # OF FIRST BLOCK IN DATBUF
105 1 C04 XB(170) ENDBLK - BLOCK # OF LAST BLOCK READ
106 1 C04 XB(171) TOCHD - INDEX INTO TOC OF CURRENT DATA ITEM
107 1 C04 XB(175) IL - # WORDS OF DATA TO READ
108 1 C04 XB(176) TOCENT - DATA BASE TOC ENTRY
109 1 C04 XB(201) NOTOC - # TOC ENTRIES IN TOCBUF
110 1 C04 XB(207) TOCBUF - TOC BUFFER
1 BEGIN XRDE
2 INITIALIZE FILE INDICES TO INDICATE NO DATA IN DATBUF
3 INITIALIZE REQUEST BUFFER TO SAY NO REQUESTS
4 DO WHILE THERE ARE NON-DATBUF FILES TO PROCESS
5 IF ALLFLG IS ZERO OR IF STORE/RESTORE BIT IS ON THEN
6 IF DATA IS NOT CURRENTLY IN DATBUF THEN
7 CALL READ TO READ 1 BUFFER BEGINNING WITH DATBLK FOR THIS ELEMENT
8 ERREXIT IF READ ERROR TO :ERR1:
9 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
10 ELSE, DATA BEGINS IN DATBUF
11 IF DATA DOES NOT END IN DATBUF THEN
12 CALL XMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
13 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
14 CALL READ TO READ ENOUGH TO FILL DATBUF
15 ERREXIT IF READ ERROR TO :ERR1:
16 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
17 ENDIF
18 BUILD AMA REQUEST TO ALLOCATE AND STORE DATA
19 CALL EXEC TO WRITE DATA TO SAN
20 ERREXIT IF ERROR FROM EXEC TO :ERR2:
21 IF AMA REQUEST BUFFER IS FULL THEN
22 CALL XRDE TO ISSUE REQUEST
23 EXIT XRDE IF ERROR IN XRDE
24 ENDIF
25 ENDIF
26 ENDDO
27 EXIT XRDE
28 :ERR1:
29 CALL XR16 TO CONVERT ERROR CODE TO ASCII
30 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
31 GO TO :ERR3:
32 :ERR2:
33 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
34 :ERR3:
35 SET ABFLG TO SAY ABOUT RESTORE
36 END XRDE
**FORTRAN CALLING SEQUENCE:**

**CALL XDRDF (DATBUF, DBDCB)**

**XDRDF HANDLES THE RESTORING OF DRDE FILES FROM A MDD/UTDB FILE**

**INPUTS IN CALLING SEQUENCE:**

DATBUF - (INTEGER, 1408 WORDS) BUFFER USED TO READ IN
RECORDS OF DRDE FILES FROM DATA BASE FILE

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

**OUTPUTS IN CALLING SEQUENCE:**

**DBDCB**

**OUTPUTS IN XB COMMON:**

XB(151) ABFLG

**INTERNAL XB COMMON USED:**

XB(151) ABFLG - ABORT FLAG

XB(152) ERFLG - ERROR MESSAGE FLAG

XB(153) ALLFLG - RESTORE ALL UTDB FLAG

XB(155) DEBUG - DEBUG FLAG

XB(159) FILENAME - DB FILE NAME

XB(162) DATBLK - BLOCK # WHERE DRDE BEGINS

XB(163) DATINDEX - WORD INDEX INTO DATBUF WHERE DRDE BEGINS

XB(164) ERR - ERROR FLAG FOR FHGR CALLS

XB(165) TOCOURS - # WORDS OF TOC IN TOCBUF (HOTOC #) + 8

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

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

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

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

XB(171) TOCINDEX- INDEX TO CURRENT TOC ENTRY

XB(172) DRDFIL - DRDE FILE NAME

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

XB(201) TOCBUF- TOC BUFFER

**COMMON USED:**

XB(151) ABFLG - ABORT FLAG

XB(152) ERFLG - ERROR MESSAGE FLAG

XB(153) ALLFLG - RESTORE ALL UTDB FLAG

XB(155) DEBUG - DEBUG FLAG

XB(159) FILENAME - DB FILE NAME

XB(162) DATBLK - BLOCK # WHERE DRDE BEGINS

XB(163) DATINDEX - WORD INDEX INTO DATBUF WHERE DRDE BEGINS

XB(164) ERR - ERROR FLAG FOR FHGR CALLS

XB(165) TOCOURS - # WORDS OF TOC IN TOCBUF (HOTOC #) + 8

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

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

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

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

XB(171) TOCINDEX- INDEX TO CURRENT TOC ENTRY

XB(172) DRDFIL - DRDE FILE NAME

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

XB(201) TOCBUF- TOC BUFFER
1061 1 BEGIN XDRDF
1062 2  DO WHILE THERE ARE TOC ENTRIES TO PROCESS
1063 3  IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
1064 4  CALL XDREN TO CREATE THE BDRE FILE NAME
1065 5  CALL CREAF TO CREATE THE BDRE FILE
1066 6  IF THERE WAS A CREATE ERROR THEN
1067 7  SET ERROR FLAG ON
1068 8  CALL XRMSG TO DISPLAY ERROR MESSAGE
1069 9  ENDIF
1070 5  CALL XRIG TO CONVERT ERROR CODE TO ASCII
1071 5  CALL XREC TO DISPLAY BDRE NAME AND ERROR CODE
1072 4  ELSE
1073 5  CALL CLOSE TO CLOSE FILE
1074 5  ERREXIT IF CLOSE ERROR TO :ERR1:
1075 5  CALL OPEN TO OPEN BDRE FILE AS TYPE 1
1076 5  ERREXIT IF OPEN ERROR TO :ERR1:
1077 5  DO UNTIL ALL BLOCKS OF BDRE HAVE BEEN PROCESSED
1078 6  IF DATA FOR BDRE IS IN DATAIF THEN
1079 7  CALL WRITF DATA TO BDRE
1080 7  ERREXIT IF WRITF ERROR TO :ERR1:
1081 6  ELSE
1082 7  CALL READF TO READ NEXT BUFFER OF DATA
1083 7  ERREXIT IF READF ERROR TO :ERR2:
1084 7  SET INDICES INDICATING DATA IN BUFFER
1085 6  ENDIF
1086 5  EENDO
1087 5  CALL CLOSE TO CLOSE BDRE FILE
1088 5  ERREXIT IF CLOSE FAILED TO :ERR1:
1089 5  BUILD ANA REQUEST TO ALLOCATE BDRE IN ANA
1090 5  IF ANA REQUEST BUFFER IS FULL THEN
1091 6  CALL XRDR TO MAKE REQUEST
1092 6  EXIT XRDF IF XRDR ERROR
1093 5  ENDIF
1094 4  EENDF
1095 3  EENDF
1096 2  EENDO
1097 1  EXIT XDRDF
1098
1099 2  :ERR1:
1100 2  CALL PURGE TO PURGE BDRE FILE
1101 2  :ERR2:
1102 2  CALL XRMSG TO DISPLAY ERROR MESSAGE WITH FILE NAME
1103 2  SET ABFLG TO SAY ABORT RESTORE
1104 1  EEND XDRDF
1106  1 C*********
1107  1 C     FORTRAN CALLING PROCEDURE
1108  1 C
1109  1 C     CALL IXREC
1110  1 C
1111  1 C*********
1112  1 C
1113  1 C81 IXREC PROCESSES THE RECALL DIRECTIVE. A UDDB IS CREATED AND
1114  1 C81 THE CONTENTS OF THE SPECIFIED PRO ARE COPIED TO IT.
1115  1 C81
1116  1 C81
1117  1 C*********
1118  1 C82 INPUT
1119  1 C82 COMMON XE - CARTR, COMBF, COMPT, FLATS, LV, TOKENS
1120  1 C82 COMMON XE - REGBF, REOPTR
1121  1 C82 COMMON XE - REGBF, REOPTR
1122  1 C82 COMMON XE - REGBF, REOPTR
1123  1 C82 FILES - *XXX* (PRO FILE SPECIFIED)
1124  1 C82 FILES - *XXX* (UDDB FILE SPECIFIED)
1125  1 C*********
1126  1 C83 OUTPUT
1127  1 C83 INTERNAL VARIABLES
1128  1 C83 DCP30 - DCB FOR THE PRO FILE; ALLOCATED IN XE COMM;
1129  1 C83 CONTAINS 1552 WORD BUFFER USED TO READ THE PRO
1130  1 C83 AND TO WRITE THE UDDB
1131  1 C83 DCBD - DCB FOR THE UDDB FILE; ALLOCATED IS XE COMMON
1132  1 C*********
1144 1 BEGIX XREC
1145 2 SET STATUS FLAG INDICATING PRO & UTD0 FILES NOT OPEN
1146 3 ERHIT IF "" IS NOT NEXT TOKEN :ERROR:
1147 4 INCREMENT TO NEXT TOKEN
1148 5 ERHIT IF NEXT TOKEN IS NOT "NAME" :ERROR:
1149 6 ERHIT IF THIS NAME IS > 1 CHARACTERS :ERR16:
1150 7 ERHIT IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERR16:
1151 8 RETAIN THIS NAME AS PRO
1152 9 RETAIN CURRENT USER IN (QUALIFIER)
1153 10 INCREMENT TO NEXT TOKEN
1154 11 IF NEXT TOKEN IS "N", THEN
1155 12 INCREMENT TO NEXT TOKEN
1156 13 ERHIT IF NEXT TOKEN IS NOT "NAME" :ERROR:
1157 14 ERHIT IF "NAME" IS GREATER THAN 1 CHARACTER :ERR19:
1158 15 ERHIT IF QUALIFIER < "A" OR > "Z" :ERR19:
1159 16 ERHIT IF QUALIFIER "R"RETAIR NAME AS UTD0
1160 17 INCREMENT TO NEXT TOKEN
1161 18 ERRHIT IF NEXT TOKEN IS NOT EOF :ERROR:
1162 19 BUILD AWA REQUEST TO VERIFY EXISTENCE OF UTD0
1163 20 CALL AWA TO PROCESS AWA REQUEST
1164 21 EXHIT IF UTD0 DOES ALREADY EXIST :ERR22:
1165 22 CALL AWA9 TO VERIFY PRO AND RETRIEVE SIZE
1166 23 ERHIT IF PRO DOES NOT EXIST :ERR16:
1167 24 ERHIT IF UTD0 NAME NOT EXIST :ERR44:
1168 25 CALL XBFNK TO CULB PRO FILE NAME
1169 26 CALL OPEN TO OPEN PRO FILE
1170 27 ERRHIT IF OPEN FAILED :ERR18:
1171 28 ERRHIT IF OPEN FAILED :ERR18:
1172 12 ERRHIT IF OPEN FAILED :ERR18:
1173 13 ERRHIT IF OPEN FAILED :ERR18:
1174 14 ERRHIT IF OPEN FAILED :ERR18:
1175 15 ERRHIT IF OPEN FAILED :ERR18:
1176 16 ERRHIT IF OPEN FAILED :ERR18:
1177 17 ERRHIT IF OPEN FAILED :ERR18:
1178 18 ERRHIT IF OPEN FAILED :ERR18:
1179 19 ERRHIT IF OPEN FAILED :ERR18:
1180 20 ERRHIT IF OPEN FAILED :ERR18:
1181 21 ERRHIT IF OPEN FAILED :ERR18:
1182 22 ERRHIT IF OPEN FAILED :ERR18:
1183 23 ERRHIT IF OPEN FAILED :ERR18:
1184 24 ERRHIT IF OPEN FAILED :ERR18:
1185 25 ERRHIT IF OPEN FAILED :ERR18:
1186 26 ERRHIT IF OPEN FAILED :ERR18:
1187 27 ERRHIT IF OPEN FAILED :ERR18:
1188 28 ERRHIT IF OPEN FAILED :ERR18:
1189 29 ERRHIT IF OPEN FAILED :ERR18:
1190 30 ERRHIT IF OPEN FAILED :ERR18:
1191 31 ERRHIT IF OPEN FAILED :ERR18:
1192 32 ERRHIT IF OPEN FAILED :ERR18:
1193 33 ERRHIT IF OPEN FAILED :ERR18:
1194 34 ERRHIT IF OPEN FAILED :ERR18:
1195 35 ERRHIT IF OPEN FAILED :ERR18:
1196 36 ERRHIT IF OPEN FAILED :ERR18:
1197 37 ERRHIT IF OPEN FAILED :ERR18:
1198 2 :ERROR: ISSUE MESSAGE - "SYNTAX ERROR. EXTRANEOUS DATA"
1199 2 :ERROR: ISSUE MESSAGE - "ILLEGAL: UTD0 NAME (NOT FOUND OR TOO LONG)"
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 - "AVA OVERFLOW - XXX NOT LOGGED"
1206 2 :ERROR22: ISSUE MESSAGE - "XXX ALREADY EXISTS"
1207 2 :ERROR44: ISSUE MESSAGE - "FILE ACCESS ERROR $-- XPDB"
1208 2 :RETURN:
1209 2 IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
1210 3 PURGE UTDB
1211 2 ENDIF
1212 2 IF STATUS FLAG INDICATES UTDB IS LOGGED IN AVA, THEN
1213 3 CALL XBR TO DELETE UTDB FROM AVA
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"
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1220</td>
<td><code>******</code></td>
</tr>
<tr>
<td>1221</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1222</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1223</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1224</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1225</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1226</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1227</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1228</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1229</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1230</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1231</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1232</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1233</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1234</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1235</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1236</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1237</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1238</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1239</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1240</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1241</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1242</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1243</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1244</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1245</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1246</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1247</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1248</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1249</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1250</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1251</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1252</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1253</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1254</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1255</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1256</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1257</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1258</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1259</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1260</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1261</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1262</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1263</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1264</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1265</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1266</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1267</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1268</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1269</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1270</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1271</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1272</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1273</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1274</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1275</td>
<td><code>CD0</code></td>
</tr>
<tr>
<td>1276</td>
<td><code>CD0</code></td>
</tr>
</tbody>
</table>

**FORTRAN CALLING PROCEDURE**

`XREN` processes a user request to rename a sequence table, interface table, data element, or pool. Only those tables or elements logged in the users file are renamed.

**COMMON XE** - CARTAG, COMBUF, COMPR, QUAL, RELPR, TOKENS

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

**COMMON XB** - (20) ICDB: EXEC BUFFER

**EXTERNAL REFERENCES**

**Routines Used** -

- EXEC
- JANO
- KCVT
- MANG
- XDBRA
- XDWRD
- XREG
- XRMOV
- XRMSG
- XRGN
- XRFR

**INTERNAL VARIABLES**

**INPUT**

**OUTPUT**

**COMMON XE - RERBUF**

**COMMON XS - INTERNAL VARIABLES**

**COMMON XS - (2) ASTM: FILE NAME**

**COMMON XS - (3) I: INDEX**

**COMMON XS - (149) JER: FILE MANAGER ERROR RETURN**

**COMMON XS - (150) FNAME: NEW NAME FOR FILE**

**COMMON XS - (153) CMNAME: NAME FOR NAME IN COMBUF**

**COMMON XS - (154) CNAM: NAME FOR NAME IN COMBUF**

**COMMON XS - (158) TOCENTRY: TOC ENTRY FOR DATA BASE NAME**

**COMMON XS - (160) TATELS: DATA CLASS CODE (STORED IN LEFT BYTE)**

**COMMON XS - (201) ICDB: EXEC BUFFER**
1280 1 BEGIN XDREM
1281 2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
1282 2 SAVE POINTER TO CURRENT NAME
1283 2 INCREMENT TO NEXT TOKEN
1284 2 IF TOKEN IS A HYPHEN, THEN
1285 2 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
1286 3 DECODE CLASS NAME
1287 2 EXIT TO :CLASER: IF CLASS SPECIFIED IS NOT VALID (S, D, F, B)
1288 2 ELSE
1289 3 SET CLASS TO BE
1290 3 ENDF
1291 1 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
1292 2 SAVE POINTER TO NEW NAME
1293 2 IF CLASS IS DATA BASE OR DRDE, THEN
1294 2 EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS
1295 2 ENDF
1296 2 IF CLASS IS DATA BASE, THEN
1297 2 EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
1298 2 ENDF
1299 2 IF CLASS IS INTERFACE TABLE, THEN
1300 3 CALL EXEC TO WRITE/READ NEW NAME
1301 3 BUILD AWA MANAGER REQUEST TO CHANGE NAME IN TOC
1302 3 BUILD AWA MANAGER REQUEST TO STORE NEW NAME IN TABLE
1303 3 CALL XREN TO ISSUE REQUESTS
1304 3 EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR
1305 2 ELSE
1306 3 IF CLASS IS DATA BASE, THEN
1307 4 BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY
1308 5 EXIT TO :NAMERR: IF RETURN INDICATES ERROR
1309 3 EXIT TO :INVALID: IF DATA BASE IS AN NDB
1310 3 ENDF
1311 3 BUILD AND ISSUE MANAGER REQUEST FOR NAME CHANGE
1312 3 EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR
1313 3 IF CLASS IS DATA BASE OR DRDE, THEN
1314 4 CALL XRPFN TO FORMAT FILE NAMES
1315 4 CALL FILE MANAGER TO CHANGE DISC FILE NAME
1316 3 EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR
1317 4 IF FILE IS A PDB, THEN
1318 5 CALL XDDDB TO DELETE OLD PDB FROM XDPB
1319 4 EXIT TO :NAMERN: IF ERROR RETURNED
1320 5 CALL XDDDB TO ADD NEW PDB NAME TO XDPB
1321 4 EXIT TO :TELUSER: IF ERROR RETURNED
1322 4 ENDF
1323 3 ENDF
1324 2 ENDF
1325 1 EXIT XREN
1326 2 :SYNTAX: CALL XRMSG TO DISPLAY SYNTAX ERROR AND EXIT
1327 2 :TOOLNG: CALL XRMSG ("NEW NAME IS TOO LONG") AND EXIT
1328 2 :CALGET:
1329 2 CALL EXEC TO GET BUFFER CONTAINING NEW TABLE NAME
1330 2 :NAMERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT
1331 2 :CLASER: CALL XRMSG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT
1332 2 :INVALID: CALL XRMSG ("AN NDB CANNOT BE RENAMED") AND EXIT
1333 2 :NAMAGM:
1334 2 CALL FILE MANAGER TO CHANGE NAME BACK
1335 2 :UNDO:
1336 2 BUILD AND ISSUE AN AMA MANAGER REQUEST TO CHANGE NAME BACK
1337 2 CALL XRMSG ("FILE MANAGER ERROR # ; RENAME UNSUCCESSFUL") AND EXIT
1338 2 :TELSRN:
1339 2 CALL XRMSG ("PDB NOT LOGGED IN XPDDB; SYSTEM ERROR # . . .") AND EXIT
1340 1 END XDREM
1361 BEGIN XDREQ
1362 CALL XRQO TO PROCESS AVA REQUEST(S)
1363 IF AN ERROR RETURNED BY AVA MANAGER, THEN
1364 IF ERROR FLAG (ERFLG) IS ZERO, THEN
1365 TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED
1366 CALL XRMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
1367 ENDIF
1368 CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
1369 IF CLASS OF ELEMENT IS DRE, THEN
1370 CALL PURGE TO DELETE THE FILE
1371 ELSE, ELEMENT RESIDES IN AVA
1372 CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
1373 ENDIF
1374 IF AVA REQUESTS EXIST IN REGBUF BEYOND FAILING REQUEST, THEN
1375 MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
1376 ENDIF
1377 ELSE
1378 SET REGPTR TO 1 INDICATING NO REQUESTS PRESENT
1379 ENDIF
1380 EXIT XDREQ
1381 END XDREQ
**FORTRAN CALLING PROCEDURE**

**CALL XDRF (DATBUF)**

**INPUT**
COMMON XE - LU, FLAGS, REPRM, REBUF, TOKENS, CARTAG, COMPR, COMBUF

**CALLING SEQUENCE**

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

**OUTPUT**
COMMON XD - REBUF

**COMMON XB - ABFGL = ABORT FLAG, NON-ZERO VALUE FROM XDRF, XDRF, IDLIS OR IDREQ INDICATES ABORT OF RESTORE**

**ALLGL = SET NON-ZERO IF LIST OF ELEMENTS SPECIFIED**

**DEBUG = DEBUG BIT OF 'FLAGS' IN XE COMMON**

**EMLISKB = BLOCK # OF LAST BLOCK READ**

**ERFGL = SET NON-ZERO IF MESSAGE #54 ISSUED SO THAT IT IS ISSUED ONLY ONCE**

**FILNAM = UTDB/MDP FILE NAME**

**TOCBUF = UTDB TOC ENTRIES, 8 WORDS EACH, MAX 1200 WORDS**

**TOCENT = AWA TOC ENTRY FOR THE UTDB**

**TOCNEX = INDEX TO NEXT UTDB TOC ENTRY**

**TOTSIZ = NO. OF BLOCKS REMAINING IN UTDB FILE**

**NOTES**

**Routines Called**

CLOSE
EXEC
OPEN
READF
XDRF
XRDF
XDRR
1476 1 BEGIN XRES
1477 2 SET ADFLG TO ZERO
1478 3 ERREXIT IF NEXT TOKEN IS NOT A COMMAND :ERR09:
1479 4 ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERR09:
1480 5 BETAIN THIS NAME AS DATA BASE TO BE RESTORED
1481 6 INCREMENT TO NEXT TOKEN
1482 7 IF TOKEN IS NOT A COMMAND 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 XRES 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 UDB, THEN
1490 15 CALL XRESF TO CONSTRUCT QUALIFIED FILE NAME
1491 16 ENDF
1492 17 CALL OPEN TO OPEN SPECIFIED FILE
1493 18 ERREXIT IF OPEN FAILED :ERROR:
1494 19 CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1495 20 ERREXIT IF READ FAILED :ERROR:
1496 21 INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1497 22 IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1498 23 CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1499 24 ERREXIT IF READ FAILED :ERROR:
1500 25 ENDF
1501 26 UPDATE TOTALS TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 27 CLEAR ERROR MESSAGE FLAG (ERFLG)
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 XDIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 33 EXIT XRES IF ADFLG SET BY XDIS
1509 34 ENDF
1510 35 CALL XDRDE TO RESTORE AWA RESIDENT ELEMENTS
1511 36 1 EXIT XRES IF ADFLG SET BY XDRDE
1512 37 CALL ZRDF TO RESTORE ORDE'S
1513 38 EXIT XRES IF ADFLG SET BY ZRDF
1514 39 CALL CLOSE TO CLOSE DATA BASE FILE
1515 40 ERREXIT IF CLOSE FAILED :ERROR:
1516 41 DO WHILE AWA REQUESTS REMAIN IN RERBUF
1517 42 CALL XRES TO PROCESS AWA REQUESTS
1518 43 EXIT XRES IF ADFLG SET BY XRES
1519 44 ENDOF
1520 1 EXIT XRES

1521 2 :ERR04: CALL XRES - 'SYNTAX ERROR. EXTRAMEMOS DATA'
1522 2 :ERR06: CALL XRES - 'FILE MANAGER ERROR .........'
1523 2 :ERR09: CALL XRES - 'SYNTAX ERROR. MISSING OR ILLEGAL FIELD'
1524 2 :ERR10: CALL XRES - '....... NOT FOUND'
1525 2 :ERR33: CALL XRES - 'CANT NOT RESTORE A PDB'
1526 2 DO UNTIL ALL AWA REQUESTS IN RERBUF HAVE BEEN PROCESSED
1527 3 IF REQUEST IS TO STORE DATA, THEN
1528 4 CALL EXEC TO FREE THE SPECIFIED CLASS NO. AND SAM BUFFER
1529 3 ENDOF
**FORTRAN CALLING PROCEDURE**

**INPUT**

1. COMMON XE - CARTAG, COMBUF, COMTRA, FLAGS, LU, TOKENS
2. COMMON XE - REGBUF, RERPTRA
3. COMMON XE - UTDB FILE SPECIFIED

**OUTPUT**

1. COMMON XE - UTDB FILE SPECIFIED

**INTERNAL VARIABLES**

1. DCSVPB - DCS FOR THE PDB FILE; ALLOCATED IN XE COMMON
2. DEBUO - DCB FOR THE UTDB FILE; ALLOCATED IN XE COMMON;
3. CONTAINS 1152 WORD BUFFER USED TO READ THE
4. UTDB AND TO WRITE THE PDB.
1572 1 BEGIN YDSAV
1573 2 SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
1574 3 ERROR IF " " IS NOT NEXT TOKEN :ERROR:
1575 4 INCREMENT TO NEXT TOKEN
1576 5 ERROR IF TOKEN IS NOT 'NAME' :ERROR:
1577 6 ERROR IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR6:
1578 7 ERROR IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR6:
1579 8 RETAIN THIS NAME AS PDB
1580 9 INCREMENT TO NEXT TOKEN
1581 10 ERROR IF " " IS NOT NEXT TOKEN :ERROR:
1582 11 INCREMENT TO NEXT TOKEN
1583 12 ERROR IF TOKEN IS NOT 'NAME' :ERROR:
1584 13 ERROR IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR6:
1585 14 ERROR IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR6:
1586 15 RETURN THIS NAME AS PDB
1587 16 INCREMENT TO NEXT TOKEN
1588 17 ERROR IF TOKEN IS NOT EOS :ERROR4:
1589 18 BUILD AWA REQUEST TO RETRIEVE UDB'S TOC ENTRY
1590 19 CALL XNUM TO PROCESS AWA REQUESTS
1591 20 ERROR IF TOC RETRIEVE FAILED :ERROR6:
1592 21 ERROR IF TOC ENTRY DOES NOT INDICATE UDB :ERROR6:
1593 22 CALL XDOQA TO ADD PDB TO XDB
1594 23 ERROR IF PDB LIMIT EXCEEDED :ERROR20:
1595 24 ERROR IF OPEN FAILED :ERROR18:
1596 25 SET STATUS FLAG INDICATING PDB LOGGED
1597 26 CALL XRFIN TO BUILD UDB FILE NAME
1598 27 CALL OPEN TO OPEN UDB FILE (SPECIFYING TYPE 1)
1599 28 ERROR IF OPEN FAILED :ERROR18:
1600 29 SET STATUS FLAG INDICATING UDB FILE OPEN
1601 30 BUILD AWA REQUEST TO ALLOCATE PDB -- USE SIZE OF U1:
1602 31 CALL OPEN TO PROCESS AWA REQUEST
1603 32 ERROR IF PDB IS DUPLICATE :ERROR17:
1604 33 ERROR IF AWA OVERFLOW :ERROR21:
1605 34 SET STATUS FLAG INDICATING PDB LOGGED IN AWA
1606 35 CALL XRFIN TO BUILD PDB FILE NAME
1607 36 CALL CREATE TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE
1608 37 OF UDB FILE E:LM TOC ENTRY
1609 38 ERROR IF CREATE FAILED :ERROR18:
1610 39 SET STATUS FLAG INDICATING PDB FILE OPEN
1611 40 DO FOR ALL DATA IN UDB FILE
1612 41 CALL...AD TO READ 1 BUFFER OF UDB DATA
1613 42 ERROR IF READ FAILED :ERROR2:
1614 43 CALL WRIT TO WRITE 1 BUFFER TO PDB FILE
1615 44 ERROR IF WRITE FAILED :ERROR:
1616 45 EENDO
1617 46 CLOSE UDB FILE
1618 47 CLOSE PDB FILE
1619 48 EXIT YDSAV
1620 49 :ERROR9: ISSUE MESSAGE "SYNTAX ERROR -- ILLEGAL OR MISSING FIELD"
1621 50 :ERROR6: ISSUE MESSAGE "INVALID UDB FILE NAME ...."
1622 51 :ERROR16: ISSUE MESSAGE "INVALID PDB FILE NAME ...."
1623 52 :ERROR4: ISSUE MESSAGE "SYNTAX ERROR. EXTRANEOUS DATA"
1624 53 :ERROR17: ISSUE MESSAGE "PDB .... ALREADY EXISTS"
1625 2  ERR18: ISSUE MESSAGE 'PRO FILE ACCESS ERROR ... '  
1626 2  ERR00: ISSUE MESSAGE 'FILE MANAGER ERROR ... '  
1627 2  ERR20: ISSUE MESSAGE 'AUTHORIZED LIMIT OF ... PRO'S ALREADY REACHED'  
1628 2  ERR21: ISSUE MESSAGE 'ANA OVERFLOW. NOT LOGGED'  
1629 2  RETURN:  
1630 2  IF STATUS FLAG INDICATES PRO FILE IS OPEN, THEN  
1631 2  PURGE PRO FILE  
1632 2  ENDIF  
1633 2  IF FLAG INDICATES PRO IS IN ANA, THEN  
1634 2  CALL PROC TO DELETE PRO FROM ANA  
1635 2  ENDIF  
1636 2  IF FLAG INDICATES UT00 IS OPEN, THEN  
1637 2  CLOSE UT00  
1638 2  ENDIF  
1639 2  IF FLAG INDICATES PRO IS IN XP00, THEN  
1640 2  CALL PROC TO DELETE PRO FROM XP00  
1641 2  ENDIF  
1642 1  END XBSAV
104 1 CPO
105 1 CCO
106 1 CCO CALL XOSTA (LU)
107 1 CTT
108 1 CTO
109 1 CT******
110 1 CTO OUTPUT ANA AND DMA USAGE DATA AND STATISTICS ON ANA COMPACT
111 1 CTO PHASES
112 1 CTO
113 1 CTO INPUT
114 1 CTO LU - LOGICAL UNIT OF OUTPUT DEVICE
115 1 CTO
116 1 CTO
117 1 CTOXB COMMON
118 1 CTO ASZ - TOTAL SIZE OF ANA
119 1 CTO FRE - AMOUNT OF FREE SPACE IN ANA
120 1 CTO DSZ - TOTAL SIZE OF DMA (IN 64 WORD SECTORS)
121 1 CTO
122 1 CTO DFR - AMOUNT OF FREE SECTORS IN DMA
123 1 CTO
124 1 CTO BP - ARRAY OF THREE COMPACT PHASE COUNTS
125 1 CTO
126 1 CTO******
127 1 CTO OUTPUT
128 1 CTO USAGE AND STATISTICS TO LU
129 1 CTO******
130 1 CTO NOTES
131 1 CTO USES EXEC, XRIG
132 1 CTO
133 1 CTO******
134 1 *
135 1 *
136 1 *
137 1 BEGIN XOSTA
138 2 FORMAT AND PRINT ANA USAGE DATA
139 2 FORMAT AND PRINT DMA USAGE DATA
140 2 FORMAT AND PRINT COMPACT STATISTICS
141 1 END XOSTA
01683 1 CO**********
01684 1 CO CALL XDSTO (DATBUF)
01685 1 CO **********
01686 1 CO XDSTO IS THE STORE DIRECTIVE HANDLER. IT VERIFIES INPUTS ON
01687 1 CO DIRECTIVE, BUILDS UTDB TOC, CREATES UTDB AND STORES UTDB TOC
01688 1 CO ENTRY IN AWA.
01689 1 CO **********
01690 1 CO INPUTS FROM CALLING SEQUENCE:
01691 1 CO DATABUF (INTEGER 1480 WORDS) - BUFFER USED TO READ IN AWA TOC.
01692 1 CO **********
01693 1 CO INTERNAL XB COMMON USED:
01694 1 CO
01695 1 CO X0(151) ABFGL - (INTEGER, 1 WORD) ABORT FLAG
01696 1 CO X0(152) ERLFLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
01697 1 CO X0(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER TO BE DISPLAYED
01698 1 CO X0(157) TOTSZ- (INTEGER, 1 WORD) TOTAL SIZE OF UTDB FILE
01699 1 CO X0(158) TOTWRD- (INTEGER, 1 WORD) TOTAL WORDS IN A UTDB FILE
01700 1 CO X0(159) FILE - (INTEGER, 3 WORDS) UTDB FILE NAME (***X)
01701 1 CO X0(162) ATREC- (INTEGER, 1 WORD) RECORD # WHERE DATA GOES NEXT
01702 1 CO X0(164) UDBERR- (INTEGER, 1 WORD) UTDB FILE ERROR FLAG
01703 1 CO X0(166) UDBNAM- (INTEGER, 1 WORD) UTDB NAME (***X)
01704 1 CO X0(201) NOTOC- (INTEGER, 1 WORD) NUMBER OF TOC ENTRIES
01705 1 CO X0(201) TOCBUF- (INTEGER, 1200 WORDS) UTDB TOC BUFFER
01706 1 CO**********
01707 1 CO**********
01708 1 CO COMMON USED:
01709 1 CO**********
01710 1 CO**********
01712 1 CO**********
01713 1 CO**********
01714 1 CO**********
01715 1 CO**********
01716 1 CO**********
01717 1 CO**********
01718 1 CO**********
01719 1 CO**********
01720 1 CO**********
01721 1 CO**********
01722 1 CO**********
01723 1 CO**********
01724 1 CO**********
01725 1 CO**********
01726 1 CO**********
01727 1 CO**********
01728 1 CO**********
01729 1 CO**********
01730 1 CO**********
01731 1 CO**********
01732 1 CO**********
01733 1 CO**********
01734 1 CO**********
01735 1 CO**********

RTE ROUTINES USED:

CLOSE, CREAT, EXEC, KCVT, PURGE, WRITF

FDX ROUTINES USED:

XDLIS, XDBRT, XRCPTR, XREG, XREXT, XRMOV,
XRMSG, XRSET, XRSFL, XRSFR, XRWH, XUDBG

ADO
1737        1 BEGIN XDSTO.
1738        2 :E1 ABLE TO ZERO (ABORT FLAG)
1739        3 ERREXIT IF UTDB NAME IS NOT VALID TO :ERR2:
1740        4 BUILD REQUEST FOR AMA TOC
1741        5 CALL XREQ TO MAKE MANAGER REQUEST
1742        6 CALL EXEC TO GET AMA TOC
1743        7 ERREXIT IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1744        8 DO FOR ALL DATA BASE CLASS ENTRIES
1745        9 ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1746       10 ENDDO
1747       11 SET ERFLG OFF (ERROR MESSAGE FLAG)
1748       12 SET TOTSZ = 0 (UTDB TOTAL SIZE)
1749       13 IF WHOLE AMA IS TO BE STORED THEN
1750       14 DO FOR IT, ST, DE, ORDE AMA TOC ENTRIES
1751       15 DO FOR EACH ENTRY IN THIS CHAIN
1752       16 IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1753       17 PREFIX IS NOT AN AMPERSAND THEN
1754       18 SET STORE/RESTORE BIT ON IN TOC ENTRY
1755       19 INCREMENT TOTSZ BY SIZE OF THIS ELEMENT
1756       20 ENDIF
1757       21 ENDDO
1758       22 ELSE
1759       23 CALL XLIST TO PROCESS LIST TO BE STORED
1760       24 ERREXIT IF ABLE TO NOT ZERO TO :ERR5:
1761       25 ENDIF
1762       26 SET NOTOC = 0 (NUMBER OF UTDB TOC ENTRIES)
1763       27 DO FOR IT, ST, DE, ORDE CHAINS
1764       28 DO FOR EACH ENTRY IN THIS CHAIN
1765       29 IF STORE/RESTORE BIT IS ON THEN
1766       30 TURN STORE/RESTORE BIT OFF
1767       31 BUILD UTDB TOC ENTRY
1768       32 INCREMENT NOTOC BY 1
1769       33 ENDIF
1770       34 ENDDO
1771       35 ELSE
1772       36 ERREXIT IF THERE ARE NO UTDB TOC ENTRIES (NOTOC=0) TO :ERR2:
1773       37 COMPUTE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1774       38 CALL XPS 'T' TO CREATE FILE NAME
1775       39 CALL TO CREATE UTDB FOR TOTSZ
1776       40 ERREXIT ERROR IN CREATE TO :ERR3:
1777       41 CALL XWRITE TO WRITE UTDB FILE
1778       42 ERREXIT IF ABLE TO IS 4 (ORDER LARGER THAN SPECIFIED) TO :ERR1:
1779       43 ERREXIT ABLE TO IS 3 (ORDER FILE ERROR) TO :ERR4:
1780       44 ERREXIT ABLE TO IS 2 (ORDER FILE ERROR) TO :ERR5:
1781       45 CALL WRITE TO WRITE TOC RECORDS AT RECORD 1
1782       46 ERREXIT IF ERROR IN WRITE TO :ERR3:
1783       47 CALL CLOSE TO CLOSE UTDB FILE
1784       48 ERREXIT IF ERROR IN CLOSE TO :ERR3:
1785       49 BUILD REQUEST TO ALLOCATE UTDB IN AMA
1786       50 CALL XREQ TO MAKE REQUEST
1787       51 ERREXIT IF AMA OVERFLOW TO :ERR1:
1789       54 EXIT XDSTO
1790       55 :ERR1:
1791       56 CALL XMSG TO DISPLAY MSGNO
1792       57 GO TO :ERR4:
1793       58 :ERR2:
1794  2 CALL XRMSTO TO DISPLAY MSGNO
1795  2 GO TO :ERR5:
1796  2 :ERR3:
1797  2 CALL XRMSTO TO DISPLAY MSGNO WITH UTDB FILE ERROR CODE
1798  2 :ERR4:
1799  2 CALL CLOSE TO CLOSE UTDB
1800  2 CALL PURGE TO PURGE UTDB
1801  2 :ERR5:
1802  2 CALL XRMSTO TO DISPLAY STORE ABORTED/ UTDB NOT CREATED MESSAGE
1803  1 END XRMSTO
**FORTRAN CALLING PROCEDURE**

**CALL XDOTC**

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

**INPUT**

**FILE**

**MANAGER**

**FNAME**

**FILES**

**OUTPUT**

**FILE**

**XCOMMON**

**COMPTR, RERBUF, RERPR**

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

**LOCAL VARIABLES**

**ASZ**

**TOTAL ALLOCATABLE SIZE OF AWA (SEE MODULE XMWA)**

**ANAL**

**FIRST PORTION OF AWA (HEADER AND TOC)**

**CLASS**

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

**CLS**

**CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE**

**CODE**

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

**EOC**

**END OF TOC CHAIN DESIGNATOR (-32768)**

**FREE**

**CURRENT AMOUNT OF FREE SPACE IN AWA (SEE MODULE XMWA)**

**NO**

**ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)**

**IDIM**

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

**LINE**

**BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE**

**MULT**

**ARRAY OF MULTIPLIERS USED TO DETERMINE NUMBER OF WORDS
IM COLUMNS OF TWO DIMENSIONAL DATA ELEMENTS (FUNCTION OF
DATA TYPE)**

**OPTION**

**ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY**

**ORG**

**ORIGIN ADDRESS OF AWA (SEE MODULE XMWA)**

**POS**

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

**SIZE**

**TOC ENTRY SIZE FIELD (WORD 7)**

**TOCMA**

**MAXIMUM SIZE OF TOC WHICH CAN BE TOTALLY ACCOMMODATED BY**

**EXTERNAL BUFFER**

**TYPE**

**TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)**

**UNIT**

**LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY**

**NOTES**

**USES CLOSE, EXEC, IAND, IXOR, KCVT, OPEN, READF, XDOTA, XREL,
XRIG, XRMOV, XRMS, XRGS, XRHF, XRUB**

**IN THE EVENT THE COMPACTED AWA TOC AND HEADER DATA WILL NOT FIT IN THE ALLOCATED BUFFER SPACE, AS MUCH OF EACH CHAIN AS POSSIBLE WILL BE DISPLAYED UNTIL THE LINK FIELDS LEAD BEYOND THE BUFFER.**

**IT IS ASSUMED THAT XDOTC AND XDSTO USE THE SAME SIZE BUFFER FOR TOC MANIPULATION; THUS, A DATA BASE FILE MAY NOT HAVE A TOC TOO LARGE FOR THE XDOTC INTERNAL BUFFER.**
1 BECM XDOTC
2 (c) FOR OUTPUT OF ALL USER CLASSES FROM AMA TO TERMINAL
3 IF NEXT TOKEN IS NOT EOS
4 THEN
5 IF TOKEN IS A HYPHEN
6 THEN
7 INCREMENT TO NEXT TOKEN
8 IF TOKEN IS NOT A CHARACTER T
9 THEN
10 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
11 SET OUTPUT UNIT FOR LINE PRINTER
12 ENDF
13 INCREMENT TO NEXT TOKEN
14 ENDF
15 IF TOKEN IS NOT EOS
16 THEN
17 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
18 INCREMENT TO NEXT TOKEN
19 IF TOKEN IS NOT A COMMA
20 THEN
21 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (0, F, I, S OR B)
22 SET OPTION FOR INDICATED CLASS
23 INCREMENT TO NEXT TOKEN
24 ENDF
25 IF TOKEN IS NOT EOS
26 THEN
27 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
28 INCREMENT TO NEXT TOKEN
29 EXIT TO :ERROR: IF TOKEN IS NOT A FOUR CHARACTER NAME
30 INCREMENT TO NEXT TOKEN
31 EXIT TO :ERROR: IF TOKEN IS NOT EOS
32 CALL XREQ TO RETRIEVE DATABASE FILE TOC ENTRY
33 EXIT TO :ERROR: IF NOT LOGGED IN TOC
34 GET TYPE AND SET NAME IN HEADER
35 CONSTRUCT FILE NAME
36 READ FIRST DATABASE FILE TOC RECORD
37 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AMA TOC
38 READ SUBSEQUENT TOC RECORDS
39 EXIT TO :ERROR7: IF FILE ACCESS FAILS
40 DO FOR EACH NON-EMPTY TOC CHAIN
41 JUMP TO TOC ENTRY POINTED TO BY CHAIN HEAD
42 IF NOT FIRST ENTRY IN TABLE, i.e., A PREVIOUS NON-NULL CHAIN EXISTED
43 THEN
44 MARK PREVIOUS ENTRY AS AN END OF CHAIN
45 ENDF
46 EMDOD
47 DO FOR EACH ENTRY IN TOC
48 IF NOT MARKED AS AN END OF CHAIN
49 THEN
50 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
51 ENDF
52 ENDMOD
53 EXCLUDE CHAIN B (DATA BASE FILES) FROM DISPLAY
54 ENDF
55 ENDF
56 IF REFERENCING AMA TOC
57 THEN
58 CALL XREQ TO RETRIEVE AMA TOC
IF DEBUG AND/OR TRACE FLAGS ARE SET
   THEN
   EXTEND OPTION TO INCLUDE SYSTEM CLASSES (0, 1, 5 AND 7)
   ENDIF
ENDIF
OUTPUT TOC HEADER
DO FOR EACH CLASS INDICATED BY OPTION
OUTPUT CLASS HEADER
LOCATE CLASS CHAIN HEAD
DO UNTIL END OF CHAIN FOUND (=-32768)
INDEX TO NEXT TOC ENTRY
IF CHAIN "OINTS WITHIN LIMIT OF BUFFER
   THEN
   FORMAT NAME & SIZE FIELDS
   IF DEBUG AND/OR TRACE FLAGS SET
   THEN
   FORMAT I-DIM FIELD IN OCTAL
   ENDIF
   FORMAT TYPE FIELD IN INTEGER
   IF CLASS 2 OR 3
   THEN
   FORMAT TYPE FIELD USING DATA TYPE TABLE
   IF CLASS 2
   THEN
   FORMAT I-DIM & J-DIM FIELDS
   ENDIF
   ELSE
   IF CLASS 0
   THEN
   FORMAT TYPE FIELD USING FILE TYPE TABLE
   ENDIF
   ELSE
   PRINT 'DATA LOST' MESSAGE
   EXIT PROCESSING FOR THIS CHAIN
   ENDIF
ENDO
ENDO
IF PROCESSING AWA TOC
THEN
CALL XDATA TO DISPLAY AWA USAGE STATISTICS
ENDIF
EXIT XDTOC
:ERROR: EXIT WITH INVALID OUTPUT DEVICE ID
:ERROR: EXIT WITH SYNTAX ERROR
:ERROR: EXIT WITH INVALID CLASS DESIGNATOR
:ERROR: EXIT WITH INVALID UMDB FILE NAME
:ERROR: EXIT WITH UMDB FILE ACCESS ERROR
END XDTOC
1976  CD----------
1977  CD
1978  CD
1979  CD
1980  CD
1981  CD
1982  CD
1983  CD
1984  CD
1985  CD
1986  CD
1987  CD
1988  CD
1989  CD
1990  CD
1991  CD
1992  CD
1993  CD
1994  CD
1995  CD
1996  CD
1997  CD
1998  CD
1999  CD
2000  CD
2001  CD
2002  CD
2003  CD
2004  CD
2005  CD
2006  CD
2007  CD
2008  CD
2009  CD
2010  CD
2011  CD
2012  CD
2013  CD
2014  CD
2015  CD
2016  CD
2017  CD
2018  CD
2019  CD
2020  CD
2021  CD
2022  CD
2023  CD
2024  CD
2025  CD
2026  CD
2027  CD
2028  CD
2029  CD
2030  CD
2031  CD----------

1976  CD----------
1977  CD
1978  CD
1979  CD
1980  CD
1981  CD
1982  CD
1983  CD
1984  CD
1985  CD
1986  CD
1987  CD
1988  CD
1989  CD
1990  CD
1991  CD
1992  CD
1993  CD
1994  CD
1995  CD
1996  CD
1997  CD
1998  CD
1999  CD
2000  CD
2001  CD
2002  CD
2003  CD
2004  CD
2005  CD
2006  CD
2007  CD
2008  CD
2009  CD
2010  CD
2011  CD
2012  CD
2013  CD
2014  CD
2015  CD
2016  CD
2017  CD
2018  CD
2019  CD
2020  CD
2021  CD
2022  CD
2023  CD
2024  CD
2025  CD
2026  CD
2027  CD
2028  CD
2029  CD
2030  CD
2031  CD----------
1 BEGIN KDRT
2 SET WRDMO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
3 SET MORES = 0 (NUMBER OF ASA REQUEST IN REQBUF)
4 DO FOR ALL UDBR TOC ENTRIES UNTIL CLASS IS DRDE
5 BUILD REQUEST FOR DATA FROM ASA
6 INCREMENT MOREQ BY 1
7 IF REQUEST BUFFER IS FULL (MOREQ=8) THEN
8 PERFORM READING TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
9 ENDF
10 IF THERE ARE REMAINING REQUESTS (MORES>0) THEN
11 SET NEXT REQUEST TO SAY END OF REQUEST LIST
12 PERFORM READING TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
13 IF THERE IS A PARTIAL DATA RECORD LEFT (WRDMO<128) THEN
14 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
15 ENDF
16 DO FOR EACH DRDE UTDB TOC ENTRY
17 STORE DATREC IN UTDB TOC ENTRY
18 CALL BNFPM TO CREATE FILE NAME
19 IF DRDE FILE IS TYPE 3 THEN
20 CALL OPEN TO OPEN FILE AS CORRECT TYPE
21 ERREXIT IF OPEN ERROR TO :ERR1:
22 DO UNTIL EOF IS READ
23 CALL REAFD TO READ 1 RECORD
24 ERREXIT IF READ ERROR TO :ERR1:
25 STORE RECORD LENGTH AT FRONT AND REAR OF DATA
26 INCREMENT WRDMO BY LENGTH + 2
27 IF THERE IS ENOUGH DATA TO WRITE (WRDMO<128) THEN
28 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
29 ENDF
30 IF THERE IS REMAINING DATA (WRDMO<128) THEN
31 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
32 ENDF
33 SET DATREC TO NEXT AVAILABLE RECORD FOR DATA
34 ELSE
35 CALL OPEN TO OPEN FILE AS TYPE 1
36 ERREXIT IF OPEN ERROR TO :ERR1:
37 COMPUTE TOTAL SIZE OF FILE IN WORDS
38 DO UNTIL ALL DATA IS COPIED TO UTDB (SIZE<0)
39 IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
40 SET LENGTH = SIZE
41 ENDF
42 CALL REAFD TO READ LENGTH DATA
43 ERREXIT IF READ ERROR TO :ERR1:
44 CALL WRITF TO WRITE LENGTH DATA
45 ERREXIT IF WRITE ERROR TO :ERR1:
46 INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
47 DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
48 ENDDO
49 ENDF
50 CALL CLOSE TO CLOSE DRDE FILE
51 ERREXIT IF CLOSE ERROR TO :ERR1:
52 ENDO
53 EXIT KDRT
BEGIN RDAT:
  CALL XREG TO MAKE REQUEST
  ERREXIT IF THERE IS AN ANSI OVERFLOW TO :ERR4:
  DO FOR NUMBER OF REQUESTS (MORE?) IN BUFFER
  CALL EXEC TO GET DATA FROM SAM
  SET DATREC AND WRONO INTO THIS UTDB TOC ENTRY
  INCREMENT WRONO BY SIZE OF THIS ELEMENT
  IF THERE IS ENOUGH DATA TO WRITE (WRONO>128) THEN
    PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
  ENDIF
ENDDO
REINITIALIZE REQUEST BUFFER (MORE=0)
END RDAT

* *

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

:ERR1:
  CALL XMSG WITH DRE NAME AND RC
  SET ABIFLG TO SAY A DRE ERROR
  GO TO :ERR3:

:ERR2:
  CALL XMSG TO SAY DRE FILE LARGER THAN SPECIFIED IN TOC
  SET ABIFLG TO SAY DRE FILE ERROR

:ERR3:
  CALL CLOSETO CLOSE DRE FILE
  SET ABIFLG TO SAY UTDB FILE ERROR
  EXIT XDWT

:ERRA:
  SET ABIFLG TO SAY ANSI OVERFLOW
  DO FOR REMAINING REQUESTS
  CALL EXEC TO FREE CLASS NUMBER
ENDDO
END XDWT
FORTRAN CALLING PROCEDURE

CALL HELDS (THREE WORD ARRAY CONTAINING 'XECAL')

FD EXECUTIVE INITIALIZATION SEGMENT MAIN ROUTINE

INPUT

COMMON XE - SUBSTA

OUTPUT

COMMON XB - INITIALIZED ACCORDING TO THE VALUE OF SUBSTA

SEE APPROPRIATE INITIALIZATION SUBROUTINE

NOTE?

USES XEIO, XEINI, XEINS, XEINX, XERIN

BEGIN XECAL

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

:GLOBAL: CALL XEINX TO INITIALIZE GLOBAL COMMON

:DIRECT: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES

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

:INTERD: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING

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

END XECAL
1CDO       FORTRAN CALLING PROCEDURE
40 1CDO
41 1CDO       CALL XEIND
42 1CDO
45 1C******
44 1C01       INITIALIZE XB COMMON FOR DIRECTIVE LEVEL
46 1C01
47 1C******
48 1C02       INPUT
49 1C02       COMMON XB - CARTRG, FLAGS
50 1C02
51 1C02
52 1C******
53 1C03       OUTPUT
54 1C03       COMMON XB - DIRECT, NUMDIR
55 1C03
56 1C******
57 1C05       NOTES
58 1C05       USES CLOSE, EXEC, OPEN, PRFM, READF, XREXT, XRIG, XRMSG, XVABN
59 1C05
60 1C******
61 1*
62 1*
63 1*
64 1*
65 1BEGIN XEIND
66 2READ FDS DIRECTIVE PROMPT FILE
67 2STORE DIRECTIVES IN DYNAMIC COMMON
68 1END XEIND
1 BEGIN XEIME
2 INITIALIZE COMMON TO ZEROS
3 SET FILE (ART=OCIE NUMBER
4 SET STATES TO DIRECTIVE LEVEL
5 SET TKNLNG
6 INITIALIZE TOKENS
7 READ LIBRARY DIRECTORY FIRST RECORD
8 EXIT TO :ERROR: IF READ FAILED
9 STORE NUMBER OF PROCESSORS IN NOPROC
10 READ PROCESSOR NAME RECORD
11 EXIT TO :ERROR: IF READ FAILED
12 CALL XREF TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AMA
13 EXIT TO :ERROR: IF REQUEST FAILED
14 DO FOR EACH ID SEGMENT IN SYSTEM
15 DO FOR EACH PROCESSOR IN LIBRARY DIRECTORY
16 IF NAMES ARE THE SAME
17 THEN
18 NOTE EXISTANCE OF ID
19 ENDIF
20 ENDDO
21 IF ANY PROCESSORS NOT MARKED
22 THEN
23 LIST PROCESSORS MISSING ID SEGMENTS
24 EXIT XEIME WITH INITIALIZATION FAILURE
25 ENDIF
26 CALL OPEN TO OPEN PDB/MDB DIRECTORY ( XPDB)
27 EXIT TO :ERROR: IF OPEN FAILED
28 CALL READF TO READ 1ST RECORD OF XPDB (LIST OF MDB'S)
29 EXIT TO :ERROR: IF READ FAILED
30 PERFORM DLOG TO LOG MDB'S IN AMA
31 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER
32 CALL READF TO READ THAT RECORD OF XPDB
33 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)
34 CALL CLOSE TO CLOSE XPOB
35 PERFORM DLOG TO LOG PDB'S IN AMA
36 IF THERE ARE REQUESTS IN THE AMA REQUEST BUFFER, THEN
37 CALL XERF TO PROCESS THE AMA REQUESTS
38 ENDIF
39 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
40 EXIT XEIME
41 BEGIN DLOG
42 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
43 CALL XERF TO BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
44 IF 8 AMA REQUESTS HAVE BEEN BUILT, THEN
45 CALL XERF TO PROCESS AMA REQUESTS
46 ENDIF
47 EXIT TO :ERROR: IF A REQUEST FAILED
48 ENDDO
49 END DLOG
50 "ERROR: LIBRARY INITIALIZATION ERROR TERMINATION
51 1 END XEIME
FORTRAN CALLING PROCEDURE

CALL XEINI

INITIALIZE XE AND XG COMMON FOR INTERFACE TABLE EDITING

INPUT COMMON XE -
- MASTER = MASTER STATE
- TOKENS = LEXICAL TOKEN VALUES
- MOPROC = NUMBER OF ENTRIES IN LIBRARY DIRECTORY
- COMTR = INDEX OF NEXT TOKEN IN COMBUF
- COMBUF = COMMUNICATIONS BUFFER
- PRGNAM = PROCESSOR NAME FROM EXECUTION CONTROLLER

COMMON XE -
- SUBSTATE = SUBSTATE -- SET TO 1 IF INIT ERROR
- REBUF = REQUEST BUFFER FOR AWI INPUTS
- PRGNAM = PROCESSOR NAME ON INTE DIRECTIVE

COMMON XG -
- WKBLSG = LENGTH OF WKBUF (CONSTANT)
- WKBUF = WORKING BUFFER CONTAINING IN/E. TABLE WITH
- SHORT PROMPTS
- DIRECT = SUPPORTED INTE DIRECTIVES
- NUMDIR = NO. OF ENTRIES IN DIRECT
- MENTAB = INTE. TABLE NAME
- NUMARG = NO. OF ARGUMENTS
- WARG = NO. OF WORDS IN SPG. AND HEADER OF WKBUF
- LITLEN = NO. OF WORDS IN LITERAL AREA
- LITRPX = INDEX TO START OF LITERAL AREA
- ISIZES = ARRAY MAPPING ARG. TYPE TO EFFECTIVE LENGTH

NOTES -
- <XXXXX = DEFAULT INTERFACE TABLE FOR THIS PROCESSOR
- >XXXXX = PROMPT TABLE FOR THIS PROCESSOR

USES FILES -
- USES ROUTINES -
- XMMOV = EXEC
- XMMSG = CLOSE
- XREXT = OPEN
- XREG = READF
- XRUPK = XRXXB
- XRCPK = XRCP
320 3  POSITION TO 1ST TOKEN AFTER 'INTE'
321 3  EREXIT IF TOKEN IS NOT ' ', ERR1:
322 3  INCREMENT TO NEXT TOKEN
323 3  EREXIT IF TOKEN IS NOT A NAME : ERR2:
324 3  SET PRCNAM TO THIS NAME
325 3  INCREMENT TO NEXT TOKEN
326 3  SET INTNAM TO 'Z' 
327 3  SET NEWTAB TO 'ZINTAB'
328 3  IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN
329 4  INCREMENT TO NEXT TOKEN
330 4  IF TOKEN IS NOT EOS, THEN
331 5  IF TOKEN IS A NAME, THEN
332 6  SET INTNAM TO THE NAME
333 6  INCREMENT TO NEXT TOKEN
334 6  ENDIF
335 6  IF TOKEN IS NOT EOS, THEN
336 6  EREXIT IF TOKEN IS NOT A NAME : ERR2:
337 6  INCREMENT TO NEXT TOKEN
338 6  IF TOKEN IS A NAME, THEN
339 7  SET NEWTAB TO THE NAME
340 7  INCREMENT TO NEXT TOKEN
341 7  ENDIF
342 7  EREXIT IF TOKEN IS NOT EOS : ERR2:
343 8  ENDIF
344 8  ELSE
345 9  INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER
346 9  NEWTAB IS SET TO 'ZINTAB'
347 9  ISSUE MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED
348 10  ENDIF
349 11  MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE
350 12  EREXIT IF REQUEST IS UNSUCCESSFUL : ERR11:
351 12  START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR
352 13  EXIT IF PROCESSOR PRCNAM IS FOUND
353 14  ENDIF
354 15  INCREMENT TO NEXT ENTRY
355 16  ENDOLOOP
356 17  EREXIT : ERR8:
357 18  ENDS
358 19  SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND
360 3  IF INTNAM = 0, THEN
361 3  CONSTRUCT THE NAME OF THE DEFAULT INTERFACE TABLE AS "CPRCNAM"
362 3  OPEN THE DEFAULT INTERFACE TABLE FILE
363 3  EREXIT IF OPEN FAILED : ERR2:
364 3  READ THE DEFAULT INTERFACE TABLE
365 3  EREXIT IF THE READ FAILED : ERR16:
366 3  READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM
367 3  OF VBUF
368 3  EREXIT IF READ FAILED : ERR14:
369 3  ELSE
370 3  MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE
371 3  EREXIT IF ERROR OR COULD NOT FIND : ERR:
372 4  ENDIF
373 4  EREXIT IF VERS .ME. VERSION NO. OF TABLE TO BE EDITED : ERR10:
374 5  CONSTRUCT NAME OF PROMPT TABLE AS "CPRCNAM"
375 5  OPEN THE PROMPT TABLE
376 5  EREXIT IF OPEN FAILED : ERR15:
READ 2ND RECORD OF THIS PROMPT TABLE
ERREXIT IF READ FAILED :ERR17:
MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUF TO AREA FOLLOWING
THE PROMPT TABLE
THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
LITERAL DISPL. THAT MATCH EACH LITERAL DATA ENTRY
THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
EXIT XEINI
:ERR2: "SYNTAX ERROR ... "
:ERR4: "INTNM" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNM DOES NOT 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 SUBSTITUTE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
304  1 CD0       FORTRAN CALLING PROCEDURE        XEINS
305  1 CD0
306  1 CD0
307  1 CD0       CALL XEINS                XEINS
308  1 CD0
309  1 CD0       COMMON FOR SEQUENCE TABLE EDITING       XEINS
310  1 CD0
311  1 CD0       INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING       XEINS
312  1 CD0
313  1 CD0       COMMON XE - CARTRG, COMBUF, COMPTR, LU, NOPROC, TOKENS       XEINS
314  1 CD0
315  1 CD0       INPUT       XEINS
316  1 CD0
317  1 CD0       COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,       XEINS
318  1 CD0       OLDTAB, PRMTAB, WBBLNG, WBBUF, XLIBD       XEINS
319  1 CD0
320  1 CD0       COMMON XE - REQBUF, REQPTR, SUBSTA,       XEINS
321  1 CD0
322  1 CD0       COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,       XEINS
323  1 CD0       OLDTAB, PRMTAB, WBBLNG, WBBUF, XLIBD       XEINS
324  1 CD0
325  1 CD0       COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,       XEINS
326  1 CD0       OLDTAB, PRMTAB, WBBLNG, WBBUF, XLIBD       XEINS
327  1 CD0
328  1 CD0       NOTES       XEINS
329  1 CD0
330  1 CD0       USES FILE - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE       XEINS
331  1 CD0
332  1 CD0       USES ROUTINES       XEINS
333  1 CD0
334  1 CD0       EXEC       XEINS
335  1 CD0       IAND       XEINS
336  1 CD0       OPEN       XEINS
337  1 CD0       READF       XEINS
338  1 CD0       XEXIT       XEINS
339  1 CD0       XEEXT       XEINS
340  1 CD0       XRPR       XEINS
341  1 CD0       XREQ       XEINS
342  1 CD0       XREX       XEINS
343  1 CD0       XRMV       XEINS
344  1 CD0       XRMSG       XEINS
345  1 CD0       XUDP6       XEINS
346  1 CD0
347  1 CD0       C*********       XEINS
1 BEGIN XEMS
2  BUILD AWA REQUEST TO RETRIEVE "XLIBD"
3  IF TOKEN IS NOT COMMA :ERROR2:
4  INCREMENT TO NEXT TOKEN
5  IF TOKEN IS NAME, THEN
6  BUILD AWA REQUEST TO RETRIEVE OLDTAB
7  RETAIN THIS NAME AS OLDTAB
8  INCREMENT TO NEXT TOKEN
9  ELSE
10  SET OLDTAB TO ZERO
11  ENDIF
12  IF TOKEN IS NOT COMMA :ERROR2:
13  INCREMENT TO NEXT TOKEN
14  IF TOKEN IS NOT NAME :ERROR2:
15  RETAIN THIS NAME AS NEWTAB
16  IF NEWTAB NOT EQUAL TO OLDTAB, THEN
17  BUILD AWA REQUEST TO VERIFY EXISTENCE OF NEWTAB
18  ENDIF
19  CALL XER TO PROCESS AWA REQUEST(S)
20  IF NEWTAB NOT EQUAL TO OLDTAB, THEN
21  IF NO ERROR FROM XER :ERROR12:
22  ENDIF
23  IF OLDTAB NOT ZERO, THEN
24  IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED :ERROR4:
25  SET PROMPT MODE AS UPDATE
26  SET NO. ENTRIES AS OLDBU SIZE / 7
27  READ OLDTAB INTO WORKING BUFFER
28  IF FIRST REQUEST (RETRIEVE "XLIBD) FAILED :ERROR1:
29  READ "XLIBD INTO COMMON
30  ELSE
31  SET PROMPT MODE AS CREATE
32  SET NO. ENTRIES TO ZERO
33  ENDIF
34  OPEN, READ AND CLOSE FILE XSPRM
35  SET COUNT AND SEREDIT DIRECTIVES INTO XB COMMON
36  EXIT XEMS
37
38  :ERROR1: CALL XRMSG - "INITIALIZATION ERROR ..."
39  :ERROR2: CALL XRMSG - "SYNTAX ERROR"
40  :ERROR4: DEFAULT MESSAGE TO '...NOT FOUND'
41  IF ERROR WAS NO AWA SPACE THEN
42  SET MSG TO '...NO AWA SPACE'
43  ENDIF
44  CALL XRMSG TO DISPLAY MESSAGE
45  :ERROR12: CALL XRMSG - '... ALREADY EXISTS'
46
47  END XEMS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XB - LITPTR, NUMARE, WKBNG, WKBUF

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

OUTPUT

COMMON XB - LITPTR, LITDN, NARC, W'SUF

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

USES ROUTINES

XIEXT
XROV
XRNKB
XRNKS
XRES
1 BEGIN XEINT
2 DO UNTIL ALL LITERAL ENTRIES HAVE BEEN PROCESSED
3 START SEARCH UNTIL ALL ARGUMENTS SEARCHED
4 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
5 B-DIT IS ON
6 BUILD 3 WORD LITERAL ENTRY (3, I-SUB, J-SUB)
7 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
8 D-BIT IS ON
9 IF ARGUMENT IS COMPLETE (C-DIT ON), THEN
10 CREATE BIT MASK WORD(S) IN NEW LITERAL ENTRY AREA
11 ELSE
12 MOVE BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA
13 ENDIF
14 DO UNTIL ALL BITS OF BIT MASK PROCESSED
15 DETERMINE NUMBER OF CONTIGUOUS BITS ON (OR OFF)
16 AND MOVE CORRESPONDING NUMBER OF DATA WORDS
17 (OR 0'S) INTO NEW LITERAL ENTRY AREA
18 ENDDO
19 ELSE
20 INCREMENT TO NEXT ARGUMENT
21 ENDDO
22 ERREXIT IF NOT CALLED BY LIBRARY MAINTANENCE :ERRS:
23 SKIP TO NEXT LITERAL ENTRY
24 ENDS
25 ENDIF
26 1 EXIT TO :RETURN:
27 :ERRS: CALL XRMSG - 'SYSTEM INITIALIZATION ERR 5'
28 2 :RETURN:
29 1 END XEINT
FORTRAN CALLING PROCEDURE

CALL XE

INITIALIZE XE AND XB COMMON FOR EXECUTION CONTROLLER

INPUT

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

AQA - SEQUENCE TABLE, LIBRARY DIRECTORY NAME TABLE

OUTPUT

COMMON XE - COMBUF, MASSTA, REBUNF, REBUNF, SENEND, SERNAME, SECPTR

SESTX, SUBSTA

COMMON XB - LIBD, NOPRCZ, SEQLNG, WBLNG, WBUF

NOTES

USES EXEC, PRTM, XREG, XREXT, XPAR, XMSG, XDEG, XVARN
551  2  :ERR03: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552  2  :ERR04: SET SUBSTA TO DIRECTIVE LEVEL
553  2  IF ERROR WAS 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  :ERR05: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
559  2  :ERR13: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH NUMBER NOT FOUND
560  1  END XEIXX
FORTRAN CALLING PROCEDURE:
CALL XELDS (SEGNM) TO CALL SEGMENT FROM MAIN
CALL XERTN TO RETURN TO MAIN PROGRAM

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

BEGIN XELDS

:CALL .ENTRY 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 CDO  FORTRAM CALLING PROCEDURE
609 1 CDO  CALL XESCN
610 1 CDO  C*********
611 1 CDO  C*********
612 1 CDO  C*********
613 1 CDO  C*********
614 1 CDO  C*********
615 1 CDO  C*********
616 1 CDO  C*********
617 1 CDO  C*********
618 1 CDO  C*********
619 1 CDO  C*********
620 1 CDO  INPUT
621 1 CDO  XE COMMON - CATPNC
622 1 CDO  ANA - AUDSTK, RCSNST (SEE DO AND SCAN PROCESSES)
623 1 CDO  OUTPUT
624 1 CDO  OUTPUT
625 1 CDO  OUTPUT
626 1 CDO  OUTPUT
627 1 CDO  OUTPUT
628 1 CDO  OUTPUT
629 1 CDO  OUTPUT
630 1 CDO  EXTERNAL ROUTINES
631 1 CDO  EXTERNAL ROUTINES
632 1 CDO  EXTERNAL ROUTINES
633 1 CDO  EXTERNAL ROUTINES
634 1 *
635 1 *
636 1 *
637 1 *
638 1 EEGIN XESCN
639 2 BUILD REQUESTS TO DELETE AUDSTK AND RETRIEVE RCSNST THEN DELETE IT
640 2 CALL XREQ TO ATTEMPT REQUESTS
641 2 ISSUE MESSAGE XE18 FOR SUCCESSFUL DELETES
642 2 IF XREQ COMPLETED REQUESTS, I.E., RCSNST EXISTED, THEN
643 3 DO FOR EACH SCAN CONTROL ENTRY IN RCSNST
644 4 CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN
645 4 CALL PURGE TO PURGE THE FILE
646 4 IF PURGE RETURNED AN ERROR, THEN
647 5 CALL XRMSG TO DISPLAY WARNING MESSAGE
648 4 ENDF
649 4 BUILD REQUEST TO DELETE DATA BOX FROM TOC
650 3 ENDDO
651 3 CALL XREQ TO DELETE DATA BOXES FROM ANA TOC
652 2 ENDF
653 1 END XESCN
CD1 FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
CD1 ONE PROGRAM PER SIGNED-ON USER
CD1
CD2 INPUT
CD2 SCHEDULING PARAMETERS - LU, CLASNO, QUAL, FLAGS (SEE XE COMMON)
CD2
CD3 OUTPUT
CD5 COMMON XE - COMBUF, COMPTR, MASSTA, SUBSTA, PLUS XEINE
CD5 INITIALIZATION
CD5 COMMON XB - INITIALIZATIONS FROM XEIRD, XEINI, XEINS, XEINX
CD3
CD5 NOTES
CD5 USES AMPAR, XDCLD, XDCLF, XELDS, XINTE, XRCPR, XRSIG, XSERE,
CD5 XTCOM, XXCT
CD5 THE CALLS TO XELDS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
CD5 XECAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
CD5
CD5 THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CONTROL OCCURS
CD5 BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XXCNT FROM
CD5 CALLING XSERE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
CD5 MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
CD5 WHEN MASSTA IS SET TO THE DIRECTIVE LEVEL.
CD5
CD5
CD5
1 BEGIN EXEC
2 RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASMO, QAL & FLAGS
3 DO FOREVER -- TERMINATES INSIDE HANDLER FOR ZOFF
4 CALL STIM FOR INPUT OF DIRECTIVE
5 IF ERROR OR NOT A VALID DIRECTIVE NAME
6 THEN
7 ISSUE MESSAGE ED6
8 ELSE
9 IF NAME IS INTE
10 THEN
11 SET STATES TO INTE LEVEL
12 CALL XEINI TO INITIALIZE DYNAMIC COMMON
13 EXIT TO :RESET; IF ERROR
14 CALL XINTO TO EDIT TABLE
15 ELSE
16 IF NAME IS SERE
17 THEN
18 SET STATES TO SERE LEVEL
19 CALL XEIMS TO INITIALIZE DYNAMIC COMMON
20 EXIT TO :RESET; IF ERROR
21 CALL XSENE TO EDIT TABLE
22 ELSE
23 IF NAME IS FOR SOME EXECUTION CONTROL OPTION
24 THEN
25 SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
26 DO UNTIL MASSTA IS AT DIRECTIVE LEVEL
27 CALL XEINTO INITIALIZE DYNAMIC COMMON
28 EXIT TO :RESET; IF ERROR
29 CALL XXINTO TO PERFORM EXECUTIONS
30 IF SUBSTA IS SET TO SERE LEVEL
31 THEN
32 CALL XEIMS TO REINITIALIZE DYNAMIC COMMON
33 EXIT TO :RESET; IF ERROR
34 CALL XSENE TO SUPPORT EXECUTION CONTROL
35 ENDIF
36 IF SUBSTA IS SET TO INTE LEVEL
37 THEN
38 CALL XEIMS TO REINITIALIZE DYNAMIC COMMON
39 EXIT TO :RESET; IF ERROR
40 CALL XINTO TO SUPPORT EXECUTION CONTROL
41 ENDIF
42 IF EXECUTION MODE WAS SEMI OR AUTO
43 THEN
44 CALL XESCNT TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES
45 ENDIF
46 ELSE
47 CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?
48 ENDIF
49 ENDIF
50 :RESET:
51 IF SUBSTA IS NOT DIRECTIVE LEVEL
52 THEN
53 CALL XEIND TO REINITIALIZE DYNAMIC COMMON
54 ENDIF
55 ENDIF
56 4
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CD0  FORTRAN CALLING PROCEDURE
2 CD0
3 CD0  CALL XINTE
4 CD0
5 CD0  COMMON
6 ********
7 CD1  OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR
8 CD1
9 CD1  ********
10 CD2
11 CD2  INPUT
12 CD2
13 CD2  COMMON XE -
14 CD2  LU = USER'S LOGICAL UNIT NO.
15 CD2
16 CD2  COMMON XB -
17 CD2  DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE
18 CD2  TABLE EDITOR Routines
19 CD2
20 CD2  ********
21 CD5  NOTES
22 CD5
23 CD5  USES ROUTINES
24 CD5  XINIX
25 CD5  XDMG
26 CD5  XETWN
27 CD5
28 CD5  ********
29 CD6
30 CD6
31 CD6
32 1 * XINTE IS THE INTERFACE ROUTINE FOR THE INTERFACE TABLE EDITOR
33 1 * WHEN CALLED IN THE FDS EXECUTIVE'S OVERLAY STRUCTURE.
34 1 *
35 1 BEGIN XINTE
36 2 CALL XINIX TO EXECUTE INTERFACE TABLE EDITOR
37 2 CALL XETWN TO RETURN TO XEXEC IN MAIN SEGMENT
38 1 END XINTE
FORTRAN CALLING PROCEDURE

CALL XIMIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

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

COMMON XB -
  LITLEN = LENGTH OF LITERAL AREA OF INTERFACE TABLE IN WBUF
  NARG = INDEX TO START OF SHORT PROMPT AREA OF WBUF
  NEWTAB = NAME OF INTERFACE TABLE TO BE CREATED BY THIS EDIT. INPUT = 0 IF XINT 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

ANA = XXXXXX = NEW INTERFACE TABLE AND LITERAL AREA (NOT DONE IF CALLED BY LIBRARY MAINTENANCE)

USING ROUTINES

EXEC
XREG
XILET
XIPRM
XINPT
1 CDS
2 XROMV
3 CDS
4 XBMGE
5 CDS
6 XTCOM

1 C

1 THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'IMT?' DIRECTIVE
1 OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
1 PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
1 TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
1 OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
1 NOTE : ALL INITIALIZATION, INCLUDING WKBUF (OLD INTERFACE TABLE),
1 HAS BEEN PERFORMED BY XEINI.

1 BEGIN XINIX
1 IF NP (NO. OF PARAMETERS) > 0, THEN
1 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
1 * PRMTND = 1 => CREATE M MODE
1 * PRMTND = 3 => CREATE A MODE
1 * PRMTND = 4 => CREATE CONTINUE MODE
1 * PRMTND = 5 => MODIFY MODE
1 DO UNTIL 'EXIT' OR 'X' IS ENTERED
1 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTND, SIZE, TYPE, AND STATUS
1 OF NEXT ARGUMENT
1 CALL XTCOM TO PROMPT USER AND RETURN PARSED INPUT
1 IF 'X' WAS NOT ENTERED, THEN
1 IF '\"' WAS ENTERED, THEN
1 SET PRMTND TO 5
1 ELSE
1 IF NOTHING WAS ENTERED (I.E. TOKEN IS COS), THEN
1 INCREMENT TO NEXT ARGUMENT
1 ELSE
1 CALL XINIP TO PROCESS THE USER'S INPUT
1 ENDIF
1 ENDIF
1 ENDDO
1 IF A 'X' WAS ENTERED, THEN
1 SET RETURN CODE INDICATING X
1 (I.E. MASSTA = 0)
1 ELSE
1 DECOMPRESSION THE LITERAL LIST AREA
1 ENDIF
1 ENDDO
1 STORE INTERFACE TABLE AS NAME
1 IF STORE INTO AMA FAILED, THEN
1 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
1 ELSE
1 SET GOOD RETURN CODE
1 ENDDO
1 END XINIX
1  * CONSTRUCT PROMPT TO BE ISSUED
2  BEGIN XIPM
3   DO UNTIL A PROMPT IS CONSTRUCTED
4   IF PRNTMD = 5, THEN
5      CONSTRUCT A ":" PROMPT
6      ELSE
7         IF PRNTMD = 4 (CONTINUE MODE), OR
8            PRNTMD = 6 (CONTINUE HERE MODE), THEN
9            IF ARGNO IS A SCALAR, THEN
10               CONSTRUCT PROMPT AS \"ARG=\" OR \"ARG\";
11            ELSE
12               IF PRNTMD NOT = 6, THEN
13                 IF THERE ARE NO EMPTY SLOTS BEYOND LAST ENTERED (LASTE)
14                SET PRNTMD TO 4
15                COMPUTE CURRENT ELEMENT NO. (SCRIPT) FROM CURRENT INDEX
16               ENDIF
17               IF THERE ARE NO EMPTY SLOTS BEYOND LASTE, THEN
18                 SET PRNTMD TO NOSAV (EXIT THE CONTINUE MODE)
19               ENDIF
20               IF ARGNO IS DOUBLY SUBSCRIPTED PARAMETER, THEN
21                 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
22                 CONSTRUCT PROMPT AS \"ARG=(I,J)\";
23               ELSE
24                 CONSTRUCT PROMPT AS \"ARG=(ISUB)\";
25               ENDIF
26            ENDIF
27         ELSE
28            IF ARGNO = BP, THEN
29               SET PRNTMD TO 5
30         ENDIF
31         ELSE
32            IF ARGNO NOT = NEXT PARAMETER
33               IF IOFLAG TO 1, 2, OR 3 INDICATING I, O, OR IO
34                 IF IOFLAG = 3, THEN
35                   IF SOME DATA VALUE(S) OR PARAM NAME EXISTS FOR ARGNO, THEN
36                     CALL XILSD TO LIST DATA FOR THIS ARGUMENT
37                   ENDIF
38                   CONSTRUCT PROMPT AS \"ARG=\" OR \"B\" OR \"O\"
39                   ELSE
40                     IF ARGNO IS MARKED INCOMPLETE, THEN
41                       IF A PARTIAL LITERAL LIST EXISTS, OR
42                          THIS ARGUMENT IS A SCALAR, THEN
43                            COMPUTE ISUB AS FIRST EMPTY ELEMENT
44                          IF DOUBLY SUBSCRIPTED PARAMETER, THEN
45                            COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
46                            CONSTRUCT PROMPT \"ARG=(I,J)\";
47                          ELSE
48                            CONSTRUCT PROMPT \"ARG=(ISUB)\";
49                          ENDIF
50                     ELSE
51                       CONSTRUCT PROMPT AS \"ARG=\" OR \"B\" OR \"O\"
52                   ENDIF
53               ENDIF
54            ENDIF
55         ENDIF
56      ENDIF
57   ENDIF
58  ENDIF
FOR CALLING PROCEDURE

CALL XIPMT

PROMPT DIRECTIVE PROCESSOR

INPUT

COMMON IE - COMBUF, COMTA, TOKENS

OUTPUT

COMMON IB - ARGMN, FRMTND

* XIPMT PROCESS IS THE PROMPT DIRECTIVE

BEGIN XIPMT

IF TOKEN IS NOT COMA :ERROR2:

POSITION TO NEXT TOKEN

IF NAME IS NOT NAME :ERROR2:

IF TOKEN(S) FOLLOW THE NAME :ERROR2:

IF NAME IS 'M', THEN

SET FRMTND TO 1

ELSE IF NAME IS 'A', THEN

SET FRMTND TO 3

ELSE

ERROR :ERROR2:

ENDIF

ENDIF

EXIT TO :RETURN:

:ERROR2: CALL XNMSG FOR 'INVALID SYNTAX'

:RETURN:

END XIPMT
337 1  * XILST PROCESSES THE LIST DIRECTIVE
338 1  BEGIN XILST
339 2  IF TOKEN IS ' - ', THEN
340 3  POSITION TO NEXT TOKEN
341 4  ERREXIT IF TOKEN IS NOT NAME : ERR02:
342 5  ERREXIT IF NAME IS NOT 'C', 'Y', OR 'A' : ERR02:
343 6  SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)
344 7  POSITION TO NEXT TOKEN
345 8  ELSE
346 9  IF MODEFG < 2
347 10  ENDIF
348 11  IF TOKEN IS EOS, THEN
349 12  WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
350 13  AND STATUS
351 14  DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
352 15  IF MODEFG = 1 OR MODEFG = 3, THEN
353 16  CALL XICHN TO WRITE CHARACTERISTICS OF THIS ARGUMENT
354 17  ENDIF
355 18  IF MODEFG = 2 OR MODEFG = 3, THEN
356 19  CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
357 20  ENDIF
358 21  3  ENDDO
359 22  ELSE
360 23  3  DO UNTIL EOS IS REACHED
361 24  ERREXIT IF TOKEN IS NOT COMMA : ERR02:
362 25  ERREXIT IF NEXT TOKEN IS NOT NAME : ERR02:
363 26  SET ARGNO TO 1
364 27  START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
365 28  EXIT IF NAME = ARGNO'S NAME IN PROMPT TABLE
366 29  IF MODEFG = 1 OR MODEFG = 3, THEN
367 30  CALL XICHN TO WRITE CHARACTERISTICS OF THIS ARGUMENT
368 31  ENDIF
369 32  IF MODEFG = 2 OR MODEFG = 3, THEN
370 33  CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
371 34  ENDIF
372 35  ENDLOOP
373 36  PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
374 37  ENDSRCH
375 38  INCREMENT TO NEXT TOKEN
376 39  ENDDO
377 40  ENDIF
378 41  EXIT TO : RETURN:
379 42  : ERR02: CALL XRM5G TO WRITE 'INVALID SYNTAX'
380 43  : RETURN:
381 44  1 END XILST
383 1 CD0             FORTRAN CALLING PROCEDURE
384 1 CD0             CALL XISUB
385 1 CD0             C************
386 1 CD0             C EFFECTIVE SUBSCRIPT CALCULATION ROUTINE
387 1 CD0             C************
388 1 CD0             INPUT
389 1 CD0             COMMON XE - COMBUF, COMPIR, TOKENS
390 1 CD0             COMMON XB - IDIM, ISIZE, LENEFF
391 1 CD0             COMMON X0 - IRETC, ISUB
392 1 CD0             C************
393 1 CD0             OUTPUT
394 1 CD0             COMMON X0 - IRETC, ISUB
395 1 CD0             COMMON X5 - USES ROUTINES
396 1 CD0             COMMON X5 - XRMSG
397 1 CD0             C************
398 1 CD0             * XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
399 1 CD0             * THE INPUT SUBSCRIPT
400 1 CD0             BEGIN XISUB
401 2 INCREMENT TO NEXT TOKEN
402 2 ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:
403 2 IF IDIM FOR THIS ARGUMENT > 0, THEN
404 3 ERREXIT IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
405 3 INCREMENT TO NEXT TOKEN
406 3 ERREXIT IF TOKEN IS NOT COMMA :ERR15:
407 3 INCREMENT TO NEXT TOKEN
408 3 ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:
409 3 CALCULATE ISUB AS (J-1)*IDIM+1
410 3 ELSE
411 2 SET ISUB TO INTEGER VALUE
412 2 ENDIF
413 2 ERREXIT IF ISUB > X:ERR16:
414 2 INCREMENT TO NEXT TOKEN
415 2 ERREXIT IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
416 1 EXIT TO RETURN:
417 2 :ERR14: CALL XRMSG - 'INVALID SUBSCRIPT SYNTAX'
418 2 :ERR15: CALL XRMSG - 'DOWNSUBSCRIPTED - MUST SPECIFY BOTH'
419 2 :ERR16: CALL XRMSG - 'INVALID SUBSCRIPT VALUE'
420 2 :RETURN:
421 1 END XISUB
FORTRAN CALLING PROCEDURE

CALL XDAT

LITERAL DATA PROCESSOR

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XE - CFLAG, COMPLE, DFLAG, IARG, LITSTK

COMMON XE - ISIIZE, ISIIZE, LITDPS, LITDPS

LITOWN, LITOWN, MODIV, MODIV, MODBUF

MODBUF, MODBUF

INTERNAL VARIABLES

OUTPUT

COMMON XE - IRETC, ISUB, LITDPS, LITOWN, MODBUF

MODBUF, MODBUF

COMMON XE - IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS

IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS

LITERAL DATUM

LITERAL DATUM

NUMBER OF CONSECUTIVE ELEMENTS TO BE

NUMBER OF CONSECUTIVE ELEMENTS TO BE

MARKED COMPLETED AS A RESULT OF THIS

MARKED COMPLETED AS A RESULT OF THIS

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

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

DESCRIBING NESTED REPEAT GROUPS

DESCRIBING NESTED REPEAT GROUPS

EACH ENTRY IS 3 WORDS:

EACH ENTRY IS 3 WORDS:

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

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

AFTER REPEAT SYMBOL

AFTER REPEAT SYMBOL

WORD 2 = REPEAT COUNT

WORD 2 = REPEAT COUNT

WORD 3 = FLAG INDICATING WHETHER

WORD 3 = FLAG INDICATING WHETHER

REPEAT GROUP IS PARENTHESES

REPEAT GROUP IS PARENTHESES

GROUPED

GROUPED

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

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

 NOTES

USES ROUTINES

XISET

XISUB

XRBIT

XRMOV

XRMSG
IF THIS IS A DATA ELEMENT, THEN
(I.E. INTEGER, REAL, DOUBLE OR CHAR.)
ELSE IF ARG TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
ERROR IF DATA TYPE IS NOT SAME AS ITYPE :ER10:
ELSE IF ARG TYPE IS FREE, THEN
SET LENGTH TO BE MOVED (LENOV) TO EFFECTIVE LENGTH
FOR DATA INPUT
ELSE THIS MUST BE CHARACTER DATA BEING INPUT
ERROR IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ER10:
INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
COMPUTE NO. WORDS IN INPUT CHARACTER STRING
ERROR IF NO. WORDS (LENOV) > EFFECTIVE LENGTH OF
THIS ARGUMENT'S DATA (LENEFF) :ER10:
ENDIF
ENDIF
INCREMENT TOKEN POINTER TO THE DATA INPUT
VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
ARGUMENT FOR DATA INPUT (ISUB <= SIZE - LENOV + 1)
ERROR IF INSUFFICIENT SPACE :ER11:
IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
ENDIF
MOVE DATA FROM INPUT COMMUNICATIONS BUFFERS TO LITERAL AREA
SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
ARGUMENT, TO LENOV
IF LENOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
MOVE LENOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
ENDIF
INCREMENT TO NEXT TOKEN
INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENOV
SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
ELSE IF TOKEN INDICATES NON-DATA ELEMENT
IF THIS IS A "(", THEN
CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITYPE) AND
SECONDARY DIMENSION (IDIM)
EXIT XIAR IF ERROR (IREC < 0)
ELSE
IF THIS IS AN "[", THEN
IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
ELSE
INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BT EFFECTIVE LENGTH OF ONE ELEMENT (LENEFF)
ENDIF
TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
INCREMENT TO NEXT TOKEN
ELSE
  IF TOKEN IS A REPEAT COUNT, THEN
    IF NEXT TOKEN IS ',', THEN
      SET PARENFLAG TO 1
      INCREMENT TO NEXT TOKEN
    ELSE
      SET PARENFLAG TO 0
    ENDIF
  ELSE
    SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
  ENDIF
ENDIF
INCREMENT TO NEXT TOKEN
DO UNTIL TOKEN IS NOT ')
  IF PUSH-DOWN STACK IS NOT EMPTY, AND
  (PREVIOUS TOKEN WAS DATA, AND
  PARENFLAG OF TOP OF STACK ENTRY IS 0), OR
  (CURRENT TOKEN IS '), AND
  PARENFLAG OF TOP OF STACK ENTRY IS 1, THEN
  GET REPEAT COUNT OF TOP OF STACK ENTRY
  DECREMENT REPEAT COUNT BY 1
  IF REPEAT COUNT > 0, THEN
    SET TOKEN INDEX TO INDEX ON PUSH-DOWN STACK
    REPLACE NEW REPEAT COUNT ON PUSH-DOWN STACK
  ELSE
    POP (I.E. REMOVE) ENTRY FROM TOP OF STACK
    IF TOKEN IS '=', THEN
      INCREMENT TO NEXT TOKEN
    ELSE
      EXIT LOOP
    ENDIF
  ENDIF
ELSE
  IF PREVIOUS TOKEN WAS NOT A SUBSCRIPT, THEN
    IF THIS TOKEN IS NOT AN EOS, THEN
      ERREXIT IF TOKEN IS NOT A COMMA :ERROR2:
    INCREMENT TO NEXT TOKEN
  ENDIF
ENDIF
ENDDO
ENDDO
IF PROMPT MODE NOT CONTINUE (=A), AND
(THERE ARE EMPTY ELEMENTS BEYOND LASTE, OR
PREVIOUS TOKEN WAS A COMMA), THEN
RETAIN PROMPT MODE (SET MODE TO PRMTMD)
SET PROMPT MODE TO CONTINUE (=A)
ENDIF
IF ALL LITERAL SLOTS FILLED, THEN
MARK ARGNO COMPLETE
ENDIF
IF ALL ARGUMENTS ARE COMPLETE, THEN
SET COMPLETE FLAG FOR INTERFACE TABLE
ENDIF
ELSE
TURN OFF COMPLETE FLAG FOR INTERFACE TABLE
ENDIF
615  1 EXIT TO :RETURN:
616  2 :ERR02: CALL XERMSG "INVALID SYNTAX"
617  2 :ERR07: CALL XERMSG "ONLY DATA VALID TO RIGHT OF =
618  2 :ERR10: CALL XERMSG "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
619  2 :RETURN:
620  1 END XIDAT
1 * XINPT PROCESSES THE USER'S INPUT TEXT
2 BEGIN XINPT
3 IF PRMD = 5, THEN
4 SET IFLAG OFF (=0)
5 ERREXIT IF TOKEN IS NOT A NAME :ERROR2:
6 SAVE NAME AND POSITION TO NEXT TOKEN
7 IF TOKEN IS "=", THEN
8 POSITION TO NEXT TOKEN
9 IF TOKEN IS 'B', THEN
10 SET IFLAG TO 10
11 POSITION TO NEXT TOKEN
12 ELSE
13 SET IFLAG TO 0
14 ENDIF
15 ELSE IF TOKEN IS 'R', THEN
16 SET IFLAG TO 0
17 ENDIF
18 IF IFLAG NOT SET, THEN
19 CASE NAME (:EXIT, :PROMPT, :LIST): ERREXIT IF ANOTHER TOKEN FOLLOWS :ERROR2:
20 :EXIT: SET IRET 0 SO THAT PROMPTING LOOP TERMINATES
21 :PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE
22 :LIST: CALL XI LIST TO PROCESS LIST DIRECTIVE
23 ENDCASE
24 ENDIF
25 START SEARCH UNTIL IF ENTRIES
26 EXIT IF NAME FOUND IN PROMPT TABLE
27 SET AGNO TO ENTRY NO.
28 SET ISUB TO 1
29 ORELSE
30 INCREMENT TO NEXT PROMPT TABLE ENTRY
31 ENDCALL
32 ERREXIT :ERROR10:
33 ENDSHELL
34 ERREXIT IF IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:
35 ENDIF
36 IF NEXT TOKEN IS A NAME, THEN
37 CALL XIPAR TO PROCESS A PARAMETER FIELD
38 ELSE
39 ERREXIT IF IFLAG IS NOT I ("=") :ERROR8:
40 CALL XIDAT TO PROCESS DATA LIST
41 ENDIF
42 EXIT XINPT
43 EXIT TO :RETURN:
44 :ERROR2: CALL XMSG "INVALID SYNTAX"
45 :ERROR8: CALL XMSG "MUST USE PARAMETER NAME TO RIGHT OF & OR = &"
46 :RETURN:
47 END XINPT
714 1 COO        FORTRAN CALLING PROCEDURE
715 1 COO       CALL XIPAR
716 1 COO
717 1 COO
718 1 COO
719 1 COO
720 1 CO1       PROCESS AN INPUT PARAMETER NAME AND ANY ASSOCIATED SUBSCRIPT
721 1 CO1      FIELD(S)
722 1 CO1
723 1 CO1
724 1 CO1
725 1 CO2
726 1 CO2
727 1 CO2
728 1 CO2
729 1 CO2
730 1 CO2
731 1 CO2
732 1 CO2
733 1 CO2
734 1 CO3
735 1 CO3
736 1 CO3
737 1 CO3
738 1 CO3
739 1 CO3
740 1 CO5
741 1 CO5
742 1 CO5
743 1 CO5
744 1 CO5
745 1 CO5
746 1 CO5
747 1 CO5
748 1 CO5
749 1 CO5
1 XIPAR PROCSES A USER SPECIFIED PARAMETER FIELD
2 BEGIN XIPAR
3 IF A NAME IS SPECIFIED, THEN
4 INCREMENT TO NEXT TOKEN
5 IF TOKEN IS (, THEN
6 PROCESS I AND J SUBSCRIPTS
7 EXIT IF INVALID SUBSCRIPTING :ERR14:
8 IF DOUBLY SUBSCRIPTED, THEN
9 SET S-FLAG IN ARGNO'S SPECS. FIELD
10 SET LITOSP IN ARGNO'S SPECS. TO NEXT LITERAL AREA SPACE (LITOWN)
11 PUT JSUB AND JSUB INTO LITERAL AREA AT THIS SPOT
12 ELSE
13 SET LITOSP IN ARGNO'S SPECS TO JSUB
14 ENDIF
15 ELSE
16 SET LITOSP IN ARGNO'S SPECS TO 0
17 ENDIF
18 EXIT IF ORDE HAS MORE THAN A CHARACTERS :ERR18:
19 EXIT IF EXTRAN'OUS FIELD INPUT :ERR02:
20 TURN OF D-FLAG (SAME LITERAL DATA) IN ARGNO'S SPECS.
21 SET PAR'S NAME INTO ARGUMENT'S CHARACTERISTICS
22 SET COMPLETE (AND $) FLAG IN ARGUMENT'S CHARACTERISTICS
23 IF ALL ARGUMENTS ARE COMPLETE, THEN
24 SET INTERFACE TABLE COMPLETE FLAG
25 ELSE
26 EXIT :ERR02: "MUST BE A & INPUT"
27 EXIT IF NOT AN AMPERSAND ($) INPUT :ERR02:
28 CLEAR PAR'METER NAME IN ARGUMENT'S CHARACTERISTICS
29 IF ARGUMENT AND INTERFACE TABLE INCOMPLETE
30 ENDIF
31 EXIT TO :RETURN:
32 EXIT :ERR02: CALL XRMSG - "INVALID SYNTAX"
33 EXIT :ERR14: CALL XRMSG - "INVALID SUBSCRIPT SYNTAX"
34 EXIT :ERR18: CALL XRMSG - "INVALID ORDE NAME"
35 :RETURN:
36 END XIPAR
833 1 * XILSD WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
834 2 * IT IS RETURNED AS A PROMPT.
835 3 BEGIN XILSD
836 4 SET ARGUMENT NAME INTO BUFFER
837 5 USE T-FLAG TO DETERMINE WHICH OF 'B', 'N', OR 'B'
838 6 WILL GO INTO THE PRINT BUFFER
839 7 IF B-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
840 8 IF A PARAMETER NAME IS SPECIFIED, THEN
841 9 PUT PARAMETER NAME INTO BUFFER
842 10 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
843 11 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
844 12 PUT SUBSCRIPT INTO BUFFER
845 13 ELSE IF LITDSP OF ARGUMENT IS > 0, THEN
846 14 COMPUTE AND CONVERT THIS SUBSCRIPT
847 15 PUT SUBSCRIPT INTO BUFFER
848 16 ENDF
849 17 ENDF
850 18 WRITE OUT THE PRINT BUFFER BUILT
851 19 ENDF
852 20 ELSE
853 21 LOCATE LITERAL LIST AND MASK
854 22 IF SYMBOLIC STRING, THEN
855 23 CALL XILSD TO PRINT SYMBOLIC STRING
856 24 ELSE
857 25 DO UNTIL ALL ELEMENTS PROCESSED
858 26 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
859 27 DO UNTIL ALL ELEMENTS PROCESSED
860 28 COMPUTE AND CONVERT THE SUBSCRIPT
861 29 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
862 30 PUT "-" INTO BUFFER
863 31 ELSE
864 32 CONVERT THE DATA USING XR06, XR14, OR XR16
865 33 PUT DATA AND "-" INTO BUFFER
866 34 ENDF
867 35 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
868 36 UNTIL TRAILING COMMA IN THE PRINT BUFFER
869 37 ENDIF
870 38 WRITE OUT THE PRINT BUFFER BUILT
871 39 ENDF
872 40 END XILSD
873 41 END XILSD
FORTRAN CALLING PROCEDURE

CALL XILSS

C*******
XILSS IS CALLED BY XILSS TO LIST SYMBOLIC STRING DATA
C*******
C*******

INPUT

COMMON XE = LU
COMMON XB = DEBUG, LISTLY, WKBUF
COMMON X$ = BUFFER = PRINT LINE BUFFER ALREADY INITIALIZED WITH NAME =
BUF$ = INDEX INTO BUFFER OF NEXT POSITLY;
DATPTR = INDEX INTO WKBUF OF SYMBOLIC STRING DATA

C*******

OUTPUT

COMMON XS = BUFFER, BUF$PTR, DATPTR
C*******
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>904</td>
<td><strong>INTERNAL VARIABLES</strong></td>
</tr>
<tr>
<td>907</td>
<td><strong>CONTROL = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING</strong></td>
</tr>
<tr>
<td>908</td>
<td><strong>FOR EACH OF THE TOKEN VALUES 1-32, EACH CONTROL TABLE</strong></td>
</tr>
<tr>
<td>909</td>
<td><strong>ENTRY IS 3 WORDS:</strong></td>
</tr>
<tr>
<td>910</td>
<td><strong>WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER</strong></td>
</tr>
<tr>
<td>911</td>
<td><strong>WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER</strong></td>
</tr>
<tr>
<td>912</td>
<td><strong>OR FLAG DESCRIBING HOW TO COMPUTE</strong></td>
</tr>
<tr>
<td>913</td>
<td><strong>THEN</strong></td>
</tr>
<tr>
<td>914</td>
<td><strong>WORD 3 (TOKSIZ) = NO. OF WORDS IN SYMBOLIC STRING</strong></td>
</tr>
<tr>
<td>916</td>
<td><strong>ENTRY</strong></td>
</tr>
<tr>
<td>918</td>
<td><strong>TOKEN</strong></td>
</tr>
<tr>
<td>919</td>
<td><strong>WORD 1</strong></td>
</tr>
<tr>
<td>920</td>
<td><strong>WORD 2</strong></td>
</tr>
<tr>
<td>921</td>
<td><strong>WORD 3</strong></td>
</tr>
<tr>
<td>930</td>
<td><strong>TOKEN FOLLOWING BEGINNING 2 FAST</strong></td>
</tr>
<tr>
<td>931</td>
<td><strong>TOKEN</strong></td>
</tr>
<tr>
<td>932</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>933</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>934</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>935</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>936</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>937</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>938</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>939</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>940</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>941</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>942</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>943</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>944</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>945</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>946</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>947</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>948</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>949</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>950</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>951</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>952</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>953</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>954</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>955</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>956</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
<tr>
<td>957</td>
<td><strong>ERROR (INVALID) 0</strong></td>
</tr>
</tbody>
</table>
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, TOKTIZ)
EXIT TO ERROR 2 IF FIELD = 0
IF SIZE < 0, THEN
SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
ENDIF
IF TOKTIZ < 0, THEN
SET TOKTIZ 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
SET BUFFER POINTER TO 1ST POSITION FOR DATA
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 X916 WITH VALUE IN NEXT WORD OF SYMB. STRING
AND PUT RESULTS INTO PRINT BUFFER
:THREE: CALL X914 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
AND PUT RESULTS INTO PRINT BUFFER
:FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
:FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
:EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
IMPLY BY 1
CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
EXIT XILSS
ENDCASE
INCREMENT PRINT BUFFER INDEX BY SIZE
1015  1 CDO  FORTRAN CALLING PROCEDURE
1016  1 CDO
1017  1 CDO
1018  1 CDO  CALL XICHR
1019  1 CDO
1020  1 C******
1021  1 CD1  PRINT THE CHARACTERISTICS OF AN ARGUMENT
1022  1 CD1
1023  1 CD1
1024  1 C******
1025  1 CD2  INPUT
1026  1 CD2
1027  1 CD2
1028  1 CD2  COMMON XE - LU
1029  1 CD2
1030  1 CD2  COMMON XB - ARGNO, IOSTAT, ISIZE, ITYPE,
1031  1 CD2  LENEFF, NARG
1032  1 CD2
1033  1 C******
1034  1 CD5  NOTES
1035  1 CD5
1036  1 CD5  USES Routines
1037  1 CD5
1038  1 CD5
1039  1 CD5  EXEC
1040  1 CD5
1041  1 CD5  XRIOD
1042  1 CD5
1043  1 CD5  XRMW
1044  1 CD5
1045  1 C******
1046  1 * WRITE ARGUMENT CHARACTERISTICS
1047  1 BEGIN XICHR
1048  2 BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPTS, I/O TYPE AND
1049  2 DATA TYPE
1050  2 WRITE OUT THE PRINT BUFFER
1051  1 END XICHR
1053 1 CDO  FORTRAN CALLING PROEDURE
1054 1 CDO
1055 1 CDO
1056 1 CDO  CALL XIEXT
1057 1 CDO
1058 1 CDO
1059 1 C******
1060 1 C01  EXTRACT VARIOUS FIELDS OF AN ARGUMENTS CHARACTERISTICS
1061 1 C01  AND PUT VALUES INTO COMMON
1062 1 C01
1063 1 C01
1064 1 C******
1065 1 C02  INPUT
1066 1 C02
1067 1 C02
1068 1 C02  COMMON XB - ARGNO, ISIZES, WBUF
1069 1 C02
1070 1 C******
1071 1 C03
1072 1 C03
1073 1 C03
1074 1 C03  COMMON XB - CFLAG, DFLAG, IARG, IARGA,
1075 1 C03  ICLASS, IDIM, IOFLAG, ISIZE,
1076 1 C03  ISUB, ITYPE, LENEFF, LIDSP,
1077 1 C03  LITSIZE, NDIBTM, NOBTM, SFLAG
1078 1 C03
1079 1 C******
1080 1 C05
1081 1 C05  NOTES
1082 1 C05
1083 1 C05  USES ROUTINES
1084 1 C05
1085 1 C05  IAMD
1086 1 C05  XREXT
1087 1 C05
1088 1 C******
1089 1 * EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS
1090 1 * ARGUMENT
1091 1 BEGIN XIEXT
1092 2 USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
1093 2 IN THE WORKING BUFFER
1094 2 EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
1095 1 END XIEXT
1097 1 C00       FORTRAN CALLING PROCEDURE
1098 1 C00
1099 1 C00
1100 1 C00  CALL XILIT
1101 1 C00
1102 1 C00********
1103 1 C00
1104 1 C00  PACK LITERAL ENTRIES INTO FORMAT FOR STORAGE OF INTERFACE
1105 1 C00
1106 1 C00  TABLE INTO AWA
1107 1 C00
1108 1 C00********
1109 1 C00
1110 1 C00  INPUTS
1111 1 C00
1112 1 C00  COMMON XB - ARGNO, DFLAG, IARG4, ISIZE,
1113 1 C00  LENEFF, LITDSP, LITDWN, LITPTR,
1114 1 C00  LITSL, MARG, MOXBTM, MOBITM
1115 1 C00
1116 1 C00********
1117 1 C00
1118 1 C00  OUTPUTS
1119 1 C00
1120 1 C00  COMMON XB - LITDWN, LITLEM, LITPTR, WBUF
1121 1 C00
1122 1 C00********
1123 1 C00
1124 1 C00  INTERNAL VARIABLES
1125 1 C00
1126 1 C00  COMMON XS - LITUP = INDEX INTO WBUF OF AREA FOR NEXT LITERAL
1127 1 C00  ENTRY TO BE MOVED INTO
1128 1 C00  LITUP = INDEX INTO WBUF OF LITERAL ENTRY TO BE
1129 1 C00  COMPRESSED AND MOVED
1130 1 C00
1131 1 C00********
1132 1 C00
1133 1 C00  NOTES
1134 1 C00
1135 1 C00  USES ROUTINES
1136 1 C00
1137 1 C00  XIDENT
1138 1 C00  XHBIT
1139 1 C00  XRMOV
1140 1 C00  XRNXB
1141 1 C00  XRSBF
1142 1 C00
1143 1 L********
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 (LEN Eff) OF LITERAL
1156 12                     UP IN THE WORKING BUFFER
1157 13                     ENDIF
1158 14                     ENDDO
1159 15                     ENDF
1160 16                     CALCULATE NEW DISPLACEMENT AND SET IN LITOSP
1161 17                     ENDF
1162 18                     ENDDO
1163 19                     ENDDO
1164 20  1  END XILIT
RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:

:RUN, XLMAN, LU, O, ISECU, FLAGS

OFFLINE MAINTENANCE PROGRAM, XLMAN CREATES, DELETES AND MODIFIES FDS FILES.

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

INTERNAL VARIABLES:

PROMT - (INTEGER, 31 WORDS) PROMPT FOR OPTION TO BE EXECUTED

TEMP - (INTEGER, 1 WORD) TEMPORARY USED FOR RESPONSE

TOKENZ - (INTEGER, 29 WORDS) THE 29 TOKENS TO BE INITIALIZED INTO COMMON

RTE FUNCTION AND SUBROUTINES USED:

EXEC, RMPAR

FDS FUNCTIONS AND ROUTINES USED:

XELDS, XLDIF, XDEL, XLMNT, XLMOD, XLMSG,

XLPRM, XLP, XTPM, XTP, XRMG

XE COMMON USED:

(EQUIVALENCE (XE(1), LU ),

(XE(2), ICLASS), (XE(3), ISECU ),

(XE(4), FLAGS ), (XE(5), MASS ),

(XE(6), SUBSTA), (XE(?), NUMER ),

(XE(?), VALFLG), (XE(85), TOKENS ),

(XE(142), ICR ), (XE(145), COMBUF)
55 1 BEGIN XLMAN
56 2 CALL RMPAR TO GET INPUT PARAMETERS
57 2 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 SUBSTATE 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 XRMSE TO WRITE INVALID RESPONSE
70 4 GO TO :PROMPT:
71 3 ENDIF
72 3 CASE INTEGER (:XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM::XLPRM:: XL
103 1 CD************
104 1 CD0
105 1 CD0  FORTRAN CALLING PROCEDURE:
106 1 CD0
107 1 CD0
108 1 CD0
109 1 CD0
110 1 CD0
111 1 CD0  CALL XELDS ('XLPRM')
112 1 CD0
113 1 CD0
114 1 CD0
115 1 CD0
116 1 CD0
117 1 CD0
118 1 CD0
119 1 CD0
120 1 CD0
121 1 CD0
122 1 CD0
123 1 CD0
124 1 CD0
125 1 CD0  INTERNAL VARIABLES:
126 1 CD0
127 1 CD0
128 1 CD0
129 1 CD0
130 1 CD0
131 1 CD0
132 1 CD0
133 1 CD0
134 1 CD0
135 1 CD0
136 1 CD0
137 1 CD0
138 1 CD0
139 1 CD0
140 1 CD5
141 1 CD5
142 1 CD5
143 1 CD5
144 1 CD5
145 1 CD5
146 1 CD5
147 1 CD5
148 1 CD5
149 1 CD5
150 1 CD5
151 1 CD5
152 1 CD5
153 1 CD5
154 1 CD5
155 1 CD5
156 1 CD5
157 1 CD5
158 1 CD5
160 1 BEGIN XLPRM
161 2 :PRM1:
162 2 CALL IXCOM TO PROMPT USER FOR 0 DIRECTIVES
163 2 ERREXIT IF RESPONSE NOT INTEGR 1-63 TO :ERR1:
164 2 CALL CREAT TO CREATE PROMPT FILE
165 2 IF FILE ALREADY EXISTS THEN
166 2 CALL IXCOM 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 :FILERR:
170 2 GO TO :PRM1:
171 2 ELSE (CREATE NEW FILE)
172 2 ERREXIT IF CREAT ERROR TO :FILERR:
173 2 CALL IXCOM TO PROMPT USER FOR LIST OF DIRECTIVES
174 2 ERREXIT IF LIST IS INCONSISTENT WITH 0 OF DIRECTIVES TO :ERR1:
175 2 DO FOR EACH DIRECTIVE
176 2 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
177 2 CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
178 2 ENDDO
179 3 CALL WRITF TO WRITE LIST OF DIRECTIVES TO FILE
180 3 ERREXIT IF WRITF ERROR TO :FILERR:
181 3 DO FOR EACH DIRECTIVE
182 4 CALL IXCOM TO PROMPT USER FOR DEFINITION
183 4 DO UNTIL EOS IS REACHED IN RESPONSE
184 5 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185 5 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186 5 MOVE RESPONSE INTO BUFFER
187 5 SET CONTROL CHARACTERS IN BUFFER
188 5 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
189 5 ENDDO
190 4 SET REMAINDER OF BUFFER TO NULL
191 4 CALL WRITF TO WRITE DEFINITION
192 4 ERREXIT IF WRITF ERROR TO :FILERR:
193 4 ENDIF
194 3 CALL CLOSE TO CLOSE FILE
195 3 ERREXIT IF CLOSE ERROR TO :FILERR:
196 3 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
197 2 ENDIF
198 1 EXIT XLPRM
199 2 :ERR1: (ERROR IN RESPONSE)
200 3 CALL XMSG TO DISPLAY ERROR IN RESPONSE
201 3 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
202 2 :FILERR: (FILE ACCESS ERROR)
203 3 CALL XMSG TO DISPLAY ERROR
204 3 CALL CLOSE TO CLOSE FILE
205 3 CALL PURGE TO PURGE FILE
206 1 END XLPRM
208 1 CD************
209 1 CD0
210 1 CD0  FORTRAN CALLING SEQUENCE:
211 1 CD0  CALL XEFD ('XLPRO')
212 1 CD0
213 1 CD0************
214 1 CD1  XEFD WILL ADD A PROCESSOR TO THE LIBRARY DIRECTORY. IF,
215 1 CD1  THE PROCESSOR HAS A DEFAULT INTERFACE TABLE, IT WILL ALSO
216 1 CD1  CREATE A PROMPT FILE AND INTERFACE TABLE FILE.
217 1 CD0
218 1 CD0************
219 1 CD1  INPUT FROM COMMON:
220 1 CD1  VALFLG = PROCESS CONTROL
221 1 CD1  0 - ORIGINAL REQUEST TO ADD
222 1 CD1  1 - COMPLETE PROCESSING
223 1 CD0
224 1 CD0************
225 1 CD1  INTERNAL VARIABLES:
226 1 CD1  ABSTR = (INTEGER, 128 WORDS) CONTAINS THE ABSTRACT OF THE
227 1 CD1  PROCESSOR IN LATER BUILDS. PRESENTLY IS A NULL RECORD.
228 1 CD1  BLOCKS = (INTEGER, 1 WORD) NUMBER OF BLOCKS TO ALLOCATE
229 1 CD1  TO INTERFACE TABLE FILE
230 1 CD1  DIRECT = (INTEGER, 6 WORDS) 3 NAME ARRAY CONTAINING VALID
231 1 CD1  DIRECTIVES FOR THE INTERFACE TABLE EDITOR
232 1 CD1  NAMEY = (INTEGER, 3 WORDS) COMBINATION OF PROCESSOR
233 1 CD1  NAME AND VERSION NUMBER
234 1 CD1  PRNAME = (INTEGER, 3 WORDS) PROCESSOR NAME
235 1 CD1  PROMS = (INTEGER) PROMPTS FOR USER TO BE PROMPTED WITH
236 1 CD1  TYPE = (INTEGER, 1 WORD) TYPE OF PARAMETER USED IN
237 1 CD1  CALCULATING SIZE
238 1 CD1  VERS = (INTEGER, 1 WORD) VERSION NUMBER OF PROCESSOR
239 1 CD0
240 1 CD0************
241 1 CD2  RTE AND FNR ROUTINES USED
242 1 CD2  EXEC, IAND, ECVT, CREATE, OPEN,
243 1 CD2  READ, WRIT, CLOSE, PURGE
244 1 CD2************
245 1 CD3  FDS ROUTINES USED:
246 1 CD3  XLIFL, XLIMS, XLFL, XLRPS
247 1 CD3  XREP, XREM, XRM5, XRC6, XRSET, XRUNK, XCTCN
248 1 CD3************
249 1 CD4  COMMON USED:
250 1 CD4  EQUIVALENCE
251 1 CD4  (XE(3), ISECU ),
252 1 CD4  '+ (XE(3), MAS3A), (XE(4), SUSTA),
253 1 CD4  '+ (XE(7), WFL6G), (XE(16), PRCH1),
254 1 CD4  '+ (XE(85), EOSTOK), (XE(86), IINTOK),
255 1 CD4  '+ (XE(87), NAROTK), (XE(113), CONTOK),
256 1 CD4  '+ (XE(142), ZCR ), (XE(145), COMUFF),
257 1 CD4  '+ (XB(26), VERS ), (XB(37), ARMO ),
258 1 CD4  '+ (XB(36), IT ), (XB(90), LITP1T),
259 1 CD4
1 BEGIN XLPRO
2 IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
3 INITIALIZE MASTER AND SUB STATES
4 IF PROCESSOR HAS AN INTERFACE TABLE THEN
5 SET RETN = 2
6 CALL XTCON TO PROMPT FOR # PARAMETERS
7 ERREXIT IF XTCON RETURN CODE IS NOT ZERO OR
8 ERREXIT IF # PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
9 CREATE HEADEP ENTRY WITH #PARAMETERS AND PROCESSOR NAME
10 CALL XLPS TO WRITE INSTRUCTIONS FOR ENTERING SPECS
11 DO FOR # PARAMETERS
12 CALL XLPS TO CREATE ONE PARAMETER ENTRY
13 ENDDO
14 SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
15 CALL XLPSL TO CREATE PROMPT FILE
16 SET RETN = 3
17 :PRMPRT:
18 CALL XTCON FOR DEFAULT VALUES DECISION
19 ERREXIT IF RETURN CODE IS NOT ZERO OR
20 ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
21 IF RESPONSE WAS YE THEN
22 SET FLAG TO CALL INTERFACE TABLE EDITOR
23 EXIT XLPRO
24 ENDIF
25 ELSE
26 SET CODES ARRAY TO ADD ONLY ABSTRACT
27 CALL XLPSL TO CREATE PROMPT FILE
28 PERFORM XLPRO - NO RETURN EXPECTED
29 ENDIF
30 END
31 ELSE
32 CALL XLPSL TO CREATE THE DEFAULT INTERFACE TABLE FILE
33 SET OSTATE TO SET ORIIGINAL REQUEST TO ADD A PROCESSOR
34 PERFORM XLPRO - NO RETURN EXPECTED
35 :PRMERR:
36 CALL XRMG TO DISPLAY ERROR MESSAGE
37 GO TO (:PRMPRT1:,;PRMPRT2:,;PRMPRT3:), RETN
38 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 XRMOV TO WRITE ERROR: TOO MANY PROCESSORS
11 EXIT XLPRO
12 ENDF
13 SET RETN = 1
14
15 :PROMPT:
16 CALL XTCOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
17 EXIT XLPRO IF RETURN CODE SAYS 1 ENTERED
18 ERREXIT IF RETURN CODE > ZERO OR
19 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
20 CALL XRMOV TO MOVE PROCESSOR NAME INTO ENTRY
21 ERREXIT IF VERSION IS NOT INTEGER VALUE 0-127 TO :PRMERR:
22 CALL XRSET TO SET VERSION IN ENTRY
23 ERREXIT IF INTERFACE TABLE OPTION IS NOT YE OR NO TO :PRMERR:
24 SET IT BIT = 0
25 IF RESPONSE IS YES THEN
26 SET IT BIT = 1
27 ENDF
28 CALL XRSET TO SET BIT ON/OFF
29 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
30 INCREMENT # PROCESSORS BY 1
31 CALL XRMOV TO MOVE NEW ENTRY INTO XLIBD
32 IF # PROCESSORS > 1 THEN
33 CALL PURGE TO PURGE OLD FILE
34 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
35 ENDF
36 CALL CREATE, WRITE, AND CLOSE TO CREATE NEW LIBRARY DIRECTORY
37 ERREXIT IF FILE ERROR TO :FILERR:
38 ENDF
39 1 END LIBD
Fortran Calling Sequence:

CALL XLCDB

XLCDB creates a new data base file (MDB/PDB) from an old data base file (MDB/PDB) and deletes the old file.

Inputs in Common:

XE(3) QUAL
XE(7) NUMBR
XE(8) SECU
XE(142) INC
XB(5) OLDFIL
XB(6) NEWMIL
XB(9) TOTSIZ

Internal Variables:

XB(17) FREC - First record number in datbuf
XB(18) LREC - Last record number in datbuf
XB(40) IDC8 - DCB for OLDFIL
XB(56) IDC82 - DCB for NEWMIL
XB(200) TOCBUF-BUFFER FOR COMPLETE DATA BASE TOC

RTE Routines Used:

CLOSE, CREAT, KCVT, OPEN
PURGE, READF, WRITF

FDS Routines Used:

XDBDB, XREXT, XRMSG

Common Used:

Equivalence (XE(3), QUAL), (XE(7), NUMBR), (XE(8), SECU),
(+XE(142),INC), (XB(3), OLDFIL),
(+XB(6), NEWMIL), (XB(9), TOTSIZ),
(+XB(10), FILCHR), (XB(12), QLCFIL),
(+XB(13), MOTOC), (XB(14), SIZE),
(+XB(15), RSE), (XB(16), TOCPR),
(+XB(17), FREC), (XB(18), LREC),
(+XB(39), JERR), (XB(40), IDC8),
(+XB(41), IDC82), (XB(72), IDUP),
(+XB(200), TOCBUF)
10   BEGIN XLCDB
11   CALL OPEN TO OPEN OLD FILL
12    ERREXIT IF OPEN ERROR TO "ERR3;"
13    CALL READ TO READ IN HEADER RECORD
14    ERREXIT IF READ ERROR TO "ERR2;"
15    COMPUTE NUMBER OF TOC RECORDS (NOTOC) AS (# ENTRIES + 16)/16
16    IF NOTOC > 1 THEN
17    CALL READ TO READ IN REMAINING TOC RECORDS
18    ERREXIT IF READ ERROR TO "ERR2;"
19    ENDIF
20    SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
21    CALL CREAT TO CREATE NEW FILL
22    ERREXIT IF CREAT ERROR TO "ERR2;"
23    DO FOR EACH TOC ENTRY
24      IF REQUEST WAS FOR TOC THEN
25        ERREXIT IF NAME > 4 CHARS OR
26        ERREXIT IF CLASS IS DNE AND NAME > 2 CHARS TO "ERR1;"
27        APPEND DNE CONVENT TO FRONT OF NAME
28      ELSE
29        REMOVE DNE CONVENTION FROM NAME
30      ENDIF
31    ENDDO
32    CALL WRITE TO WRITE NEW TOC RECORDS TO NEW FILL
33    ERREXIT IF WRITE ERROR TO "ERR2;"
34    SET TOCPTR TO FIRST TOC ENTRY
35    SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
36    CONV SIZE FROM BLOCKS TO WORDS
37    DO UNTIL ALL RECORDS ARE CITED (SIZE = 0)
38      SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
39      IF SIZE < LENGTH THEN
40        SET LENGTH TO SIZE
41      ENDIF
42      CALL READ TO READ LENGTH WORDS OF DATA
43      ERREXIT IF READ ERROR TO "ERR2;"
44      DECREMENT SIZE BY LENGTH READ
45      UPDATE FIRST AND LAST RECORD NUMBERS
46      START SEARCH FOR TOCPTR = TOCPTR TO LAST TOC ENTRY
47      EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
48      IF CLASS IS AN INTERFACE TABLE THEN
49        COMPUTE INDEX INTO DATABUF FROM FIRST RECORD #, RECORD # IN TOC ENTRY
50        AM# INDEX IN TOC ENTRY
51        SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
52      ENDIF
53      ENDS EARCH
54      CALL WRITE TO WRITE LENGTH WORDS OF DATA TO NEW FILL
55      ERREXIT IF WRITE ERROR TO "ERR2;"
56      ENDDO
CALL CLOSE TO CLOSE NEWFIL
CALL CLOSE TO CLOSE OLDFIL
CALL XDDBD TO DELETE OLDfil FROM PDB LOG FILE
CALL PURGE TO PURGE OLDfIL FROM SYSTEM
EXIT XLCDB

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

:ERR2:
CALL CLOSE TO CLOSE OLDfIL

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

CALL XEBF ('XEBF')

XEBF IS THE DATA BASE FILES MAIN ROUTINE. IT DETERMINES WHICH
DATA BASE ROUTINE IS NEEDED AND GATHERS INPUTS NEEDED BY THAT
ROUTINE. HANDLES ALL DBF 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 XRFWM
XE(8) SECU - TEMPORARY SLOT FOR ISECU DURING THIS OVERLAY
XB(5) OLDFIL - OLD FILE NAME
XB(6) NEWFIL - NEW FILE NAME
XB(9) TOTSIZ - SIZE OF OLDFIL/NEWFIL IN BLOCKS
XB(10)FILCHR - 4 BASE CHARACTERS OF FILE NAME
XB(12)QUALIF - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPENW

FDS ROUTINES USED:

XDBRA, XDBW, XERW, XLCDX, XLPCH

COMMON USER:

ENQUIVALENCE (XE(3), ISECU),
+XE(3) QUAL , (XE(7), NUMBR ),
+XE(8) SECU , (XE(85), TOKENS),
+XE(142), ICR , (XE(145) COMDF),
+XB(5), OLDFIL , (XB(6), NEWFIL),
+XB(9), TOTSIZ , (XB(10), FILCHR),
+XB(12), QUALIF , (XB(99), IERR ),
+XB(100), IEDCB )
1 BEGIN XDBF:
2 IF REQUEST IS TO CREATE/MODIFY LOG FILE THEN
3 CALL OPEN TO OPEN MDB/PDB LOG FILE
4 IF OPEN ERROR SAYS FILE NOT FOUND THEN
5 CALL XLPGR TO CREATE MDB/PDB LOG FILE
6 ELSE
7 ERREXIT IF OPEN ERROR TO :FILEERR:
8 CALL XLPMD TO MODIFY MDB/PDB LOG FILE
9 ENDIF
10 ELSE
11 DO UNTIL USER REQUESTS EXIT (")
12 CALL XCOM TO PROMPT USER FOR NAME AND USER ID
13 IF RESPONSE IS NOT EXIT (PERCENT) THEN
14 ERREXIT IF RESPONSE IS INVALID TO :ERR1:
15 SAVE A CHARACTER NAME AND ID IN COMMON
16 IF REQUEST WAS PDB TO MDB THEN
17 SET QUALIFIER TO SEARCH FOR PDB NAME
18 ELSE (REQUEST WAS FOR MDB TO PDB)
19 SET QUALIFIER TO SEARCH FOR MDB NAME
20 ENDIF
21 CALL XDDBV TO VERIFY EXISTENCE OF MDB/PDB DEPENDING ON QUALIFIER
22 ERREXIT IF NAME WAS NOT FOUND TO :ERR1:
23 ERREXIT IF FILE MANAGER ERROR TO :FILEERR:
24 IF REQUEST WAS PDB TO MDB THEN
25 SET QUALIFIER TO ADD PDB TO LOG FILE
26 ELSE (REQUEST WAS MDB TO PDB)
27 SET QUALIFIER TO ADD PDB TO LOG FILE
28 ENDIF
29 CALL XDDBA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
30 ERREXIT IF DUPLICATE NAME OF
31 ERREXIT IF MAXIMUM NUMBER OF ENTRIES EXIST TO :ERR1:
32 ERREXIT IF FILE MANAGER ERROR TO :FILEERR:
33 IF REQUEST WAS PDB TO MDB THEN
34 CALL XRDFM TO SET OLDfil TO PDB NAME
35 SET NEWFIL TO PDB NAME
36 ELSE (REQUEST WAS MDB TO PDB)
37 SET OLDfil TO MDB NAME
38 CALL XRDFM TO SET NEWFIL TO PDB NAME
39 ENDIF
40 CALL XLCDB TO COPY OLDfil TO NEWFIL
41 ENDIF
42 ENDIF
43 END; 1 EXIT XDBF

2 :ERR1:
3 CALL XRMSG TO DISPLAY ERROR
4 RETURN TO PROMPT FOR ANOTHER 4 CHARACTERS AND USER ID
5 :FILEERR:
6 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
7 1 END XDBF
1 BEGIN XLDEL

2  :PRMPT:
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 :PRMPT:
8   EXIT XLDEL IF RETURN CODE SAYS % ENTERED
9   ENSIF
10   CALL OPEN, READ AND CLOSE TO READ IN LIBRARY DIRECTORY
11   ERREXIT IF FILE ERROR TO :FILERR:
12   IF PROCESSOR IS NOT IN LIBRARY DIRECTORY THEN
13     CALL XRMSG TO DISPLAY ERROR
14   GO TO :PRMPT:
15   ENSIF
16   DECIMATE # PROCESSORS BY 1
17   EXIT XLDEL IF #PROCESSORS IS ZERO
18   CALL CREATE, WRITF AND CLOSE TO RECREATE LIBRARY DIRECTORY
19   ERREXIT IF FILE ERROR TO :FILERR:
20   CREATE THE PROMPT FILE NAME
21   CALL PURGE TO PURGE THE PROMPT FILE
22   ERREXIT IF PURGE ERROR TO :FILERR:
23   CALL XRMSG TO SAY FILE PURGED SUCCESSFULLY
24   IF PROCESSOR HAD AN INTERFACE TABLE THEN
25     CREATE DEFAULT IT NAME
26     CALL PURGE TO PURGE DEFAULT IT
27     ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:
28     CALL XRMSG TO DISPLAY 'FILE :AGED' MESSAGE
29   ENSIF
30   GO TO :PRMPT:
31   .:FILERR: CALL XRMSG TO WRITE FILE ACCESS ERROR
32   1 END XLDEL
1 CD**********
2 CD
3 CD FORTRAN CALLING SEQUENCE:
4 CD
5 CALL XLIFL
6 CD**********
7 CD
8 CD XLIFL CREATES THE DEFAULT INTERFACE TABLE FILE
9 CD
10 CD**********
11 CD
12 CD INTERNAL VARIABLES
13 CD
14 CD BLOCKS - (INTEGER, 1 WORD) # BLOCKS TO BE ALLOCATED TO
15 CD THE FILE
16 CD
17 CD**********
18 CD
19 CD FDS ROUTINES USED:
20 CD
21 CD XEXIT, XMSG
22 CD
23 CD RTE ROUTINES USED:
24 CD
25 CD CLOSE, CREAT, WRITF
26 CD
27 CD**********
28 CD
29 CD COMMON USED:
30 CD
31 CD
32 CD EQUIVALENCE (X(E(3)), ISECU),
33 CD + (X(E(142), ICR ), (X(E(90), LITPR),
34 CD + (X(E(91), LITLEN),
35 CD + (X(E(96), IOPARM), (X(E(101), MEDP ),
36 CD + (X(E(108), PARMS ), (XS(1), IERR ),
37 CD + (XS(2), I0CB )
38 CD
39 CD**********
CALL XLINS

XLINS DISPLAYS INSTRUCTIONS FOR ENTERING PARAMETER SPECS

INTERNAL VARIABLES:

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

RTE ROUTINES USED:

EXEC

COMMON USED:

EQUIVALENCE (RE(1), LU )

BEGIN XLINS

CALL EXEC TO DISPLAY ALL 7 LINES

END XLINS
FORTRAN CALLING PROCEDURE:

CALL XELDS (XLINT)

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

FDS FUNCTIONS AND SUBROUTINES USED:

XEINT, XERMT, XINIX, XRMOV

COMMON USED:

EQUIVALENCE (KE(5), MASSTA),
+(KE(6), SUBSTA), (KB(1), NUMDR),
+(KE(7), DIRECT), (KB(25), LSTFLG),
+(KE(37), ARCH), (KB(41), PRINTR),
+(KE(73), ISIZE), (KB(89), LITDWM),
+(KE(90), LITPR), (KB(91), LITLEN),
+(KE(92), NAMG),
+(KE(96), NUMARG), (KB(97), HEMTAB),
+(KE(100), WRMLNG), (KB(101), WRBUF),
+(KE(1400), EMB)
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, WRIT

FDS ROUTINES USED:

XLI, XLI4, XLPIL, XLSIP, XRPCB, XREXT,

COMMON USED:

EQUIVALENCE (XE(3), ISECU),

* (X(5), NASSA, XE(6), SUBSTA),

* (X(7), VALTH0), (XE(135), PMH1AM),

* (X(105), PREFH1A), (XE(85), EOSTK),

* (X(86), INTOK), (XE(69), MANTK2),

* (X(97), MEGTO), (XE(142), ECH),

* (X(145), CORPUS), (XB(261), VERS),

* (X(33), SYFLG), (XB(35), SYFLAG),

* (X(37), ARCNO), (XK(38), XIT),

* (X(43), LEM), (XK(44), NEGLFLG),

* (X(88), LITBSP), (XK(90), LITPBR),

* (XG(92), INOK), (XK(96), NODEP),

* (XG(101), HEBR), (XK(105), LEB01),

* (XG(104), LIB02), (XK(109), PARMS)
BEGIN XLMOD
150  IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
151  SET RTH = 1
152
153  PROMP:
154  CALL XLIN TO DISPLAY SPEC INSTRUCTIONS
155  INITIALIZE MASTER AND CURRENT STATE Flags
156  CALL XTRCP TO PROMPT FOR PROCESSOR NAME
157  EXIT XLMOD IF RETURN CODE SAYS 1 ENTERED
158  ERREXIT IF XICON RETURN CODE NON-ZERO OR
159  ERREXIT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO :PROMP:
160  CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
161  IF THERE IS A FILE ERROR TO :FILERR:
162  ERREXIT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO :PROMP:
163  SAVE INTERFACE TABLE BIT AND VERSION NUMBER
164  PERFORM VERSION TO UPDATE VERSION NUMBER
165  IF THE PROCESSOR HAD AN INTERFACE TABLE THEN
166  SET NEW VERSION NUMBER IN INTERFACE TABLE
167  CREATE DEFAULT INTERFACE TABLE NAME
168  CALL OPEN AND READF TO READ IN HDR AND SPECs
169  IF THERE ARE LITERALS THEN
170  CALL READF TO READ IN LITERALS
171  ENDIF
172  CALL CLOSE TO CLOSE FILE
173  ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
174  CREATE PROMPT TABLE NAME
175  CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
176  ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
177  CALL NAME TO REUSE PROMPT FILE > XLMP
178  ERREXIT IF NAME ERROR TO :FILERR:
179  SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPECs
180  PERFORM DELPRM TO DELETE PARAMETERS
181  PERFORM MODPRM TO MODIFY PARAMETERS
182  PERFORM ADDPRM TO ADD PARAMETERS
183  CALL XLPLF TO CREATE NEW PROMPT FILE
184  PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
185  ELSE
186  CALL NAME TO REUSE PROMPT FILE > XLMP
187  ERREXIT IF NAME ERROR TO :FILERR:
188  SET CODES ARRAY TO MODIFY ABSTRACT ONLY
189  CALL XLPLF TO CREATE NEW PROMPT FILE
190  PERFORM XLMOD - NO RETURN EXPECTED
191  ENDIF
192
193  CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
194  ERREXIT IF FILE ERROR TO :FILERR:
195  CALL XLFL TO CREATE NEW DEFAULT INTERFACE TABLE FILE
196  SET VALFILE TO SAY ORIGINAL REQUEST TO MODIFY
197  PERFORM XLMOD - NO RETURN EXPECTED
198  :PROMP:
199  CALL XRMG TO DISPLAY ERROR MESSAGE
200  GO TO (:PROMP1:.PROMP2:.PROMP3:.PROMP4:.PROMP5:.PROMP6:.PROMP7):RTH
201  :FILERR:
202  CALL XRMG TO DISPLAY FILE ERROR
203  END XLMOD

REPRODUCIBILITY OF THE
ORIGINAl DAE IS POOR
BEGIN VERSION
SET RTM = 2

:PRIM2:
CALL Xicom TO PROMPT USER FOR VERSION NUMBER
IF RETURN CODE IS NOT CR THEN
ERROR IF RETURN CODE IS NON-ZERO ON
ERROR IF VERSION IS INVALID TO :PRIM2:
IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
CALL XRSET TO PUT NEW VERSION IN ENTRY
CALL OPEN, WRITF, CLOSE TO UPDATE LIBRARY DIRECTORY
ERROR IF FILE ERROR TO :FILEERR:
ENDIF
1 END VERSION
2 END PRIM
3 BEGIN DELPRM
4 SET RTM = 3

:PRIM3:
DO UNTIL RETURN CODE IS CR ENTERED
IF NUMBER OF PARAMETERS > 1 THEN
CALL Xicom TO PROMPT FOR DELETE PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPMD TO INTERPRET RESPONSE
PERFORM CHDAD TO CHECK FOR EXISTING DATA
SET ARCGN TH WORD IN CODES TO SAY 'DELETED'
CALL XREM TO MOVE DATA TO DELETE PARAMETER
DECREMENT # PARAMETERS BY 1
ENDIF
ELSE
CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
EXIT DELPRM
ENDIF
2 END DELPRM
1 END MODPRM
4 SET RTM = 2

:PRIM4:
DO UNTIL RETURN CODE IS CR ENTERED
CALL Xicom TO PROMPT FOR MODIFY PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPMD TO INTERPRET RESPONSE
PERFORM CHDAD TO CHECK FOR EXISTING DATA
SET ARCGN TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
CALL CLSFX TO PROMPT USER FOR SPECIFICATIONS
SET IT COMPLETE BIT OFF
ENDIF
2 END MODPRM
1 END MODPRM
1 BEGIN RSPNDO
2 ERREXIT IF RETURN CODE IS NON-ZERO TO :PRMRR:
3 SET NEGATIVE FLAG OFF
4 IF NEGATIVE IS REQUESTED THEN
5 SET NEGATIVE FLAG ON
6 ENDEF
7 ERREXIT IF PARAMETER NAME NOT IN INTERFACE TABLE TO :PRMRR:
8 CONVERT PARAMETER NAME TO ARGUMENT NUMBER
9 ENDRSPNDO
10 * 1
11 * 1
12 * 1
13 BEGIN CHOT
14 EXTRACT SFLAG, CFLAG AND DISP FROM SPEC ENTRY
15 IF DOUBLE SUBSCRIPT FLAG IS ON THEN
16 SET LITERAL ENTRY LENGTH TO 2
17 ELSE
18 IF THERE IS COMPLETE LITERAL DATA THEN
19 SET LITERAL ENTRY LENGTH TO SIZE
20 ENDF
21 ENDF
22 ENDRCHOT
CD************
CD CALLING PROCEDURE
CD CALL ("XMSG")
CD************
CD1 XMSG PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XMSG
CD************
CD2 INPUT
CD2 COMMON - LU, ISEQU, FLAGS, TOKENS, ICR
CD2 TERMINAL - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES
CD2 UPDATE MODE, MESSAGE NUMBER AND TEXT
CD2 MESSAGE FILE - DIRECTORY AND OLD TEXT
CD************
CD3 OUTPUT
CD3 COMMON - COMBUF
CD************
CD3 MESSAGE FILE - DIRECTORY AND TEXT UPDATES
CD************
CD4 LOCAL
CD4 AREA - NUMERICAL AREA INDICATOR FOR MESSAGE
CD4 DIRECT - MESSAGE DIRECTORY (SEE SD 6.2.4.12)
CD4 INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY
CD4 FILE MANAGER DATA CONTROL BLOCK
CD4 FILE MANAGER & XCOM RETURN CODE
CD4 MESSAGE NUMBER WITHIN MESSAGE AREA
CD4 MESSAGE BLOCK NUMBER WITHIN FILE
CD4 CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION
CD4 UPDATE MODE - NUMBER OF BLOCKS CONTAINING MESSAGE
CD4 MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 33, 65 OR 97)
CD************
CD5 NOTES
CD5 USES APDSW, CLOSE, CREATE, EXEC, IAND, KCVT, OPEN, READ, WRTF,
CD5 XERIN, XER, XRMOV, XRMSG, XCOM, XUDBG
CD5 WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE
CD5 EXISTING TEXT IN PLACE.
CD5 MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X
1067 1 BEGIN XLMSG
1068 2 OPEN XLMSG
1069 3 IF FILE NOT FOUND
1070 4 THEN
1071 5 OUTPUT 'MESSAGE FILE CREATION'
1072 6 DO FOR EACH OF THE 32 DIRECTORY ENTRIES
1073 7 PROMPT FOR AREA ID AND MAXIMUM NUMBER OF MESSAGES FOR THIS ENTRY NUMBER
1074 8 IF RESPONSE WAS NULL
1075 9 THEN
1076 10 CLEAR ENTRY
1077 11 ELSE
1078 12 STORE ID
1079 13 COMPUTE AREA ORIGIN AND STORE
1080 14 CLEAR LAST MESSAGE NUMBER
1081 15 COMPUTE NUMBER OF BLOCKS AND STORE
1082 16 ENDIF
1083 17 ENDDO
1084 18 CREATE A CLEARED FILE OF TOTAL REQUIRED SIZE
1085 19 ELSE
1086 20 READ DIRECTORY
1087 21 ENDIF
1088 22 DO UNTIL USER INPUTS 'X'
1089 23 PROMPT FOR MESSAGE NUMBER
1090 24 SEPARATE AREA AND MESSAGE NUMBER AND COMPUTE BLOCK NUMBER AND MESSAGE LOC
1091 25 IF VALID AREA AND BLOCK NUMBER <= NUMBER OF BLOCKS
1092 26 THEN
1093 27 READ BLOCK
1094 28 IF FIRST WORD OF MESSAGE IS NOT NULL (MESSAGE ALREADY EXISTS)
1095 29 THEN
1096 30 DISPLAY OLD MESSAGE TEXT
1097 31 ENDIF
1098 32 PROMPT FOR TEXT
1099 33 IF NON-NULL RESPONSE
1100 34 THEN
1101 35 STORE TEXT IN BLOCK
1102 36 REWRITE BLOCK
1103 37 IF MESSAGE NUMBER > LAST MESSAGE NUMBER
1104 38 THEN
1105 39 REPLACE LAST MESSAGE NUMBER WITH NEW NUMBER
1106 40 ENDIF
1107 41 ENDIF
1108 42 ELSE
1109 43 OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110 44 ENDIF
1111 45 ENDDO
1112 46 REWRITE DIRECTORY BLOCK
1113 1 END XLMSG
BEGIN XL_CR
CALL CREAT TO CREATE MDB/PDB LOG FILE
ERREXIT IF CREATE ERROR TO :FILERR:
INITIALIZE LOG RECORD BUFFER TO ZEROS
SET # MDB FILES CURRENTLY USED TO ZERO
SET MAXIMUM NUMBER MDBS TO 20
CALL WRITE TO WRITE MDB RECORD TO LOG FILE
ERREXIT IF WRITE ERROR TO :FILERR:
DO FOR EACH REMAINING LOG RECORD
DO FOR THIS PAIR OF USER ID'S
CALL XCTO TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
SET MAXIMUM # PDB FILES TO RESPONSE
SET # PDB FILES CURRENTLY USED TO ZERO
ENDDO
CALL WRITE TO WRITE 1 PDB RECORD TO LOG FILE
ERREXIT IF WRITE ERROR TO :FILERR:
ENDDO
CALL CLOSE TO CLOSE FILE
ERREXIT IF CLOSE ERROR TO :FILERR:
EXIT XLPCR

:FIERR:
CALL XRMSG TO DISPLAY FILE ERROR
CALL CLOSE TO CLOSE FILE
END XLPCR
FORTRAN CALLING SEQUENCE:

CALL XLPFL (MOCOD, CODES)

XLP/L CREATES PROMPT FILE FOR PROCESSORS. IT ALSO RE-CREATES IT WHEN A PROCESSOR IS MODIFIED.

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XE(1) LU, XE(3) ISECU, XE(16) PROCNAM,
XE(142) IC, XB(96) NOPARM, XB(108) PARMS

RTE FUNCTIONS USED:

CLOSE, CREAT, EXEC, OPEN,
POSN, PURGE, READF, WRITF

FDS FUNCTIONS USED:

XERTN, XRCPR, XPMOV, ZRMNG,
XRCPK, XRUPK, XCOM

COMMON USED:

EQUIVALENCE (XE(1), LU),
+ (XE(3), ISECU), (XE(7), VALFLG),
+ (XE(16), PROCNAM), (XE(85), TOKEM3),
+ (XE(142), IC), (XE(144), COMPRH),
+ (XE(145), COMBUP), (XB(96), NOPARM),
+ (XB(108), PARMS)
1227 1 BEGIN XLPL
1228 2  COMPUTE SIZE OF FILE AS # PARAMETERS +3
1229 3  CALL CREAT TO CREATE PROMPT FILE
1230 4  ERREXIT IF CREAT ERROR TO :FILERR:
1231 4  STUFF SYNTAX RECORD # AND # PARAMETERS INTO LIST OF SHORT PROMPTS
1232 4  CALL WRITF TO WRITE SHORT PROMPT RECORD(S)
1233 4  ERREXIT IF WRITF ERROR TO :FILERR:
1234 4  IF ABSTRACT CODE IS MODIFY THEN
1235 3  CALL OPEN TO OPEN OLD PROMPT FILE >XLTMP
1236 3  ERREXIT IF OPEN ERROR TO :FILERR:
1237 3  CALL READ TO READ EXISTING ABSTRACT
1238 3  ERREXIT IF READ ERROR TO :FILERR:
1239 3  CALL EXEC TO DISPLAY EXISTING ABSTRACT
1240 3  CALL XTCOM TO PROMPT USER TO MODIFY ABSTRACT
1241 3  IF RESPONSE IS CR (NO MODIFICATION) THEN
1242 4  CALL WRITF TO WRITE EXISTING ABSTRACT TO NEW FILE
1243 4  ERREXIT IF WRITF ERROR TO :FILERR:
1244 3  ELSE (NEW ABSTRACT WAS ENTERED)
1245 4  PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1246 3  ENSIF
1247 2  ELSE (ABSTRACT CODE IS AD)
1248 3  CALL XTCOM TO PROMPT USER TO ENTER NEW ABSTRACT
1249 3  PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1250 2  ENDF
1251 2  IF PARAMETERS IS NOT ZERO THEN
1252 3  PERFORM EXT.deep TO GET DEFINITIONS FOR EACH PARAMETER
1253 2  ENDF
1254 2  CALL CLOSE TO CLOSE PROMPT FILE
1255 2  ERREXIT IF CLOSE ERROR TO :FILERR:
1256 2  IF THERE WAS AN OLD FILE THEN
1257 3  CALL CLOSE TO CLOSE OLD FILE >XLMP
1258 3  CALL PURGE TO PURGE OLD FILE >XLMP
1259 2  ENDF
1260 1 EXIT XLPL
1261 2  :FILERR:
1262 2  CALL XPAR TO DISPLAY ERROR CODE
1263 2  CALL CLOSE TO CLOSE NEW FILE
1264 2  CALL CLOSE TO CLOSE OLD FILE >XLMP
1265 2  CALL PURGE TO PURGE OLD FILE >XLMP
1266 2  SET VALFLG =0
1267 2  CALL XERTN TO RETURN TO MAIN ***NO RETURN TO HERE***
1268 1 END XLPL
1270 1 BEGIN FORMAT
1271 2 INITIATE TOKEN POINTER AND TOTAL WORD COUNT
1272 3 DO UNTIL EOS IS DETECTED IN RESPONSE
1273 4 ERREXIT IF RESPONSE IS NOT CHARACTER STRING TO :ERR1:
1274 5 ERREXIT IF RESPONSE IS TOO LONG (>128 WORDS) TO :ERR1:
1275 6 CALL ISMOV TO MOVE RESPONSE TO BUFFER
1276 7 SET CONTROL CHARACTERS IN BUFFER
1277 8 INCREMENT TOTAL WORD COUNT BY THIS RESPONSE
1278 9 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
1279 0 INCREMENT TOKEN POINTER TO NEXT CHARACTER STRING
1280 1 ENDDO
1281 2 SET REMAINDER OF BUFFER TO NULL
1282 3 CALL WRITF TO WRITE NEW RESPONSE TO PROMPT FILE
1283 4 ERREXIT IF WRITF ERROR TO :FILERR:
1284 5 END FORMAT
1285 1 :ERR1:
1286 2 CALL XRMNG TO DISPLAY INVALID RESPONSE
1287 3 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
1289 1 BEGIN EXTPRN
1290 2 DO FOR EACH ENTRY IN CODES ARRAY
1291 3 BEGIN CASE (:NOCHNG:, :MOD:, :DEL:, :ADD:), CODES+1
1292 4 1 :NOCHNG: (SPICS HAD NO CHANGES)
1293 4 2 CALL READ TO READ EXISTING DEFINITION FROM XLTHP
1294 4 3 ERREXIT IF READF ERROR TO :FILER:
1295 4 4 CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
1296 4 5 ERREXIT IF WRITF ERROR TO :FILER:
1297 4 6 :MOD: (SPICS WERE MODIFIED)
1298 4 7 CALL READ TO READ EXISTING DEFINITION FROM XLTHP
1299 4 8 ERREXIT IF READF ERROR TO :FILER:
1300 4 9 CALL EXEC TO DISPLAY EXISTING DEFINITION
1301 4 10 IF RESPONSE IS CR (NO RESPONSE) THEN
1302 4 11 CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
1303 4 12 ERREXIT IF WRITF ERROR TO :FILER:
1304 4 13 ELSE (NEW DEFINITION WAS ENTERED)
1305 4 14 PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
1306 4 15 ENDIF
1307 4 16 :DEL: (SPICS WERE DELETED)
1308 4 17 CALL POSNT TO POSITION XLTHP OVER THIS ENTRY
1309 4 18 ERREXIT IF POSNT ERROR TO :FILER:
1310 4 19 :ADD: (SPICS WERE ADDED)
1311 4 20 CALL XT:OM TO PROMPT USER TO ENTER NEW DEFINITION
1312 4 21 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1313 4 22 END CASE
1314 3 DO UNTIL RESPONSE IS CR (EXIT)
1315 4 1 CALL XT:OM TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
1316 4 2 EXIT EXTPRN IF RESPONSE IS CR
1317 4 3 SEARCH SHORT PROMPTS FOR RESPONSE
1318 4 4 ERREXIT IF NOT FOUND TO :ERR:
1319 4 5 CALL READY TO READ EXISTING DEFINITION FROM NEW FILE
1320 4 6 ERREXIT IF READF ERROR TO :FILER:
1321 4 7 CALL EXEC TO DISPLAY EXISTING DEFINITION
1322 4 8 CALL XT:OM TO PROMPT USER TO MODIFY DEFINITION
1323 4 9 IF RESPONSE IS POSITIVE
1324 4 10 CALL POSNT TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
1325 4 11 ERREXIT IF POSNT ERROR TO :FILER:
1326 4 12 PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
1327 4 13 ENDIF
1328 4 14 ENDB
1329 3 ENDB
1330 2 ENDB
1331 1 END EXTPRN
C**************

FORTRAN CALLING SEQUENCE:

CALL XLPMO

C**************

XLPMO MODIFIES THE EXISTING PDB LOG FILE

C**************

INPUTS FROM COMMON:

XE(1) LU, XE(3) QUAL, XE(8) SECU, XE(142) ICA, XB(99) IERR, XB(100) INCB,

C**************

INTERNAL VARIABLES IN COMMON:

XE(52) UNO - NUMBER OF USER ID A-Z (1-26)

XE(53) RECN - RECORD # WHERE UNO'S PDB LIST IS (2-14)

XE(52) PHAM - FILE NAME OF USER'S PDB (FULL 6 CHAR) XE(55) DRCN - NUMBER OF FILES THAT MUST BE DELETED

XE(244) OUTBUF-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

C**************

FILE Routines USED:

CLOSE, EXEC, KEVT, PURGE, RREAD, WRITF

C**************

DFS Routines Used:

XCPB, XCPY, XPD, XRP, XRSF, XTBL

C**************

COMMON USED:

C**************

XEDOJOEUV (XE(1), LU )

+XE(3), QUAL ,

+XE(8), SECU ,

+XE(142), ICA ,

+XE(50), UNO ,

+XE(55), PHAM ,

+XB(56), RTN ,

+XB(99), IERR ,

+XB(244), OUTBUF ,

+XB(304), INCB
1387 1 BEGIN XLPMO
1388 2 DO UNTIL USER RESPONSES EXIT (2)
1389 3 CALL Xtcom 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 Xtcom 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 WRIT 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 Xtcom 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 DECREASE 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 WRIT TO REWRITE UPDATED LOG RECORD
1418 32 ERREXIT IF WRITE ERROR TO :FILERR:
1419 33 DO FOR #PURGED FILES
1420 34 CALL XKNR TO CREATE FILE NAME
1421 35 CALL PURGE TO PURGE FILE
1422 36 ENDDO
1423 37 ENDFD
1424 38 ENDIF
1425 39 ENDIF
1426 40 ENDDO
1427 41 CALL CLOSE TO CLOSE MDB/PDB LOG FILE
1428 42 EXIT XLPMO
1429 43 :ERR1:
1430 44 CALL XHMSG TO DISPLAY ERROR IN RESPONSE
1431 45 RETURN TO REISSUE LAST PROMPT
1432 46 :FILERR:
1433 47 CALL XHMSG TO DISPLAY FILE ERROR
1434 48 CALL CLOSE TO CLOSE MDB/PDB FILE
1435 49 END XLPMO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:
CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

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

ROUTES USED:
KCVT, XRCPY, XMMOV, XRMSG, XRSET, XTCOM

XE COMMON USED:
EQUIVALENCE (XE(8), RETC), (XE(9), RETN),。
  + (XE(10), CLASS), (XE(11), TYPE )。
  + (XE(145), COMBUF), (XE(37), ARGNO )。
  + (XE(96), NOPARM), (XE(108), PARM )
1 BEGIN XLSPS
2  CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII

:PRMP1:
2  SET RETURN TO 1
2  CALL XCOM TO PROMPT FOR PROMPT,CLASS,TYPE AND I/O FLAGS
2  ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
2  INITIALIZE THIS SHORT PROMPT AND ENTRY
2  ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
2  ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
2  CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
2  ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
2  CALL XRSET TO SET CLASS IN ENTRY
2  ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
2  ERREXIT IF CLASS IS ORDER AND TYPE IS SYMBOLIC STRING TO :PRMERR:
2  CALL XRSET TO SET TYPE IN ENTRY
2  ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
2  ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
2  CALL XRSET TO SET I/O FLAGS IN ENTRY
2  IF CLASS IS DATA ELEMENT THEN
2  IF TYPE IS NOT SYMBOLIC STRING THEN
2  SET RETURN TO 2

:PRMPT2:
4  CALL XCOM TO PROMPT FOR I AND J DIMENSIONS
4  ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
4  ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
4  I: J DIMENSION IS NOT ENTERED THEN
5  SET J DIMENSION TO 1
4  ELSE
5  ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
5  IF J DIMENSION < 2 THEN
6  SET J DIMENSION TO 1
5  ELSE
6  STORE I DIMENSION IN ENTRY
5  ENDIF
5  ENDIF
5  SET SIZE =I DIMENSION * J DIMENSION * TYPE LENGTH
5  ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
5  ELSE
5  SET RETURN TO 3

:PRMPT3:
4  CALL XCOM TO PROMPT FOR MAXIMUM SIZE
4  ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
4  ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
4  SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
3  ENDIF
2  ENDIF
1  EXIT XLSPS

:PRMERR:  CALL XRMSG TO DISPLAY ERROR MESSAGE
2  GO TO (:PRMP1:, PRMPT2:, PRMPT3:, RETURN
1  END XLSPS
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
**REQUEST & NN IS LU**

61 6 * SET UP FOR FDS TERMINATION

62 6 * EXIT TO :XMEND

63 5 ENDF

64 4 ENDF

65 3 ENDF

66 2 ENDDO

67 2 :XMEND

68 2 IF DUMP OPTION THEN

69 3 CALL XUDMP TO DUMP OUR PARTITION

70 2 ENDF

71 2 RELEASE DWA TRACK SPACE

72 2 FREE EXEC'S & PROCESSOR'S CLASS NUMBERS

73 2 CALL RNRQ SET LOCK ON STATUS TABLE

74 2 UNLOCK THE MANAGER FROM THE PARTITION

75 2 CALL SLIBR DISABLE

76 2 CLEAR EXEC'S ID SEGMENT

77 2 CALCULATE EQT ADDRESS FOR ATTENTION ID

78 2 CLEAR THE STATUS TABLE ENTRY

79 2 CALL SLIBX ENABLE

80 2 CALL RNRQ CLEAR THE LOCK ON THE STATUS TABLE

81 2 ISSUE MESSAGE "***XMDZ SIGN OFF FOR LU "NN"!"

82 2 CALL SLIBX GET PRIVLEDGED AND DISABLED

83 2 CALL SLIST TO MAKE US DORMANT

84 2 CLEAR OUR OWN ID

85 2 RESET EQT TO REMOVE ATTENTION ID

86 1 EXIT $XEQ TO THE RTE DISPATCHER

87 1 END  XMGR

88 1 XMGR
CALLING PROCEDURE

JSB XMXGT

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

SEQUENCE TABLE EXECUTION FROM &SEQTB

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

INPUT

XMAWM, XMPRM, XVSTA

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

OUTPUT

XMCTP, XMPRM, XVSTA

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

ROUTINES USED

CMUNP, EXEC, XMAGM, XMKL, XMPAW (XMSCH),

XMSGT, XMTF4, SLIB3, SLIB4

************
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>BEGIN XNXT</td>
</tr>
<tr>
<td>112</td>
<td>CALL XNXT TO FIND BSETB TOC ENTRY</td>
</tr>
<tr>
<td>113</td>
<td>FIND ADDRESS IF BSETB</td>
</tr>
<tr>
<td>114</td>
<td>CALL XNXT TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT</td>
</tr>
<tr>
<td>115</td>
<td>CALL XNXT TO CONVERT STARTING SEQUENCE NUMBER INTO CURRENT DISPLACEMENT</td>
</tr>
<tr>
<td>116</td>
<td>COMPUTE CURRENT ENTRY ADDRESS</td>
</tr>
<tr>
<td>117</td>
<td>DO UNTIL THE LAST TABLE ENTRY IS EXECUTED OR</td>
</tr>
<tr>
<td>118</td>
<td>UNTIL THE TERMINATION ENTRY IS EXECUTED</td>
</tr>
<tr>
<td>119</td>
<td>IF PROCESSOR REQUIRES AN INTERFACE TABLE (WORD 3 BIT 8 IS SET) THEN</td>
</tr>
<tr>
<td>120</td>
<td>EXIT TO :ERROR: IF INTERFACE TABLE NOT SPECIFIED (WORD 4 = 0) (PAM5 = 1)</td>
</tr>
<tr>
<td>121</td>
<td>CALL XNXT TO SEARCH AWA FOR INTERFACE TABLE (CHAIN 4)</td>
</tr>
<tr>
<td>122</td>
<td>EXIT TO :ERROR: IF TABLE NOT FOUND (PAM5 = 2)</td>
</tr>
<tr>
<td>123</td>
<td>IF TABLE NOT IN AWA, THEN</td>
</tr>
<tr>
<td>124</td>
<td>CALL INDR TO RETRIEVE FROM AWA</td>
</tr>
<tr>
<td>125</td>
<td>EXIT TO :ERROR: IF SPACE IS NOT FOUND</td>
</tr>
<tr>
<td>126</td>
<td>ENDF</td>
</tr>
<tr>
<td>127</td>
<td>EXIT TO :ERROR: IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PAM5 = 3)</td>
</tr>
<tr>
<td>128</td>
<td>EXIT TO :ERROR: IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PAM5 = 4)</td>
</tr>
<tr>
<td>129</td>
<td>EXIT TO :ERROR: IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PAM5 = 5)</td>
</tr>
<tr>
<td>130</td>
<td>ENDF</td>
</tr>
<tr>
<td>131</td>
<td>SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED</td>
</tr>
<tr>
<td>132</td>
<td>ERREXIT TO :ERROR: IF NOT FOUND</td>
</tr>
<tr>
<td>133</td>
<td>CALL SLIBR TO BE PRIVILEGED</td>
</tr>
<tr>
<td>134</td>
<td>SET CURRENT TASK TO THIS PROCESSOR</td>
</tr>
<tr>
<td>135</td>
<td>CALL SLIBX TO BE UN-PRIVILEGED</td>
</tr>
<tr>
<td>136</td>
<td>IF PROCESSOR REQUIRES AN INTERFACE TABLE THEN</td>
</tr>
<tr>
<td>137</td>
<td>CALCULATE LENGTH OF INTERFACE TABLE HEADER AND SPECS</td>
</tr>
<tr>
<td>138</td>
<td>CALL EXEC TO CLASS I/O WRITE HEADER AND SPECS</td>
</tr>
<tr>
<td>139</td>
<td>ENDF</td>
</tr>
<tr>
<td>140</td>
<td>CALL EXEC TO SCHEDULE PROCESSOR WITH WAIT</td>
</tr>
<tr>
<td>141</td>
<td>CALL XASCH TO RETRIEVE PARAMETERS FROM PROCESSOR</td>
</tr>
<tr>
<td>142</td>
<td>CALL XNXT TO FIND BSETB TOC ENTRY</td>
</tr>
<tr>
<td>143</td>
<td>FIND ADDRESS OF BSETB</td>
</tr>
<tr>
<td>144</td>
<td>COMPUTE CURRENT ENTRY ADDRESS</td>
</tr>
<tr>
<td>145</td>
<td>DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PAM1 = 1)</td>
</tr>
<tr>
<td>146</td>
<td>CALL XHANG TO HONOR AWA REQUEST</td>
</tr>
<tr>
<td>147</td>
<td>ENDDO</td>
</tr>
<tr>
<td>148</td>
<td>CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR</td>
</tr>
<tr>
<td>149</td>
<td>ERREXIT TO :ERROR: IF PROCESSOR REQUESTED TERMINATION (PAM1 = 8)</td>
</tr>
<tr>
<td>150</td>
<td>ERREXIT TO :ERROR: IF PROCESSOR ABENDED (PAM1 = 32768)</td>
</tr>
<tr>
<td>151</td>
<td>IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PAM1 = 3) THEN</td>
</tr>
<tr>
<td>152</td>
<td>ERREXIT TO :ERROR: IF RESET NUMBER IS ZERO</td>
</tr>
<tr>
<td>153</td>
<td>CALL XMST TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY - DISPLACEMENT</td>
</tr>
<tr>
<td>154</td>
<td>ERREXIT TO :ERROR: IF RESET SEQUENCE NUMBER IS NOT FOUR</td>
</tr>
<tr>
<td>155</td>
<td>IF TERMINAL ENTRY WAS JUST EXECUTED THEN</td>
</tr>
<tr>
<td>156</td>
<td>SET UP PARAMS TO SHOW RESET SEQUENCE NUMBER</td>
</tr>
<tr>
<td>157</td>
<td>PERFORM :END: = <strong>NO RETURN EXPECTED</strong></td>
</tr>
<tr>
<td>158</td>
<td>ENDF</td>
</tr>
<tr>
<td>159</td>
<td>ELSE</td>
</tr>
<tr>
<td>160</td>
<td>IF REQUEST IS NOT NORMAL COMPLETION (PAM1 = 0) THEN</td>
</tr>
<tr>
<td>161</td>
<td>DISPLAY ERROR MESSAGE - INVALID REQUEST</td>
</tr>
<tr>
<td>162</td>
<td>ERREXIT TO :ERROR: IF CURRENT AT IS DORMANT</td>
</tr>
<tr>
<td>163</td>
<td>CALL XMXTL TO SET PARAMETERS TO ABEND ASSOCIATED TASK</td>
</tr>
<tr>
<td>164</td>
<td>CALL XMPAR TO RESCHEDULE PROCESSOR</td>
</tr>
<tr>
<td>165</td>
<td>ERREXIT TO :ERROR: TO TERMINATE SEQUENCE</td>
</tr>
<tr>
<td>166</td>
<td>ENDF</td>
</tr>
<tr>
<td>167</td>
<td>ENDF</td>
</tr>
<tr>
<td>168</td>
<td>INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY</td>
</tr>
<tr>
<td>169</td>
<td>ENDDO</td>
</tr>
</tbody>
</table>
171 2 SET PARM1 = 0 (NORMAL COMPLETION)
172 2 PERFORM :END: - NO RETURN EXPECTED
173 2 :ERROR1: SET PARM1 = 1
174 2 :ERROR2: SET PARM1 = 8 AND PARM5 TO APPROPRIATE REASON CODE
175 2 :END:
176 2 CALL SLIBR TO BECOME PRIVLEDGED
177 2 SET CURRENT TASK IN MPK, STATUS TABLE, AND ANA TO EXEC
178 2 CALL SLIBX TO BECOME UN-PRIVLEDGED
179 2 CALL IMPAN TO POST EXEC AND WAIT FOR NEXT REQUEST
180 1 END XMXQT
CALLING PROCEDURE

182 1 *00 jsb xafr
183 1 *00 def **3
184 1 *00 def adrr
185 1 *00 def size
186 1 *00
187 1 *00
188 1 *00
189 1 ********
190 1 *01 place a fe on the fe chains and merge with any adjacent fes
191 1 *01
192 1 *00
193 3 ********
194 1 *02 input
195 1 *02 addr - address of area being freed
196 1 *02 size - size of area being freed. if less than 3 the following
197 1 *02 words(s) will also be freed such that the minimum fe size
198 1 *02
199 1 *02
200 1 *02 external symbols from xmafa
201 1 *02 xmbcp, xmfcp, xmfnc, xmfre
202 1 *02
203 1 ********
204 1 *03 output (external symbols from xmafa)
205 1 *03 xmbcp, xmfcp, xmfnc, xmfre
206 1 *03
207 1 ********
208 1 *05 notes
209 1 *05 uses .ent
210 1 *05
211 1 ********
CALLING PROCEDURE

JBB XMAGT
DEF #3
DEF OPTM
DEF SIZE

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

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

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

INPUT

QPTM - INDICATOR OF WHICH FREE CHAIN TO SEARCH
D = FORWARD POINTER CHAIN (FOR TDC SPACE)
B = 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 XMAYA
XMFCP, XMFCF, XMFCF

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

OUTPUT

A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
INDICATING NONE AVAILABLE
EXTERNAL SYMBOLS FROM XMAYA
XMFCP, XMFCF, XMFCF
XMFCF, XMFCF

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

NOTES

IF SOME EXACTLY 'SIZE' WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
ERROR RETURN (A-REG = -32768) IS TAKEN
USES .EMIT
1 BEGIN XMACT
2 IF TOTAL FREE SPACE < MAX(SIZE,3) THEN
298 CALL XMPK1 PURGE DNA ELEMENTS FROM AMA
299 SET PHASE1 FLAG
300 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
301 RETURN VALUE= -32768
302 EXIT TO :XMAGR
303 ENDIF
304 ENDIF
305 ENDIF XMPERC(OPTY,SIZE)
306 IF RETURN CODE IS NOT FOUND, THEN
307 CALL XMPK
308 IF RETURN CODE IS NOT FOUND, THEN
309 RETURN VALUE= -32768
310 EXIT TO :XMAGR
311 ENDIF
312 RETURN VALUE= AREA
313 :XMAGR
314 ENDIF
315 CLEAR PHASE1 FLAG
316 1 END XMACT
BEGIN XMSRC
SET INDEX TO APPROPRIATE CHAIN HEAD, E.G., FCHAIN(OPTH)
START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
WHILE TOC SPACE FENCE HAS NOT BEEN Crossed
EXIT IF FS SIZE = MAX("SIZE", 3)
DECREMENT TOTAL FREE SPACE BY MAX("SIZE", 3)
BEGIN FE
IF ALLOCATING FROM HEAD OF SPACE (OPTH = 0)
THEN
CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
RECHAIN NEW FE
RETURN ADDRESS OF AREA
ELSE
CHANGE SIZE FIELD TO FS SIZE = MAX("SIZE", 3)
ENDIF
OR ELSE
INDEX TO NEXT FE
END LOOP
SET RETURN CODE TO 0 (NOT FOUND)
END SEARCH
IF TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
INCREMENT TOC SPACE FENCE by "SIZE"
ENDIF
END XMSRC
CALLING PROCEDURE

JSB XMANG
DEF CLSIO

**********

*01 PROVIDE AWA MANAGEMENT BASED ON REQUEST LIST (SEE FDS SDD TABLE 6.2 - III)

*01 REQUEST LIST (SEE TABLE 6.2 - III)

*02 CLSIO - CLASS I/O NUMBER CONTAINING REQUEST LIST

*02 REQUEST LIST FIELD 8

** NOTES

*05 USES EXEC,XMAFR,XMACT,XMAPK,XMINF

*05 ZMOIN,ZMODL,ZMODA,ZMODT,ZMDT

*05 NEITHER EXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS

*05 FOR DWA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DWA

*05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO

*05 THE DWA AND COPY THEM BACK TO THE AWA AS NECESSARY.

**********
1 BEGIN XNAG
2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER
3 CLEAR RETURN PARM1
4 CLEAR RETURN PARM2
5 INITIALIZE TO FIRST REQUEST CODE
6 DO UNTIL END OF LIST (D), 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 XNAPK TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
11 ENDF
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 RETURN PARM1
16 ELSE
17 IF REQUESTS TO CLEAR (CODE 17)
18 THEN
19 GET A(KDWA) FROM XDWA
20 SAVE HEADER AND DIRECTORY-SIZE
21 CLEAR KDWA THRU KMDO
22 BUILD AN FE AT KDWA FOR AWA SIZE
23 CALL XMAGT TO ALLOCATE A TOC ENTRY FOR KDWA
24 CALL XMAGT TO ALLOCATE SPACE FOR KMDO
25 SET LOCATION, SIZE, & KEY IN THE TOC
26 SET DIRECTORY ADDRESS AT KDWA
27 CLEAR THE DIRECTORY
28 SET LOCATION, & NUMBER OF TRACKS IN THE DIRECTORY
29 ELSE
30 CALL XMAGT TO SEARCH TOC FOR INDICATED ENTRY
31 CASE (:VERIFY; :ERALO; :ERALO; :RENAMe; :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 ENDF
38 :ERALO:
39 IF ENTRY ALREADY EXISTS
40 THEN
41 IF ALLOCATE REQUEST (33)
42 THEN
43 SET RETURN PARM1 AND PARM2 (3 & INDEX)
44 ELSE
45 IF TYPE, SIZE AND 1-DIM FIELDS DO NOT MATCH
46 THEN
47 SET RETURN PARM1 AND PARM2 (4 & INDEX)
48 ENDF
49 ELSE
50 ENDIF
51 CALL XMAGT TO ALLOCATE TOC SPACE
52 IF CLASS EQ 3 OR 8, THEN
53 CALL XMAGT TO ALLOCATE DATA SPACE ADDRESS TO ZERO
54 ELSE
55 CALL XMAGT TO ALLOCATE DATA SPACE
56 ENDIF
57 END
58 END
437 8 IF SPACE NOT AVAILABLE
438 8 THEN
439 9 SET RETURN PARM1 AND PARM2 (1 & INDEX)
440 9 ELSE
441 9 IF DATA ELEMENT (CLASS 2)
442 10 THEN
443 10 IF CHARACTER STRING (TYPE 4 - 8)
444 10 THEN
445 10 INITIALIZE AREA TO BLANKS
446 10 ELSE
447 10 INITIALIZE AREA TO ZEROS
448 10 ENDIF
449 9 CHAIN IN NEW TOC ENTRY
450 9 ENDIF
451 8 IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE
452 8 THEN
453 8 CALL XMDAL DWA ALLOCATION
454 8 ENDIF
455 7 ENDIF
456 6 457 6 :RENAME:
458 6 IF ENTRY NOT FOUND
459 6 THEN
460 7 SET RETURN PARM1 AND PARM2 (2 & INDEX)
461 7 ELSE
462 7 CALL XMTFN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION
463 7 IF ENTRY FOUND
464 7 THEN
465 8 SET RETURN PARM1 AND PARM2 (3 & INDEX)
466 8 ELSE
467 8 CALL XMTAFT TO ALLOCATE NEW TOC ENTRY
468 8 IF SPACE NOT AVAILABLE
469 8 THEN
470 9 SET RETURN PARM1 AND PARM2 (1 & INDEX)
471 9 ELSE
472 9 COPY OLD ENTRY ATTRIBUTES INTO NEW ENTRY AND CHAIN IN TO TOC
473 9 DECHAIN OLD ENTRY
474 9 CALL XMAFR TO RETURN OLD ENTRY TOC SPACE TO FE POOL
475 8 ENDIF
476 7 ENDIF
477 6 ENDIF
478 6 :DELIVER:
479 6 IF ENTRY FOUND
480 6 THEN
481 7 GENERATE KEY 1 LESS THAN FOUND KEY
482 7 CALL XMTFN FOR GENERATED KEY
483 7 DECHAIN TOC ENTRY
484 7 CALL XMAFR TO RETURN TOC ENTRY SPACE TO FE POOL
485 7 IF CLASS EQ 4 OR 6, THEN
486 7 CALL XMDAA DWA DEALLOCATION
487 7 ENDIF
488 7 ELSE
489 6 IF DELETE REQUEST (5)
490 7 THEN
491 8 SET RETURN PARM1 AND PARM2 (2 & INDEX)
492 8 ENDIF
493 7
ENDIF

495 6  
STORE:
496 6  
IF ENTRY NOT FOUND
497 6  THEN
498 7  SET RETURN PARM1 AND PARM2 (2 & INDEX)
499 6  ELSE
500 7  IF (TOC TYPE > 0 AND INCONSISTENT WITH REQUEST TYPE) OR
501 8  DISPLACEMENT OR SPECIFIED SIZE < 0, OR
502 8  DISPLACEMENT + REQUESTED SIZE > ALLOCATED SIZE
503 7  THEN
504 8  SET RETURN PARM1 AND PARM2 (4 & INDEX)
505 7  ELSE
506 8  GET DATA FROM INDICATED CLASS I/O; STORE INTO AWA
507 8  FREE CLASS NUMBER
508 8  IF CLASS EQ 4 OR 6, THEN
509 9  CALL XMST DMA STORE DATA
510 8  ENDF
511 7  ENDF
512 6  ENDF

513 6  
RETRIEVE:
514 6  IF ENTRY NOT FOUND
515 6  THEN
516 7  SET RETURN PARM1 AND PARM2 (2 & INDEX)
517 6  ELSE
518 7  IF VALUES REQUESTED (8)
519 7  THEN
520 8  IF (TOC TYPE .NE. 0 AND .NE. REQUESTED TYPE), OR
521 9  DISPLACEMENT OR SPECIFIED SIZE < 0, OR
522 9  DISPLACEMENT + SPECIFIED SIZE > ALLOCATED SIZE
523 8  THEN
524 9  SET RETURN PARM1 AND PARM2 (4 & INDEX)
525 8  ELSE
526 9  IF REQUESTED SIZE = ZERO
527 9  THEN
528 10  CALCULATE AMOUNT OF DATA TO RETRIEVE AS ACTUAL SIZE MINUS DISPLACEMENT
529 10  STORE COMPUTED SIZE IN REQUEST WORD SIX
530 9  ENDF
531 9  IF CLASS EQ 4 OR 6 AND TOC ADDRESS EQ 0, THEN
532 10  THE ELEMENT EXISTS ONLY ON THE DMA
533 10  CALL XMRT MOVE INTO AWA
534 10  IF NO SPACE THEN
535 11  SET RETURN PARM1 AND PARM2 TO (1, INDEX)
536 10  EXIT TO XMREX
537 10  ENDF
538 9  ENDF
539 9  WRITE VALUES TO CLASS I/O
540 9  STORE TYPE IN LOW BYTE OF REQUEST WORD 1
541 9  STORE CLASS NUMBER IN REQUEST WORD 2:CHT
542 9  SET REWRITE FLAG
543 8  ENDF
544 7  ELSE
545 8  WRITE TOC ENTRY TO CLASS I/O
546 8  STORE CLASS NUMBER IN REQUEST WORD 8:CHT
547 8  SET REWRITE FLAG
548 8  :XMREX
549 7  ENDF
649 1 BEGIN XMPAY
650 2 *00  ENTRY: JSB XMPAY OR CALL XMPAY
651 2 *00  DEF RETURN ADDRESS
652 2 *02  INPUTS: XMPAY HAS REPLY
653 2 *02  OUTPUTS: XMPAY HAS REQUEST,
654 2 *03  AND XUSTB IS UPDATED.
655 2 *03  DO UNTIL GOOD PARM RECEIVED OR TOP AT TERMINATES
656 3 1 IF ABORT CURRENT FLAG ON (-STBLU) THEN
657 4 1 SET ABORT FLAG IN XMPAY
658 4 1 TURN OFF ABORT CURRENT FLAG
659 3 1 ENDIF
660 3 1 GET CURRENT TOP ASSOCIATED TASK(AT)
661 3 1 SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
662 3 1 JSB XVPAY
663 3 1 DEF *#3 RETURN
664 3 1 DEF O  MANAGER CALL
665 3 1 DEF XMPAY PARM FIELD
666 3 1 THIS IS AN IMPLIED WAIT
667 3 1 XMSCN GET CURRENT XUSTB ENTRY (XUSTA)
668 3 1 IF THERE HAS BEEN A CALL TO PRMP (P1 FIELD IS NOT TOP AT) OR
669 4 1 ID SEGMENT IS DORMANT OR
670 4 1 ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
671 4 1 IF PARM IS NOT 0,3,8,9, OR -32768 THEN
672 5 1 SET PARM TO -32768
673 5 1 PRINT ERROR "INVALID REQUEST"
674 4 1 ELSE
675 5 1 SET PARM FIELD FROM MANAGER'S ID SEGMENT
676 4 1 ENDIF
677 3 1 ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PARM)
678 4 1 IF PARM NOT 1 OR 2 THEN
679 5 1 CALL XMKIL TO ABORT TOP AT
680 5 1 PRINT ERROR "INVALID REQUEST"
681 4 1 ELSE
682 5 1 SET PARM FROM CURRENT ID SEGMENT
683 4 1 ENDIF
684 3 1 ENDIF
685 2 1 ENDDO
686 1 1 END XMPAY
CALLING PROCEDURE

JSB XMDIN

FUNCTION

INITIALIZE TOC

& DWA FUNCTIONS

OUTPUT

BREG D=COMPLETE

MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

XMTFN, XMAGT

BEGIN XMDIN

GET NUMBER OF DWA TRACKS FROM PS

CALCULATE SIZE OF DWA(3+6H) N IS # OF TRACKS

CALL XMTFN (EDWA)

CALL XMAGT (O,6) TOC ENTRY FOR EDWA

CALL XMAGT (1,SIZE) DATA AREA FOR EDWA

INITIALIZE & CHAIN EDWA TOC ENTRY

SET # OF TRACKS FOR EDWA

CLEAR EDWA

CALL EXEC (DISC TRACK ALLOCATION)

IF DISC ADDRESS .EQ. -1 TRACKS NOT AVAILABLE

THEN

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

ERROR RETURN

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DWA MANAGEMENT

ENDIF

END XMDIN
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
802 9 BEGIN XNOST
803 2 * DMA STORE DATA
804 2 *00 CALLING PROCEDURE
805 2 *00 JSB XNOST
806 2 *00
807 2 *01 FUNCTION
808 2 *01 UPDATE DMA ELEMENT WHICH CORRESPONDS
809 2 *01 TO THE AMA ELEMENT
810 2 *01
811 2 *02 INPUTS
812 2 *02 TRC ENTRY ADDRESS OF AMA ELEMENT
813 2 *02
814 2 *02
815 2 *02 OUTPUTS
816 2 *03 UPDATE DMA ELEMENT ON DISC TRACKS
817 2 *03
818 2 *05 NOTES
819 2 *05 USES WRITE
820 2 *05
821 2 IF A(EDUA).NE.0. THEN
822 3 SET DISC ADDRESS FOR THE WRITE
823 3 SET DATA ADDRESS FOR THE WRITE
824 3 WRITE DATA
825 2 ENDF
826 1 END XNOST
829 1 BEGIN XNOCT
830 2 0 DNA RETRIEVE
831 3 CALLING PROCEDURE
832 4 JS0 XNOCT
833 5 FUNCTION
834 6 RETRIEVE DNA DATA INTO AWA
835 7 IMPTS
836 8 A(TOC ENTRY) IN VREG
837 9 IMPUTS
838 10 OUTPUTS
839 11 ADDRESS OF DATA IN THE TOC
840 12 BREG=0, RETRIEVE SUCCESSFUL
841 13 MINUS, ERROR NO AWA DATA
842 14 NOTES
843 15 USES XNAGT, XNOMA, EXEC(READ)
844 16 IF NO DNA DIRECTORY, THEN
845 17 SET ERROR CODE -5
846 18 ELSE
847 19 CALL XNAGT, GET DATA SPACED
848 20 IF NO SPACE, THEN
849 21 SET ERROR CODE -1
850 22 ELSE
851 23 SET DATA ADDRESS IN TOC
852 24 GET DISC ADDRESS
853 25 READ DATA INTO AWA
854 26 SET RETURN CODE TO 0
855 27 ENDIF
856 28 ENDIF
857 29 END XNOCT
860 31 END XNOCT
862 1 BEGIN XMBST
863 2 *00 CALLING PROCEDURE
864 2 *00 JSB XMBST
865 2 *00 XMBST
866 2 *01 FUNCTION
867 2 *01 SET ALLOCATION & DEALLOCATION IN DWA DIRECTORY
868 2 *01 XMBST
869 2 *02 INPUTS
870 2 *02 XMBST
871 2 *03 OUTPUTS
872 2 *03 XMBST
873 2 *03 UPDATES DWA DIRECTORY
874 2 *05 XMBST
875 2 GET TRACK-ADDRESS FROM TOC
876 2 SUBTRACT START OF TRACKS FROM DWA DIRECTORY FOR RELATIVE TRACK
877 2 DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)
878 2 QUOTIENT IS NUMBER OF RELATIVE WORDS
879 2 REMAINDER IS BIT-POSITION(BP)
880 2 WORD ADDRESS=DWA ADDRESS*RELATIVE TRACK+RELATIVE WORDS
881 2 SUBTRACT BIT-POSITION FROM 16 GIVING NBFW(NUMBER BITS IN FIRST WORD)
882 2 GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)
883 2 BITS-NBFW=REMAINING BITS(RB)
884 2 IF RB .JE. 0, THEN
885 2 SET NUMBER OF WORDS(NW) TO ZERO
886 2 SET NUMBER OF BITS LAST WORD(NBLW) TO ZERO
887 2 SET NBFW TO BITS
888 2 ELSE
889 2 DIVIDE RB BY 16
890 2 SET NW TO QUOTIENT
891 2 SET NBLW TO REMAINDER
892 2 ENDF
893 2 L-LOAD DATA POINTED TO BY WORD ADDRESS
894 2 ROTATE LEFT (BP+NBFW-1 BITS) SAVE BP BITS & POSITION NBFW BITS
895 2 EXCLUSIVE OR SIGN BIT; (ON TO OFF; OFF TO ON)
896 2 IF NBFW .GT. 1, THEN
897 2 SHIFT RIGHT (ARITHMETIC) BY NBFW-1 PROPAGATE BITS
898 2 ENDF
899 2 IF DATA IS NEGATIVE, THEN
900 2 SET FILL WORD TO -1
901 2 ELSE
902 2 SET FILL WORD TO 0
903 2 ENDF
904 2 ROTATE RIGHT BY BP RESET SAVED BITS
905 2 STORE WHERE WORD-ADDRESS POINTS
906 2 DO WHILE NW .GT. 0
907 2 WORD-ADDRESS=WORD-ADDRESS+8
908 2 SET FILL WORD INTO WHERE WORD-ADDRESS POINTS
909 2 ENDDO
910 2 IF NBLW .GE. 0, THEN
911 2 WORD-ADDRESS=WORD-ADDRESS+8
912 2 L-LOAD DATA POINTED TO BY WORD ADDRESS
913 2 IF NBLW .GT. 1, THEN
914 2 ROTATE LEFT NBLW-1 BITS
915 2 ENDF
916 2 EXCLUSIVE OR SIGN BIT
917 2 IF NBLW .GT. 1, THEN
918 2 SHIFT RIGHT ARITHMETIC BY NBLW-1
919 2 ENDF
920 2 STORE DATA WHERE WORD-ADDRESS POINTS

XMBST
BEGIN XMAPK

*00
CALING PROCEDURE
*00
JSB XMAPK
*00
DEF OPTION
*00
DEF PHASE 1 FLAG
*00

GENERAL COLLAPSE INTERFACE

OUTPUT

*00
RETURNS AREA ADDRESS FOR AREA FOUND
*00
RETURNS -32768 AREA NOT FOUND
*00

NOTES

*05
USES XMPS1,XMPS2,XMPS3,XMSRC

*05

IF PHASE1 FLAG NOT SET, THEN

*00
CALL XMPS1 PURGE DWA ELEMENTS FROM AMA

*00
PERFORM XMSRC(OPTN,SIZE)

ELSE

*00
SET RETURN CODE TO NOT FOUND

*00
ENDIF

*00
IF RETURN CODE IS NOT FOUND, THEN

*00
IF OPTN=1(BACKWARD CHAIN), THEN

*00
CALL XMPS2 PACK TOC CHAIN

*00
CALL XMSRC(OPTN,SIZE)

ENDIF

*00
IF RETURN CODE IS NOT FOUND, THEN

*00
CALL XMPS3 PACK AMA DATA AREAS

*00
PERFORM XMSRC(OPTN,SIZE)

ENDIF

*00
ENDIF

*00
SET RETURN VALUE TO RETURN CODE

END XMAPK
1 BEGIN XMPK2
2  CALLING PROCEDURE
3
4
5 PHASE 2 OF COLLAPSE(COLLAPSE TOC ENTRIES)
6
7 OUTPUT
8 REORDERS TOC ENTRIES
9 UPDATES XMFCN( TOC FENCE ADDRESS)
10 NOTES
11 USES XMFCN,XMFCP,XNTFCN,XNBCP,XMAFR,XMPK2
12
13 INCREMENT XMPP2 UPDATE PHASE 2 COUNT
14 DO WHILE XMFCN .GT. XMFCP AND XMFCP .NE. -32768
15 * ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
16 * WILL A TOC COMPRESS BE DONE.
17 GET FIRST-FREE(FREE) FROM XMFCP
18 GET JLDST FROM THE FREE ELEMENT
19 NWSIZ=OLDSZ-8
20 IF NWSIZ .LT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
21 GET NEXT-FREE FROM FIRST-FREE'S FCP
22 SET INTO XMFCP DELETE FROM THE FORWARD CHAIN
23 IF XMFCP .NE. -32768, THEN IF NOT THE LAST FREE ELEMENT
24 SET NEXT-FREE'S BCP TO -32768
25 ELSE
26 SET XMCP TO -32768 DELETE FROM BACKWARD CHAIN
27 ENDIF
28 ELSE
29 OLDsz=NWSIZ UPDATE LENGTH IN FREE ELEMENT
30 ENDIF
31 MWTOC=(FIRST)+NWSIZ THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
32 OLTOC=XMFCN-8 OLD TOC IS THE ENTRY NBEF THE FENCE
33 TMPKEY=OLTOC'S KEY -1 CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
34 CALL XNTFCN(TMPKEY) FIND PREVIOUS
35 SET PRTOC FROM ARG
36 COPY TOC ENTRY FROM OLTOC TO MWTOC
37 STORE MWTOC ADDRESS INTO PRTOC'S CHAIN
38 CALL XMAFR(OLTOC,TOSIZE) FREE THE OLD TOC ENTRY
39 END
40 1 END XMPK2
1033 1 BEGIN XMHP3
1034 2 *00    CALLING PROCEDURE
1035 2 *00    JSB XMHP3
1036 2 *00    FUNCTION
1037 2 *01    COMPRESS DATA AREA BY MIGRATING DATA AREAS
1038 2 *01    TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
1039 2 *01    OUTPUT
1040 2 *01    UPDATES XMBCP, XMFCP, XMFCN, XMFR, XMHP3
1041 2 *03    NOTES
1042 2 *05    USES XMFR
1043 2 *05    INCREMENT XMHP3 (NUMBER OF PHASE 3's)
1044 2 *05    IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
1045 3      CALL XMHP2, INSURE TOC IS COMPRRESSED
1046 3      DO WHILE XMBCP .GT. XMFCN UNTIL 1 FREE AREA ADJACENT TO THE FENCE
1047 3      IF RCPL OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
1048 4      SET HIGH-WATER TO XMFCN
1049 4      ELSE
1050 5      SET HIGH-WATER TO BCP OF LAST-FREE
1051 5      ENDF
1052 5      ENDIF
1053 5      PERFORM XMHC(LAST-FREE)
1054 5      EXIT IF TOC-ADDRESS .EQ. 0
1055 5      PERFORM XMHC(TOC-ADDRESS)
1056 3      ENDDO
1057 2      ENDF
1058 1     END XMHP3
1064 1 BEGIN XMTSC  TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR O((NONE FOUND)).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XNMAA, THE.
1070 2 * LAST TOC-ENTRY IS AT XNFCN-B.
1071 2 TOC-ENTRY=XNMAA
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 GO UNTIL TOC-ENTRY .GE. XNFCN
1074 3 IF DATA ADDRESS IN TOC-ENTRY Z:
1075 4 ME 0, AND IS
1076 4 LT LOW-WATER, AND IS
1077 4 GT HIGH-WATER, AND IS
1078 4 GT TEST-AD,
1079 3 THEN
1080 4 TEST-AD=DATA ADDRESS
1081 4 TEST-TOC=TOC-ENTRY
1082 3 ENDIF
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 XMANV  AXA MOVE
1089 2 * MOVE THE DATA DEFINED BY THE TOC(WHICH IS
1090 2 * IMMEDIATELY ABOVE THE LAST FREE AREA) INTO
1091 2 * THE BOTTOM OF THE LAST FREE
1092 2 * UPDATE THE LENGTH OF THE RESULTING FREE AREA.
1093 2 * GET DATA ADDRESS FROM THE TOC
1094 2 SAVE THE FIRST THREE WORDS OF THE DATA AREA
1095 2 CALL XOR(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 XMANV
FORTRAN CALLING PROCEDURE

CALL XPATR (LU, INBUF, INTLEN, MRBUFF, IMUN, NAME, TYPE, SIZE, IDIM, DSPT)

*********

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

*********

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL
INBUF - INPUT/OUTPUT BUFFER OF 7*8 PARAMETERS + 1) WORDS,
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTLEN - LENGTH OF INBUF
MRBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS.
IMUN - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE WHOSE
ATTRIBUTES ARE REQUESTED.

*********

OUTPUT

NAME - AN ALPHANUMERIC NAME OF UP TO SIX CHARACTERS WHICH
IDENTIFIES THE DATA ELEMENT OR DDE WHERE THE DATA IS TO
BE OBTAINED/STORED. A ZERO ENTRY INDICATES INPUT DATA
WHICH IS LITERAL DATA STORED WITHIN THE INTERFACE TABLE.
TYPE - DATA TYPE CODE OF THE PARAMETER
SIZE - TOTAL NUMBER OF WORDS OF LITERAL DATA,
TOTAL NUMBER OF WORDS OF REFERENCED INPUT DDE,
TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DDE IF
SUBSCRIPTED OR ZERO IF NOT SUBSCRIPTED,
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DDE OR
ZERO IF OUTPUT DDE.
IDIM - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DDE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIBED OUTPUT.
DSPT - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIBED DATA ELEMENTS ELSE ZERO.
DSPT - FOR AN INPUT DDE THE RTE FILE MANAGER TYPE CODE IS
RETURNED IN DSPT.

*********

EXTERNAL SYMBOLS
(SEE XPEG)

*********

INTERNAL VARIABLES

*********

NOTES

USES .ENTR, XPE13(XPEG), XPINI(XPEG), XPMF(XPEG), XPREQ,
APSO(XPEG), XPRIT, XVSDB

XPATR IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
POINTS XPATR, XPEG AND XRPUT
1 BEGIN XPATR
2 PERFORM XPMI(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
3 EXIT TO :XPE15: (XPGET) IF PARAMETER IS OUT OF RANGE
4 EXTRACT NAME FROM INTERFACE TABLE
5 SET DSCPT TO ZERO
6 IF LITERAL PARAMETER (NAME IS ZERO)
7 THEN
8 ENDIF
9 ELSE
10 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
11 IF SUBSCRIPTED (INTERFACE TABLE DISP OR S FIELDS ARE NON-ZERO)
12 THEN
13 PERFORM XPSB(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
14 STORE TYPE, SIZE, IDIM AND DSCPT
15 ELSE
16 IF DRE
17 THEN
18 PERFORM XPSFM(XPGET) TO QUALIFY FILE NAME
19 ENDIF
20 IF INPUT PARAMETER
21 THEN
22 CALL XPRED TO RETRIEVE TOC ENTRY
23 COPY TYPE, SIZE, IDIM AND DSCPT FROM TOC ENTRY
24 ELSE
25 SET TYPE, SIZE AND IDIM TO ZERO
26 ENDIF
27 ENDIF
28 ENDIF
29 END XPATR
FORTAN CALLING PROCEDURE

CALL XPGET (LU, INTBUF, INTLING, MREDUFF, N, INUMS, IN(1), ..., IN(N))

**********
XPGET ALLOWS PROCESSORS TO OBTAIN DATA FROM THE FDS AWA. IT
VERIFIES THE EXISTENCE OF THE DATA ELEMENTS AND ORDERS
REFERENCED BY THE INTERFACE TABLE AND RETRIEVED VALUES DATA
BY ONE OF THE FOLLOWING WAYS:
1) REQUESTED DATA FROM THE INTERFACE TABLE IS RETURNED OR
2) REQUESTED DATA FROM A DATA ELEMENT REFERENCED BY
THE INTERFACE TABLE IS RETURNED OR
3) THE NAME OF THE ORDER REFERENCED BY THE INTERFACE
TABLE WITH A / SYMBOL AS A PREFIX AND A ONE CHARACTER
USER CODE AS A SUFFIX IS RETURNED SO THE CALLING
PROGRAM MAY RETRIEVE THE DATA.
DATA ELEMENT AND BORE NAMES ARE SEARCHED FOR IN THE AWA TOC.
IF A "?" FIELD, A MESSAGE IS ISSUED AND PROCESSING TERMINATED.

**********
INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL
INTBUF - INPUT/OUTPUT BUFFER OF 7582 CHARACTERS + 1) WORDS.
INTLING - ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE STORING ONLY ON
FIRST USE TO CAUSE INTERFACE INITIALIZATION.

LENGTH OF INTBUF
MREDUFF - MANAGER REQUEST BUFFER (46 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCANNED
AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
N - NUMBER OF PARAMETERS TO BE RETRIEVED. IF N=0, ALL
PARAMETERS ARE REQUESTED AND INUMS IS NOT USED.
INUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
INTERFACE TABLE TO BE RETRIEVED. THE ORDER OF THE
NUMBERS INDICATES THE STORAGE ORDER FOR THE IN(1)

OUTPUT

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

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

AMULT - ARRAY OF 20 BYTES CONTAINING MULTIPLIERS FOR USE BY THE
PROCEDURE SUNCRP IN DETERMINING WORD DISPLACEMENTS FOR
VARIOUS TYPES OF SUBSCRIBED ARRAYS
CLFLX - ADDRESS OF THE CLASS "FIELD" (SECOND WORD) OF THE REQUEST
ARRAY REQUEST
STPT - FLAG INDICATING GET OR PUT PROCESSING. VALUES CORRESPOND
TO BIT SETTINGS FOR INPUT OR OUTPUT FLAGS IN THE FIFTH
WORD OF INTERFACE TABLE ENTRIES

1 - COUNTER OF THE NUMBER OF PASSES THROUGH THE PARAMETER
155 1 *04    PROCESSING LOOP
156 1 *04    NP = VALUE OF -P, WHERE P IS THE NUMBER OF PARAMETERS IN THE
157 1 *04    INTERFACE TABLE
158 1 *04    XNFIL = ADDRESS OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
159 1 *04    ARY = ARRAY RESULT
160 1 *04    NARY = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
161 1 *04    SEQUENCE
162 1 *04    BREST = EIGHT WORD ARRAY USED FOR CONSTRUCTING AMA MANAGEMENT
163 1 *04    REQUESTS FOR XPRE
164 1 *04    USRID = USER FILE IDENTIFIER CHARACTER (=IPARM(3))
165 1 *04    EXTERNAL VARIABLES (SEE XPRE)
166 1 *04    XPCLS
167 1 *04    XPUL
168 1 *04    XPRQD
170 1 *04    NOTES
171 1 *05    USES .ENTR, EXEC, XPRE, XPSIT, XUSTB
173 1 *05    XPGET MUST BE INCLUDED IN PROCESSOR AT FBS BUILD TIME.
174 1 *05    SPRM INT5F MUST BE USED BY XPGET, XPPUT, AND XPATH AND NEQDS
175 1 *05    TO BE INITIALIZED ONLY ONCE BY ANY OF THE TIME ROUTINES.
176 1 *05    XPGET MIGHT REQUEST FOR INPUT DATA FROM THE AMA MAY BE PAID FOR UP TO
180 1 *05    EIGHT AT A TIME, XPRE RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
181 1 *05    REQUESTED IN MULTIPLES OF EIGHT.
FORTRAN CALLING PROCEDURE

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

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

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

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

INPUT - LOGICAL UNIT NUMBER OF USER TERMINAL
INTBUF - INPUT/OUTPUT BUFFER OF (# PARAMETERS + 1) WORDS,
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTLG - LENGTH OF INTBUF
NBOUND - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH
AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
N - NUMBER OF PARAMETERS TO BE STORED. IF N=0, ALL
OUTPUT PARAMETERS ARE ASSUMED TO BE SUPPLIED.
IMUNS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
INTERFACE TABLE REFERENCING DATA ELEMENTS AND ORDER'S
WHERE THE OUTPUT DATA IS TO BE STORED. THE ORDER OF THE
NUMBERS INDICATES THE STORAGE ORDER FROM OUT(1).....OUT(N)
LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE
STORED FROM.
XPGFF - FLAG INDICATING XPGET/XPPUT NORMAL PROCESSING (-1) OR
XPGET/XPPUT SPECIAL PROCESSING BY-PASSING SUBSCRIPT
RESOLUTION (0-19)
OUTPUT
NONE
************
INTERNAL VARIABLES
(SEE XPGET)
************
MOVES
(SEE XPGET)
243 1 BEGIN XPGET
244 2 SET FOR "GET"
245 2 PERFORM ACCESS TO RETRIEVE DATA
246 1 END XPGET
247 1 BEGIN XPPUT
248 2 SET FOR "PUT"
249 2 PERFORM ACCESS TO STORE DATA
250 1 END XPPUT
251 1 BEGIN ACCESS
252 2 PERFORM SPI 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 XPRED TO PURGE QUEUED REQUESTS
256 4 EXIT TO :XPENT:
257 5 ENDIF
258 5 IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/"PUT" PROCESSING
259 6 THEN
260 6 CALL XPRED TO PURGE QUEUED REQUESTS
261 7 EXIT TO :XPENT:
262 8 ENDIF
263 8 IF DEBUGGING CALLING SEQUENCE
264 9 THEN
265 10 CALL XPRED TO PURGE QUEUED REQUESTS
266 11 EXIT TO :XPENT:
267 12 ENDIF
268 12 IF PROCESSING FOR "GET"
269 13 THEN
270 14 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
271 15 THEN
272 16 IF INPUT IN LITERAL FORM
273 17 THEN
274 18 BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT
275 19 ELSE
276 20 IF NORMAL XPGET/PUT PROCESSING (XPENF = -1)
277 21 THEN
278 22 IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
279 23 THEN
280 24 IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
281 25 THEN
282 26 PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
283 27 ELSE
284 28 DISPLACEMENT IS ZERO
285 29 ENDIF
286 30 ELSE
287 31 USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
288 32 ENDIF
289 33 BUILD REQUEST WITH DE NAME AND DISPLACEMENT
290 34 ENDIF
291 35 CALL XPRED TO QUEUE RETRIEVAL AND STORAGE OF INPUT
292 36 ELSE PARAMETER IS DRE (CLASS 3)
293 37 THEN
294 38 BUILD REQUEST FOR TOC ENTRY
295 39 CALL XPRED TO IMMEDIATELY RETRIEVE TOC ENTRY
296 40 PERFORM XPENF TO CONSTRUCT AND STORE QUALIFIED FILE NAME
297 41 "STORE FILE ATTRIBUTES"
298 42 ENDIF
299 43 ELSE PROCESSING FOR "PUT"
300 44 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
301 45 THEN
302 46 IF NORMAL XPGET/PUT PROCESSING (XPENF = -1)
303 47 THEN
304 48 ENDIF
305 49 ENDIF
306 49 ENDIF
307 49 ENDIF
IF SUBSCRIPTED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE ORDE
CALL XPREQ TO QUEUE REALLOCATION OF ORDE
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AWA
CALL XPREQ TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS ORDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD RFI ST TO REALLOCATE ORDE WITH NEW ATTRIBUTES
EXIT TO :ERR INJ: 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
TERMINATE PROCESSOR FOR INVALID ORDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
BEGIN XPLINI

INITIALIZE GLOBAL VALUES FROM LU AND XVSDB

TERMINATE PROCESSOR WITH XPL10 ERROR IF LU NOT IN XVSDB

IF INTERFACE TABLE BUFFER NOT INITIALIZED

THEN

RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER

IF RETRIEVAL NOT SUCCESSFUL

THEN

TERMINATE PROCESSOR WITH 'XPL10 PROCESSOR INITIALIZATION ERROR'

ENDIF

ENDIF

EXIT TO XPE13: IF N = 0

END XPLINI

BEGIN XPSFN

SET / IN FIRST CHARACTER POSITION

MOVE FOUR WORD NAME INTO MIDDLE POSITIONS

LOCATE FIRST BLANK CHARACTER

REPLACE BLANK WITH USER ID

END XPSFN

BEGIN XPSBC

IF DOUBLE SUBSCRIPTED

THEN

BUILD REQUEST FOR SUBSCRIPTS STORED IN LITERAL BLOCK

CALL XPEN 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 XPEN 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 XPSBC

END ACCESS
FORTAN CALLING PROCEDURE

CALL XPGT1 (LU, INBUF, INLIN, MBUFF, INUM, IN, SIZE, DISP)

XPGT1 ALLOWS THE CALLING PROGRAM TO RETRIEVE A SPECIFIED AMOUNT OF DATA FROM AN INPUT PARAMETER BEGINNING AT ANY POINT IN THE PARAMETER ARRAY.

**** INPUT ****

**LU** - LOGICAL UNIT NUMBER OF USER TERMINAL

**INBUF** - INPUT/OUTPUT BUFFER OF PARAMETERS + 1) WORDS,

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

**MBUFF** - 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 FROM WHICH TO RETURN DATA

**SIZE** - TOTAL NUMBER OF WORDS TO BE RETURNED

**DISP** - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER AT WHICH TO BEGIN DATA RETRIEVAL. 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., RETRIEVAL FROM AN AREA PREVIOUS TO THE USERS SUBSCRIPTS IS NOT SUPPORTED

**** OUTPUT ****

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

**** INTERNAL ****

**BSDSP** - DISPLACEMENT FROM THE REAL ORIGIN OF THE DATA ELEMENT AS A RESULT OF USER SUBSCRIBING. DISP IS ADDED TO THIS VALUE IN THE REQUEST TO THE MANAGER

**CPY** - DISPLACEMENT INTO JETPY DEPENDING ON WHETHER GET (D) OR PUT (I) PROCESSING IS INVOLVED

**JETPY** - TWO WORD VECTOR CONTAINING 'JSB XPGET' OR 'JSB XPFUT'

**XPPUT** - INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGET AND XPFUT CALLS

**** NOTES ****

*USES EXEC., ENTR, XPATR, XPE13(XPATR), XPGET(XPATR), XPFUT(XPATR), XPPUT(XPATR)*

****
CALL XPITI (LU, INBUF, INTLG, MRBUFF, INUM, OUT, SIZE, DISP)

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

XPITI allows the calling program to store a specified amount of data into an output parameter beginning at any point in the parameter array.

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

INPUT

LU - logical unit number of user terminal

INBUF - input/output buffer of \( \times \) (\# parameters + 1) words,

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

INTLG - first use to cause initialization

MRBUFF - manager request buffer (64 words) used to communicate with the FDS manager. May be used as a scratch area by

THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS

INUM - relative number of parameter in interface table into

OUT - which to store data

SIZE - array of at least 12 words from which output data is to be taken

DISP - total number of words to be stored

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 XPITI

-----------

NOTES

XPITI IS AN ENTRY POINT INTO XPITI

-----------
1 BEGIN XPGET
2 SET FOR GET PROCESSING
3 PERFORM XPXXI TO RETRIEVE DATA
4 END XPGET
5 BEGIN XPPTI
6 SET FOR PUT PROCESSING
7 PERFORM XPXXI TO STORE DATA
8 END XPPTI
9 BEGIN XPPI
10 SET XPSPF(XPGET) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
11 CALL XPATR TO ASSURE INITIALIZATION OF INBUF AND RETURN BASE DISPLACEMENT
12 EXIT TO :XPE15: (XPATR) IF INDICATED PARAMETER IS A DRE FILE
13 SAVE INTERFACE TABLE ENTRY CLASS/TYPED 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 INC BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
19 SET TYPE FIELD TO FREE
20 ELSEIF
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 XGPF(XPGET) TO NOMINAL VALUE
27 EXIT XPXXI
28 :ERR15: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
29 END XPXXI
CALLING PROCEDURE

JSB XPREQ

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

DEF OPN REQUIRED OPTION

DEF REXT REQUIRED REQUEST

DEF ADRES OPTIONAL ADDRESS (SEE OPN)

*******

XPGET/XPPUT BUFFERED AW MANAGEMENT SERVICE

IM-UT

O PTN - OPTON 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

REQUEST - EIGHT WORD REQUEST (SEE SDD 6.2-6.2)

ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT

VALUES CUSED ONLY WHEN OPN(RIGHT BIT) = 0

EXTERNAL SYMBOLS

EXTERNAL SYMBOLS

XPLU - TERMINAL LOGICAL UNIT NUMBER

XPRLY - ADDRESS OF 64 WORD MANAGER REQUEST BUFFER

XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

*******

OUTPUT

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

*******

LOCAL VARIABLES

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

ADDS - TABLE OF ADDRESSES FOR STORAGE OF FETCCHED DATA

APRMS - ADDRESS OF PARTS AREA

CLASS - CLASS NUMBER USED TO TRANSFER DATA TO MANAGER

PMS - PARAMETER RETURN AND SCRATCH AREA

PTR - SAVE AREA FOR ADDS AND XPRLB POINTER

*******

NOTES

USES .ENTR, EXE., XPSXT, XPSVW

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

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

CALL XPARS (LU, PRMLEN, PROMPT, TYPE, DATLEN, IDIM, DATA, RETCS)

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

INPUTS FROM CALLING SEQUENCE:

    LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
    PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER STRING USED FOR THE USER PROMPT
    PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
           0 - FREE
              1 - INTEGER
              2 - REAL
              3 - DOUBLE PRECISION
   CALLERS:
    STRING LENGTH 2
    STRING LENGTH 6
    STRING LENGTH 10
    STRING LENGTH 18
    STRING LENGTH 36
    STRING LENGTH 72
    DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE NUMBER OF WORDS.
    IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0 IF IT IS A VECTOR.

OUTPUTS FROM CALLING SEQUENCE:

DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE) AREA TO CONTAIN THE USER'S RESPONSE
RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO CALLER:
       0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
       1 - USER ENTERED I. CONTENTS OF BUFFER UNPREDICTABLE.
       2 - USER ENTERED A CR, THERE IS NO RESPONSE.
       3 - USER PROMPT WAS TOO LONG, MAXIMUM LENGTH IS 34 CHARACTERS OR 17 WORDS.
       4 - PARAMETER LIST IS INVALID. EITHER:
          A. IDIM LESS THAN ZERO
          B. DATLEN LESS THAN ONE
          C. INVALID TYPE SPECIFIED.

INTERNAL VARIABLES
DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT ELEMENT IS TO BE STOR

EFTYP - 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.

SKIND - COUNT OF NUMBER OF NESTED REPEATS AND INDEX TO THE CURRENT TOP OF THE STACK.

STKPN - 4 WORD ARRAY FOR STACK OF PARENTHESES FLAGS.

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
BEGIN STRING

EXIT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERRMSG

DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH

EXIT IF TYPE OF RESPONSE > TYPE REQUESTED AND

EXIT IF TYPE IS NOT FREE THEN

SET EFFECTIVE LENGTH = LENGTH REQUESTED

END IF

SET MESSAGE NUMBER TO XPOP

EXIT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERRMSG

CALL XMOV TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH

CALL XMOV TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH

SET PREVIOUS TOKEN IS DATA

INCREMENT POINTER IN DATA AREA

INCREMENT TO NEXT TOKEN

END STRING

BEGIN SUBSCR

INCREMENT POINTER TO NEXT TOKEN

SET MESSAGE NUMBER TO XPOP

EXIT IF TOKEN IS NOT AN INTEGER TO PERFORM ERRMSG

IF 1-DIMENSION > 1 THEN

SET I TO INTEGER VALUE

INCREMENT POINTER TO NEXT TOKEN

EXIT IF TOKEN IS NOT AN INTEGER OR

EXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG

SET MESSAGE NUMBER TO XPOP

EXIT IF INTEGER > 1-DIMENSION TO PERFORM ERRMSG

EXIT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERRMSG

ELSE

EXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG

SET MESSAGE NUMBER TO XPOP

EXIT IF SUBSCRIPT IS OUT OF RANGE

END IF

ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT

INCREMENT POINTER BY 3 TOKENS

SET PREVIOUS TOKEN = SUBSCRIPT

SET MESSAGE NUMBER TO XPOP

EXIT IF TOKEN IS AN EOS OR

EXIT IF TOKEN IS A REPEAT OR

EXIT IF TOKEN IS A CLOSE PAREN OR

EXIT IF TOKEN IS A SUBSCRIPT TO PERFORM ERRMSG

END SUBSCR

END STRING
1 BEGIN REPET
2 SET MESSAGE NUMBER TO XPOS
3 ERREXIT IF TOKEN IS AN EOS TO PERFORM ERRMSG
4 INCREMENT STACK POINTER
5 SET MESSAGE NUMBER TO XPOS
6 ERREXIT IF THERE ARE TOO MANY NESTED REPEATS PERFORM ERRMSG
7 PUSH REPEAT COUNT ON STACK
8 SET PARENTHESIS FLAG TO 0
9 IF TOKEN IS AN OPEN PARENTHESIS THEN
10 INCREMENT POINTER TO NEXT TOKEN
11 SET PARENTHESIS FLAG TO 1
12 ENDIF
13 PUSH TOKEN INDEX AND PAREN FLAG ON STACK
14 SET PREVIOUS TOKEN IS A REPEAT
15 END REPET
16
17 BEGIN ERRMSG
18 CALL EXEC TO WRITE ERROR MESSAGE
19 PERFORM XPRDS - NO RETURN
20 END ERRMSG
**FORTRAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE**

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

**CD01**

**W**RI**T**ES "PRMLEN" **W**OR**D**S **O**F "PROMPT" **TO** USER **R**EADS **T**HE USER'S **R**E**S**PONSE; **C**ONV**ERT**S **R**ESPONSE **T**O **T**OKENS **I**N "COMBU" **A**ND **P**ASSES **B**ACK A **R**ETUR**N** **C**ODE "RETC" **I**NDICATING **T**HE **S**UCCESS **O**F **XPRDM** **A**ND **T**HE **U**SEFULNESS **O**F "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 STRING USED FOR THE USER PROMPT.
- **PROMPT** - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING USED AS THE USER PROMPT.
- **COMLEN** - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE COMMUNICATIONS BUFFER (COMBUF)

**OUTPUTS FROM CALLING SEQUENCE:**

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

**INTERNAL VARIABLES**

- **COUNT** - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN STRING AND NUMER DIGITS IN A NUMBER.
- **DBLMRT** - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE.
- **DBLVDD** - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE.
- **VALUE** - (INTEGER, 1 WORD) VALUE.
- **FLGCOM** - COMMA FLAG
- **Q** - LAST CHARACTER NOT A COMMA
- **FLGCOM** - CONTINUC FLAG
- **Q** - THIS IS NOT A RESPONSE TO A CONTINUE
- **T** - THIS IS A CONTINUC RESPONSE
- **FLGNUM** - NUMBER FLAG
- **Q** - POSITIVE NUMBER
- **T** - NEGATIVE NUMBER
- **FLGPOW** - POWER FLAG
- **Q** - POSITIVE POWER
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>881</td>
<td>1 CD4</td>
<td>1 - NEGATIVE POWER</td>
</tr>
<tr>
<td>882</td>
<td>1 CD4</td>
<td>FLTYP - TYPE OF REAL NUMBER</td>
</tr>
<tr>
<td>883</td>
<td>1 CD4</td>
<td>0 - SINGLE PRECISION</td>
</tr>
<tr>
<td>884</td>
<td>1 CD4</td>
<td>1 - DOUBLE PRECISION</td>
</tr>
<tr>
<td>885</td>
<td>1 CD4</td>
<td></td>
</tr>
<tr>
<td>886</td>
<td>1 CD4</td>
<td></td>
</tr>
<tr>
<td>887</td>
<td>1 CD5</td>
<td>SUBROUTINES AND FUNCTIONS USED:</td>
</tr>
<tr>
<td>888</td>
<td>1 CD5</td>
<td>EXEC, CVX, VTRMV, XRCPK, XRUPK, OVF</td>
</tr>
<tr>
<td>889</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>890</td>
<td>1 CD5</td>
<td>PDL ROUTINES USED:</td>
</tr>
<tr>
<td>891</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>892</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>893</td>
<td>1 CD5</td>
<td>XPROM, TOKENS, QUOTE, DIGIT, DCOL, DECP, EORD,</td>
</tr>
<tr>
<td>894</td>
<td>1 CD5</td>
<td>INTGR, REAL, DBL, INVAL, COMPUL</td>
</tr>
<tr>
<td>895</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>896</td>
<td>1 CD**</td>
<td></td>
</tr>
</tbody>
</table>
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 :COMLOP:
8 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
9 CALL EXEC TO READ RESPONSE
10 CALL XRPK TO CONVERT A2 RESPONSE TO R1 FORMAT
11 IF NUMBER OF WORDS READ IS NOT ZERO THEN
12 SET COMMA FLAG ON
13 DO WHILE CONTINUE FLAG IS OFF
14 IF INPUT BUFFER IS COMPLETELY SCANNED THEN
15 SET CONTINUE FLAG ON
16 IF COMMA FLAG IS ON THEN
17 CALL EXEC TO WRITE CONTINUE
18 GO TO :COMLOP:
19 ELSE
20 EXIT IF COMBUF IS FULL PERFORM COMFUL
21 IF INPUT CHARACTER IS A COMMA THEN
22 IF COMMA FLAG IS ON THEN
23 ERREXIT IF COMBUF CANNOT HOLD TOKEN PERFORM COMFUL
24 STORE NULL FIELD TOKEN IN COMBUF
25 INCREMENT #WORDS IN COMBUF BY 1
26 INCREMENT #TOKENS IN COMBUF BY 1
27 ENDIF
28 SET COMMA FLAG ON
29 GET NEXT INPUT CHARACTER
30 ELSE
31 SET COMMA FLAG OFF
32 PERFORM TOKENS
33 ENDIF
34 ENDIF
35 ELSE
36 IF CONTINUE FLAG IS OFF THEN
37 SET XPRDM RETURN CODE TO SAY USER ENTERED CR
38 ENDIF
39 ENDIF
40 ELSE
41 SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
42 ENDIF
43 1 END XPRDM
945 1 BEGIN TOKENS
946 2  SET NEGATIVE NUMBER FLAG OFF
947 3  SET NEGATIVE POWER FLAG OFF
948 4  SET POWER = 0
949 5  IF INPUT CHARACTER IS A DIGIT THEN
950 6      PERFORM DIGIT
951 7      ELSE
952 8      X " ( ) + -
953 9      CASE J (:A;:B;:C;:D;:F;:G;:I;)
954 10     :A:
955 11     SET XTPRM RETURN CODE TO SAY I ENTERED
956 12     SET CONTINUE FLAG ON
957 13     :B:
958 14     PERFORM QUOTE
959 15     :C:
960 16     SET PAREN INDICATOR = 0
961 17     GO TO :E:
962 18     :D:
963 19     SET PAREN INDICATOR = 1
964 20     GO TO :E:
965 21     :E:
966 22     ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN TO PERFORM COMFUL
967 23     SIZE TOKEN (OPAR + PAREN INDICATOR) IN COMBUF
968 24     INCREMENT #WORDS IN COMBUF BY 1
969 25     INCREMENT #TOKENS IN COMBUF BY 1
970 26     GET NEXT CHARACTER
971 27     :F:
972 28     ERREXIT IF NEXT CHARACTER IS NOT A DIGIT PERFORM INVAL
973 29     SET INTEGER = 0
974 30     PERFORM DECP
975 31     :G:
976 32     IF INPUT CHARACTER IS A - THEN
977 33     SET NEGATIVE NUMBER FLAG ON
978 34     ENDF
979 35     GET NEXT CHARACTER
980 36     IF INPUT CHARACTER IS A DIGIT THEN
981 37     PERFORM DIGIT
982 38     ELSE
983 39     IF INPUT CHARACTER IS A . THEN
984 40     GO TO :F:
985 41     ELSE
986 42     PERFORM INVAL - NO RETURN
987 43     ENDF
988 44     ENDF
989 45     ENDCASE
990 46     ENDF
991 47     END TKNCS
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 TEMPORARY BUFFER (#CHARACTERS)
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
GET NEXT CHARACTER
ENDDO
ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
ERREXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
STORE CHARACTER STRING TOKEN IN COMBUF
STORE NUMBER OF CHARACTERS IN COMBUF
CALL XRPCK TO CONVERT CHARACTERS FROM R1 TO A2 FORMAT
INCREMENT #WORDS IN COMBUF BY 2* (#CHARACTERS + 1) / 2
INCREMENT #TOKENS IN COMBUF BY 1
GET NEXT CHARACTER
END QUOTE
1019 1 BEGIN DIGIT
1020 2 PERFORM DCOL
1021 3 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1022 4 IF INPUT CHARACTER IS A ' ' THEN
1023 5 PERFORM DECPT
1024 6 ELSE IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1025 7 PERFORM END
1026 8 ELSE
1027 9 IF INPUT CHARACTER IS AN "R" THEN
1028 10 ERREXIT IF THERE IS NO ROOM IN CONBUF FOR THIS TOKEN PERFORM CONFUL
1029 11 ERROR IF INTEGER IS ZERO PERFORM INVALID
1030 12 STORE REPEAT TOKEN IN CONBUF
1031 13 INCREMENT #WORDS IN CONBUF BY 2
1032 14 INCREMENT TOKENS BY 1
1033 15 GET NEXT CHARACTER
1034 16 ELSE
1035 17 ENDIF
1036 18 ENDIF
1037 19 IF INTEGER ENDIF
1038 20 ELSE
1039 21 ENDIF
1040 22 IF INTEGER
1041 23 PERFORM INTEGER
1042 24 ELSE
1043 25 ENDIF
1044 26 END IF
1045 27 BEGIN DCOL
1046 28 SET INTEGER = D
1047 29 SET COUNTER = D
1048 30 DO WHILE CHARACTER IS A DIGIT AND
1049 31 WHILE INPUT BUFFER IS NOT EXHAUSTED
1050 32 SET INTEGER = (INTEGER * 10) + INPUT CHARACTER - 48
1051 33 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1052 34 INCREMENT COUNTER BY 1
1053 35 GET NEXT CHARACTER
1054 36 END DO
1055 37 END DCOL
1056 38 END DCOL
1057 39 END DCOL
1059    1 BEGIN DECPT
1060    2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061    3 GET NEXT CHARACTER
1062    2 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063    3    IF INPUT CHARACTER IS A DIGIT THEN
1064    4    PERFORM DCOL
1065    4    ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066    4    EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067    3    ENDIF
1068    3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069    4    PERFORM 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 EORD
1078 IF INPUT CHARACTER IS A "E" THEN
1079 SET TYPE FLAG TO "E"
1080 ELSE
1081 SET TYPE FLAG TO "0"
1082 ENDIF
1083 GET NEXT CHARACTER
1084 ERREXIT IF INPUT BUFFER IS EXHAUSTED PERFORM INVAL
1085 IF INPUT CHARACTER IS A - THEN
1086 SET NEGATIVE POWER FLAG ON
1087 GET NEXT CHARACTER
1088 ELSE
1089 IF INPUT CHARACTER IS A + THEN
1090 SET NEGATIVE POWER FLAG OFF
1091 GET NEXT CHARACTER
1092 ENDIF
1093 ENDIF
1094 ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
1095 ERREXIT IF INPUT CHARACTER IS NOT A DIGIT PERFORM INVAL
1096 PERFORM DECPL
1097 IF NEGATIVE POWER FLAG IS ON THEN
1098 SET POWER = -POWER
1099 ENDIF
1100 IF TYPE FLAG IS "E" THEN
1101 PERFORM REAL
1102 ENDIF
1103 ELSE
1104 BEGIN DBL
1105 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
1106 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
1107 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
1108 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL
1109 ERREXIT IF DOUBLE OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1110 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1111 SET DOUBLE = -DOUBLE
1112 ENDIF
1113 STORE DOUBLE TOKEN IN COMBUF
1114 INCREMENT #TOKENS IN COMBUF BY 1
1115 INCREMENT #WORDS IN COMBUF BY 4
1116 END DBL
1117 ENDIF
1118 1 END EORD
1 BEGIN INTEGER
2 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
3 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
4 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
5 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
6 CONVERT DOUBLE TO INTEGER
7 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM 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 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
19 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
20 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
21 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
22 SET REAL = DOUBLE * 10 ** POWER
23 ERREXIT 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 OVERFLOW
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 OVERFLOW
1074 1 CD0  FORTRAN CALLING PROCEDURE
1075 1 CD0  CALL XPIT (LJ, RPAM5)
1076 1 CD0  
1077 1 CD0  *********
1078 1 CD0  *********
1079 1 CD0  TERMINATE A PROGRAM AFTER WAITING ON ANY BUFFERED I/O TO Lj TO
1080 1 CD0  COMPLETE THEN PASS PARAMETERS BACK TO THE FATHER TASK.
1081 1 CD0  *********
1082 1 CD0  *********
1083 1 CD0  INPUT
1084 1 CD0  LU - LOGICAL UNIT OF TERMINAL OR OTHER OUTPUT DEVICE FOR WHICH
1085 1 CD0  PRINT MAY NOT BE COMPLETED. A VALUE OF ZERO WILL
1086 1 CD0  INDICATE NO I/O WAIT
1087 1 CD0  RPAM5 - ARRAY OF FIVE PARAMETERS TO BE RETURNED TO FATHER TASK
1088 1 CD0  *********
1089 1 CD0  *********
1090 1 CD0  OUTPUT
1091 1 CD0  RPAM5 IS RETURNED TO THE FATHER TASK
1092 1 CD0  *********
1093 1 CD0  *********
1094 1 CD0  NOTES
1095 1 CD0  USES EXEC, PRN
1096 1 CD0  *********
1097 1 CD0  XPIT DOES NOT RETURN TO THE CALLER.
1098 1 CD0  *********
1099 1 CD0  1200
1100 1 CD0  1201 1 * 1202
1101 1 CD0  1203 1 * 
1102 1 CD0  1204 1 *
1103 1 CD0  1205 1 BEGIN XPIT
1104 1 CD0  1206 2 IF Lj IS NON-ZERO
1105 1 CD0  1207 2 THEN
1106 1 CD0  1208 3 WAIT ON ANY CLASS I/O TO COMPLETE (CLASS I/O CONTROL THEN GET)
1107 1 CD0  1209 3 ENDIF
1108 1 CD0  1210 2 RETURN PARAMETERS TO FATHER
1109 1 CD0  1211 2 TERMINATE PROGRAM
1110 1 CD0  1212 1 END XPIT
**FORTRAN CALLING PROCEDURE**

1.00 CALL XRBIT (BIT, BITNUM, STRING)

**********

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

**********

1.02 INPUT

1.02 BIT - INTEGER VALUE OF ZERO OR ONE TO BE SET IN BIT NUMBER

1.02 BITNUM - INTEGER BIT NUMBER OF BIT TO BE SET/CLEARED (FIRST BIT OF STRING IS BIT NUMBER ZERO)

1.02 STRING - BIT STRING OF AT LEAST 'BITNUM' BITS IN LENGTH INTO WHICH 'BIT' IS TO BE SET

**********

1.03 OUTPUT

1.03 STRING - BIT NUMBER 'BITNUM' HAS THE VALUE OF 'BIT'

**********

2.05 NOTES

2.05 USES .ENTR, XRSET

**********

1.04 BEGIN XRBIT

2.02 TRANSFER CALLING SEQUENCE

2.02 COMPUTE ADDRESS OF WORD CONTAINING BITNUM

2.02 COMPUTE BIT NUMBER WITHIN WORD

2.02 CALL XRSET TO SET/CLEAR BIT

1.04 END XRBIT
INTEGER FUNCTION

XRCPR(LENGTH, ARRAY1, ARRAY2)

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

1 001.

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

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

1 002.

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

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

1 003.

OUTPUT

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

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

1 004.

NOTES

1 005.

USES .ENTR

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

1 010.

BEGIN XRCPR

2 TRANSFER CALLING SEQUENCE

2 SETUP COMPARE

2 COMPARE ARRAY1 AND ARRAY2

2 RETURN RESULT FLAG

1 END XRCPR
80 1 CDO  FORTRAN CALLING PROCEDURE.
81 1 CDO
82 1 CDO  CALL XR018 (DOUBLE, ASCII)
83 1 CDO
84 1 CDO
85 1 CDO  CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN
86 1 CDO  FP018.1F FORMAT
87 1 CDO
88 1 CDO
89 1 CDO  INPUT
90 1 CDO  DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED
91 1 CDO
92 1 CDO
93 1 CDO
94 1 CDO  OUTPUT
95 1 CDO  ASCII - NINE WORD ASCII CHARACTER STRING REPRESENTATION OF
96 1 CDO  'DOUBLE'
97 1 CDO
98 1 CDO
99 1 CDO  LOCAL
100 1 CDO  D - WORKING LOCATION FOR ABSOLUTE VALUE OR 'DOUBLE'
101 1 CDO
102 1 CDO
103 1 CDO
104 1 CDO  NOTES
105 1 CDO  USES DBLE, FLOAT, IAN0, IDINT, IOR, KCVT, XREXT
106 1 CDO
107 1 CDO
108 1 *
109 1 *
110 1 *
111 1 *
112 1 BEGIN XR018
113 2  SET SIGN FIELD
114 2  MOVE ABSOLUTE VALUE OF 'DOUBLE' INTERNAL
115 2  COMPUTE EXPONENT
116 2  SET SIGN AND VALUE OF EXPONENT FIELD
117 2  REDUCE VALUE TO RANGE OF 1 < VALUE <10
118 2  EXTRACT FIRST DIGIT, MERGE WITH SIGN AND STORE FIELD
119 2  EXTRACT SECOND DIGIT, MERGE WITH DECIMAL AND STORE FIELD
120 2  DO FOR NEXT FIVE PAIRS OF DIGITS
121 3  MULTIPLY BY 100 TO EXTRACT PAIR
122 3  EXTRACT DIGITS AND STORE FIELD
123 2  ENDO00
124 1 END XR018
FORTRAN CALLING PROCEDURE

1 CD0
2 CD3
3 CD0
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0
9 CD0
10 CD0
11 CD0
12 CD0
13 CD0
14 CD0
15 CD0
16 CD0
17 CD0
18 CD0
19 CD0
20 CD0
21 CD0
22 CD0
23 CD0
24 CD0
25 CD0
26 CD0
27 CD0
28 CD0
29 CD0
30 CD0
31 CD0
32 CD0
33 CD0
34 CD0
35 CD0
36 CD0
37 CD0
38 CD0
39 CD0
40 CD0
41 CD0
42 CD0
43 CD0
44 CD0
45 CD0
46 CD0
47 CD0
48 CD0
49 CD0
50 CD0
51 CD0
52 CD0
53 CD0
54 CD0
55 CD0
56 CD0
57 CD0
58 CD0
59 CD0
60 CD0
61 CD0
62 CD0
63 CD0
64 CD0
65 CD0
66 CD0
67 CD0
68 CD0
69 CD0
70 CD0
71 CD0
72 CD0
73 CD0
74 CD0
75 CD0
76 CD0
77 CD0
78 CD0
79 CD0
80 CD0
81 CD0
82 CD0
83 CD0
84 CD0
85 CD0
86 CD0
87 CD0
88 CD0
89 CD0
90 CD0
91 CD0
92 CD0
93 CD0
94 CD0
95 CD0
96 CD0
97 CD0
98 CD0
99 CD0
100 CD0
101 CD0
102 CD0
103 CD0
104 CD0
105 CD0
106 CD0
107 CD0
108 CD0
109 CD0
110 CD0
111 CD0
112 CD0
113 CD0
114 CD0
115 CD0
116 CD0
117 CD0
118 CD0
119 CD0
120 CD0
121 CD0
122 CD0
123 CD0
124 CD0
125 CD0
126 CD0
127 CD0
128 CD0
129 CD0
130 CD0
131 CD0
132 CD0
133 CD0
134 CD0
135 CD0
136 CD0
137 CD0
138 CD0
139 CD0
140 CD0
141 CD0
142 CD0
143 CD0
144 CD0
145 CD0
146 CD0
147 CD0
148 CD0
149 CD0
150 CD0
151 CD0
152 CD0
153 CD0
154 CD0
155 BEGIN XREQ
156 2 PERFORM TRACE
157 2 OUTPUT TRACING TO MANAGER
158 2 REQUEST AREA MANAGEMENT AND WAIT FOR RESPONSE
159 2 RETRIEVE MANAGER RESPONSE
160 2 PERFORM TRACE
161 2 RETURN RESPONSE IN REQPR
162 1 END XREQ
163 1 BEGIN TRACE
164 2 IF TRACE REQUESTED
165 2 THEN
166 3 DO FOR EACH REQUEST
167 4 CALL XUDPL TO FORMAT LINE
168 4 OUTPUT LINE
169 3 ENDDO
170 3 OUTPUT PARM1 AND PARM2
171 2 ENDDO
172 1 END TRACE
INTEGRATOR FUNCTION

INPUT: START, LENGTH, SOURCE

1. EXTRACT LENGTH BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
2. INPUT OPERANDS INTO PROGRAMMING LEFT MOST BIT OF FIELD TO BE EXTRACTED
3. LENGTH: POSITIVE INTEGRATOR SIZE OF FIELD TO BE EXTRACTED
4. USES: INTR

BEGIN INTEGRATOR CALLING SEQUENCE
IF START NOT = 0

LOAD A WITH SOURCE
CONSTRUCT SHIFT
ELSE LOAD A WITH SOURCE
SAVE A
END IF

CONSTRUCT SHIFT
RESTORE A, EFFECT LENGTH BITS
MOVE RESULT FROM A TO A
END INTEGRATOR
FORTRAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

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

REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED

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

LOCAL

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

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

** NOTES

BEGIN XRE14

SET SIGN FIELD

MOVE ABSOLUTE VALUE OF 'REAL' INTERNAL

COMPUTE EXPONENT

SET SIGN AND VALUE OF EXPONENT FIELD

REDUCE VALUE TO RANGE OF 1 <= VALUE < 10

EXTRACT FIRST DIGIT AND STORE FIELD

SET DECIMAL FIELD

DO FOR NEXT THREE PAIRS OF DIGITS

MULTIPLY BY 100 TO EXTRACT PAIR

EXTRACT DIGITS AND STORE FIELD

END XRE14
FORTRAN CALLING PROCEDURE

CALL XRI6 (INTEGER, ASCII)

**CD1**

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

**CD2**

INPUT

**CD3**

OUTPUT

**CD4**

LOCAL

**CD5**

USES XMOV AND XRCK

**CD6**

BEGIN XRI6

**CD7**

BLANK WORKING SPACE

**CD8**

CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING

**CD9**

SET SIGN OF 'INTEGER' IN 'ASCII'

**CD10**

CALL XRCK TO CONVERT FROM R1 TO A2 FORMAT
FORTRAN CALLING PROCEDURES

CALL XRLOCK (RCODE)
CALL XRLK (RCODE)

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

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

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

** INPUT
** YVSTB RESOURCE NUMBER
**

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

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

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

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

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

** NOTES
**

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

THIS ROUTINE MAY NOT BE OVERLAYED
1 BEGIN XLCK
2 SET REQUEST FOR LOCK FUNCTION
3 PERFORMロック
4 END XLCK
5 BEGIN XLULK
6 SET REQUEST FOR UNLOCK FUNCTION
7 PERFORMロック
8 END XLULK
9 BEGIN LOCK
10 IF REQUEST IS CONSISTENT WITH STATUS
11 THEN
12 SET NEW STATUS
13 IF RM IN STG IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
14 THEN
15 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
16 ENDIF
17 CLEAR RETURN CODE
18 ELSE
19 SET RETURN CODE
20 ENDIF
21 END LOCK
22
389 1 *00  FORTRAN CALLING PROCEDURE
390 1 *00
391 1 *00  CALL XRMOV (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 MOV.
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 *05  NOTES
408 1 *05  USES .ENTR
409 1 *05
410 1 *05
411 1 *
412 1 *
413 1 *
414 1 *
415 1 BEGIN XRMOV
416 1 TRANSFER CALLING SEQUENCE
417 1 INITIALIZE MOVE
418 1 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
419 1 END XRMOV
FORTRAN CALLING PROCEDURE

CALL XMSG (NUMBER, LOCATE, LENGTH, SOURCE)

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

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

TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL.

INPUT

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

A - AREA INDICATOR AS FOLLOWS

1 - AS

2 - XD

3 - XE

4 - XI

5 - XS

6 - YT

7 - RX

8 - XL

9 - DF

10 - SC

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

WORDS OF 'SOURCE' TO BE OUTPUT.

LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO 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)

COMMON

LU - USERS LOGICAL UNIT NUMBER

=====

OUTPUT

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

=== ANN MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE

NOTES

USES FDS SYSTEM MESSAGE FILE XMSG

USES CLOSE, EXEC, JAVA, KCUT, OPEN, READY, XMOV, XUDG

=====
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>470</td>
<td>BEGIN XMSG</td>
<td>BEGIN XMSG</td>
</tr>
<tr>
<td>471</td>
<td>SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER</td>
<td>SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER</td>
</tr>
<tr>
<td>472</td>
<td>SET NUMBER IN PREFIX</td>
<td>SET NUMBER IN PREFIX</td>
</tr>
<tr>
<td>473</td>
<td>READ MESSAGE DIRECTORY RECORD</td>
<td>READ MESSAGE DIRECTORY RECORD</td>
</tr>
<tr>
<td>474</td>
<td>IF AREA VALID THEN</td>
<td>IF AREA VALID THEN</td>
</tr>
<tr>
<td>475</td>
<td>SET AREA CODE IN PREFIX IF MESSAGE NUMBER &gt; 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</td>
<td>SET AREA CODE IN PREFIX IF MESSAGE NUMBER &gt; 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</td>
</tr>
<tr>
<td>477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>478</td>
<td></td>
<td></td>
</tr>
<tr>
<td>479</td>
<td></td>
<td></td>
</tr>
<tr>
<td>480</td>
<td></td>
<td></td>
</tr>
<tr>
<td>481</td>
<td></td>
<td></td>
</tr>
<tr>
<td>482</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483</td>
<td></td>
<td></td>
</tr>
<tr>
<td>484</td>
<td></td>
<td></td>
</tr>
<tr>
<td>485</td>
<td></td>
<td></td>
</tr>
<tr>
<td>486</td>
<td></td>
<td></td>
</tr>
<tr>
<td>487</td>
<td></td>
<td></td>
</tr>
<tr>
<td>488</td>
<td>CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)</td>
<td>CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)</td>
</tr>
<tr>
<td>489</td>
<td>IF MESSAGE NUMBER &gt; 0 THEN CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS) ELSE ENDIF</td>
<td>IF MESSAGE NUMBER &gt; 0 THEN CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS) ELSE ENDIF</td>
</tr>
<tr>
<td>490</td>
<td></td>
<td></td>
</tr>
<tr>
<td>491</td>
<td></td>
<td></td>
</tr>
<tr>
<td>492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>493</td>
<td></td>
<td></td>
</tr>
<tr>
<td>494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>495</td>
<td>:ERROR: CALL XRMV TO MOVE 'XMSG ERROR' INTO BUFFER</td>
<td>:ERROR: CALL XRMV TO MOVE 'XMSG ERROR' INTO BUFFER</td>
</tr>
<tr>
<td>496</td>
<td>CALL XRMV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)</td>
<td>CALL XRMV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)</td>
</tr>
<tr>
<td>497</td>
<td></td>
<td></td>
</tr>
<tr>
<td>498</td>
<td></td>
<td></td>
</tr>
<tr>
<td>499</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>501</td>
<td></td>
<td></td>
</tr>
<tr>
<td>502</td>
<td></td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>END XMSG</td>
<td>END XMSG</td>
</tr>
</tbody>
</table>
INTEGER FUNCTION

RENBF (BIT, BITNUM, STRING)

***********

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

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

***********

#5 OUTPUT
#5 FUNCTION VALUE - BIT NUMBER OF NEXT OCCURRENCE OF 'BIT' >= 'BITNUM'

***********

#5 NOTES

#5 USES .ENTR

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

#5 A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4096 WORDS) CAN BE
#5 MEANINGFULLY ACCOMMODATED.
1 BEGIN XRNKB
2 COMPUTE STARTING WORD NUMBER
3 COMPLEX STARTING BIT NUMBER
44 LOAD STARTING WORD
45 SHIFT WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
46 INITIALIZE BIT COUNT
47 DO WHILE SIGN BIT IS NOT EQUAL TO BIT
48 IF WORD NOT FINISHED
49 THEN
50 SHIFT WORD LEFT
51 INCREMENT BIT COUNT
52 ELSE
53 DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
54 LOAD NEXT WORD
55 ENDDO
56 CLEAR BIT COUNT
57 ENDDO
58 ENDDO
59 RETURN VALUE OF MATCHING BIT NUMBER
60 END XRNKB
1     40     FORTRAN CALLING PROCEDURE
2     40     CALL XRO6 (BINARY, OCTAL)
3     40     ********
4     41     *1     CONVERT A WORD FROM BINARY TO SIX CHARACTER OCTAL REPRESENTATION
5     42     ********
6     43     *2     INPUT
7     44     *2     BINARY - BINARY WORD TO BE CONVERTED
8     45     ********
9     46     *3     OCTAL - THREE WORD ARRAY CONTAINING OCTAL REPRESENTATION OF
10     47     "BINARY" IN ASCII FORMAT (06)
11     48     ********
12     49     *5     NOTES
13     50     *5     USES ENTR
14     51     *5     ********
15     52     40     **
16     53     40     40
17     54     40     BEGIXRO6
18     55     40     2     TRANSFER CALLING SEQUENCE
19     56     40     2     FORM 18-BIT WORD USING 8 AND 2 MOST SIGNIFICANT BITS OF A
20     57     40     2     SET BYTE FLAG HIGH
21     58     40     2     DO FOR EACH PAIR OF OCTAL DIGITS (3)
22     59     40     3     SET PREFIX BITS (000000)
23     60     40     3     SHIFT IN DIGIT
24     61     40     3     IF BYTE FLAG SET HIGH
25     62     40     3     THEN
26     63     40     4     SHIFT FOR ACCOMODATION OF LOW BYTE
27     64     40     4     ELSE
28     65     40     4     STORE PAIR OF DIGITS IN OCTAL(1)
29     66     40     4     CLEAR FOR NEXT PAIR OF DIGITS
30     67     40     4     ENDIF
31     68     40     3     FLIP BYTE FLAG
32     69     40     2     ENDRO6
33     70     1     ENDRO6
FORTRAN CALLING PROCEDURE

CALL XRPACK (LENGTH, UNPKED, PACKED)

**********

**1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39**  **40**  **41**  **42**  **43**  **44**  **45**  **46**  **47**  **48**  **49**  **50**  **51**  **52**  **53**  **54**  **55**  **56**  **57**  **58**  **59**  **60**  **61**  **62**  **63**  **64**  **65**

**********

**1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39**  **40**  **41**  **42**  **43**  **44**  **45**  **46**  **47**  **48**  **49**  **50**  **51**  **52**  **53**  **54**  **55**  **56**  **57**  **58**  **59**  **60**  **61**  **62**  **63**  **64**  **65**
FORTRAN CALLING PROCEDURE

CALL XRFM (PREFIX, NAME4, NAME6)

*********

XRFM builds a qualified file name of up to six characters in
length by prefixing the input one to four character name with
the prefix character and appending a user qualifier code to the
end.

*********

INPUT

PREFIX - file type prefix stored in R1 format
NAME4 - one to four character packed name to be qualified
COMMON XE - QUAL

*********

OUTPUT

NAME6 - three to six character packed qualified name

*********

NOTES

USES .ENTR

*********

*

BEGIN XRFM

STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING
MOVE NAME4 INTO NEXT FOUR POSITIONS
STORE BLANK IN SIXTH POSITION
LOCATE FIRST BLANK CHARACTER
REPLACE BLANK WITH USER ID CHARACTER (QUAL)
MOVE QUALIFIED NAME TO NAME6

END XRFM
FORTRAN CALLING PROCEDURE

1 *00 CALL XRSET (START, LENGTH, SOURCE, OBJECT)
1 *00
1 *00
1 *00
1 *00
1 *00
1 *00
1 *00
1 *00

1 *01 REPLACE 'LENGTH' BITS OF 'OBJECT', BEGINNING WITH BIT 'START',
1 *01 WITH THE RIGHT MOST 'LENGTH' BITS OF 'SOURCE'
1 *01
1 *01
1 *01
1 *01
1 *01
1 *01
1 *01

1 *02 INPUT
1 *03 START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE REPLACED
1 *04 (SIGN BIT = 0)
1 *05
1 *05 LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED
1 *05
1 *05 SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED
1 *05
1 *05
1 *05
1 *05
1 *05
1 *05

1 *03 OUTPUT
1 *04 OBJECT - FIELD INTO WHICH FIELD IS TO BE INSERTED
1 *04
1 *04
1 *04

1 *05 NOTES
1 *05 USES .ENTR
1 *05
1 *05
1 *05

1 *05 BEGIN XRSET
1 *05
1 *05
1 *05
1 *05
1 *05

1 *06 TRANSFER CALLING SEQUENCE
1 *07 CONSTRUCT SHIFT INSTRUCTIONS
1 *07 SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER
1 *07 SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD
1 *07 CONSTRUCT MASK AND CLEAR FIELD OF OBJECT
1 *07 INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT
1 *07 END XRSET
INTEGER FUNCTIONS

1 *DO
2 *DO
3 XRSFL(COUNT, SOURCE)
4 XMSFR(COUNT, SOURCE)

1 *DO
2 SHIFT 'SOURCE' LEFT/RIGHT LOGICALLY 'COUNT' BITS
3 *DO
4 **********
5 INPUT
6 COUNT - POSITIVE INTEGER SPECIFYING NUMBER OF BITS TO SHIFT
7 SOURCE - WORD TO BE SHIFTED
8 *
9 **********
10 NOTES
11 USES .ENTR
12 *
13 **********
14 *
15 *
16 BEGIN XRSFL
17 SET FOR LEFT SHIFT
18 PERFORM SHIFT (FLAG, COUNT, SOURCE)
19 END XRSFL
20 BEGIN XRSFL
21 SET FOR RIGHT SHIFT
22 PERFORM SHIFT (FLAG, COUNT, SOURCE)
23 END XRSFL
24 BEGIN SHIFT
25 TRANSFER CALLING SEQUENCE
26 CONSTRUCT SHIFT INSTRUCTION
27 LOAD A WITH SOURCE
28 CLEAR B
29 SHIFT BA AS SPECIFIED
30 END SHIFT
1 *DO FORTRAN CALLING PROCEDURE
2 *DO CALL XRUNG (NAME6, NAME4)
3 *DO
4 ********
5 *DO XRUNG REMOVES THE PREFIX AND SUFFIX QUALIFYING CHARACTERS FROM
6 *DO A SIX CHARACTER FILE NAME
7 *DO
8 ********
9 *DO INPUT
10 *DO NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME
11 *DO
12 ********
13 *DO OUTPUT
14 *DO NAME4 - ONE TO FOUR CHARACTER PACKED NAME WITH PREFIX AND SUFFIX
15 *DO REMOVED
16 *DO
17 ********
18 *DO NOTES
19 *DO USES .ENTR
20 *DO
21 ********
22 *DO
23 *DO
24 *DO
25 *DO
26 *DO
27 *DO
28 *DO
29 *DO
30 *DO
31 *DO
32 *DO
33 *DO
34 *DO
35 *DO
36 *DO
37 *DO
38 *DO
39 *DO
40 *DO
41 *DO
42 *DO
43 *DO
44 *DO
45 *DO
46 *DO
47 *DO
48 *DO
49 *DO
50 *DO
51 *DO
52 *DO
53 *DO
54 *DO
55 *DO
56 *DO
57 *DO
58 *DO
59 *DO
60 *DO
61 *DO
62 *DO
63 *DO
64 *DO
65 *DO
66 *DO
67 *DO
68 *DO
69 *DO
70 *DO
71 *DO
72 *DO
73 *DO
74 *DO
75 *DO
76 *DO
77 *DO
78 *DO
79 *DO
80 *DO
81 *DO
82 *DO
83 *DO
84 *DO
85 *DO
86 *DO
87 *DO
88 *DO
89 *DO
90 *DO
91 *DO
92 *DO
93 *DO
94 *DO
95 *DO
96 *DO
97 *DO
98 *DO
99 *DO
100 1 BEGIN XRUNG
101 2 MOVE CHARACTERS 2-5 OF NAME6 INTO NAME4
102 2 IF SIXTH CHARACTER IS BLANK, THEN
103 3 LOCATE LAST NON-BLANK CHARACTER OF NAME4
104 3 BLANK THAT CHARACTER
105 2 ENDIF
106 1 END XRUNG
FORTRAN CALLING PROCEDURE

CALL XRUPK (LENGTH, PACKED, UNPKED, COUNT)

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

CONVERT 'LENGTH' WORDS OF 'PACKED' FROM A2 FORMAT TO R1 FORMAT,

REMOVING BLANKS NOT DELIMITED BY QUOTE MARKS, AND RETURN IN

'UNPKED' WITH THE NON-BLANK CHARACTER COUNT IS 'COUNT'.

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

INPUT

LENGTH - POSITIVE INTEGER NUMBER OF WORDS IN PACKED

PACKED - ARRAY OF CHARACTER DATA I' A2 FORMAT

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

OUTPUT

UNPKED - ARRAY OF NON-BLANK CHARACTERS IN R1 FORMAT

COUNT - NUMBER OF CHARACTERS IN UNPKED

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

NOTES

USES: .ENTR

IF PACKED AND UNPKED ARE THE SAME ADDRESS SPACE UNPKED MAY OVERLAY

PACKED

** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER

STRINGS.

************
839 1 BEGIN XRUNK
840 2 TRANSFER CALLING SEQUENCE
841 2 TURN ON BLANK REMOVAL
842 2 INITIALIZE COUNT
843 3 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 5 THEN
854 5 IF A NOT = BLANK
855 5 THEN
856 6 INCREMENT COUNT
857 6 STORE A IN UNPKED
858 5 ENDIF
859 4 ELSE
860 5 INCREMENT COUNT
861 5 STORE A IN UNPACKED
862 4 ENDIF
863 4 RELOAD A WITH WORD
864 3 ENDDO
865 2 ENDDO
866 2 RETURN VALUE OF COUNT
867 1 END XRUNK
FORTRAN CALLING PROCEDURE
CALL XRISP (CHSTR, LNGTH)
*********
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 LNGTH WORDS IN A2 FORMAT
LNGTH - LENGTH OF CHSTR IN WORDS
*********
OUTPUT
CHSTR - CHARACTER STRING OF LNGTH WORDS IN A2 FORMAT WITH
DUPLICATE BLANKS REMOVED
LNGTH - NEW LENGTH OF CHSTR IN WORDS
*********
EXTERNAL REFERENCES
ENTR
*********
SPECIAL REMARKS
THIS ROUTINE ASSUMES
1. THE INPUT CHARACTER STRING HAS A POSITIVE LENGTH > 0
2. SPECIAL HANDLING OF STRINGS WITHIN QUOTATION MARKS DOES NOT
BEGIN UNTIL THE FIRST OCCURRENCE OF QUOTATION MARKS ON EACH
ENTRY
3. THE FIRST/LAST CHARACTER IN A CHARACTER STRING HAS THE
HIGH ORDER BIT OF THE QUOTE CHARACTER 'ON' TO INDICATE
BEGIN/END OF A CHARACTER STRING. THIS BIT IS TURMED 'OFF'
BY THIS PROGRAM PRIOR TO EXIT. THIS CHANGE MADE TO
INCORPORATE QUOTE MARKS W/I CHARACTER STRINGS.
OTHER PROCESSES CHANGED FOR THIS MODIFICATION WERE:
A. XILAN,KPRDM
B. XILSS/XILSE
*********
FORTRAN CALLING PROCEDURE

CALL ISPRT

GSPRTN BUILDS SEQUENCE TABLE EDITOR PROMPTS.

INPUT

COMMON IX - LU

COMMON IX - DEBUG, NUMENT, PRNTMD, TABMDX, WKBUF

OUTPUT

COMMON IX - PRMLEN, PROMPT, SERNO, TABMDX

USES Routines

XRI6
XRMOV
XRM6
XRM7
XRSET
XUDCG
156 1 CD0    FORTRAN CALLING PROCEDURE
157 1 CD0    CALL XSNPT
158 1 CD0
159 1 CD0
160 1 CD0
161 1 CD0
162 1 CD0
163 1 CD1    XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE
164 1 CD1    TABLE EDITOR
165 1 CD1
166 1 CD2
167 1 CD2
168 1 CD2    INPUT
169 1 CD2
170 1 CD2    COMMON XE - COMBUF, COMPTR, LU, TOKENS
171 1 CD2
172 1 CD2    COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, P3M3MD
173 1 CD2    SEGNO, TABNDX, WKBUF
174 1 CD2
175 1 CD3
176 1 CD3
177 1 CD3    OUTPUT
178 1 CD3
179 1 CD3    COMMON XE - COMPTR
180 1 CD3
181 1 CD3    COMMON XB - INSERT, IRETC, NUMENT, P3M3MD, SEGNO,
182 1 CD3    TABNDX, TABSZ, WKBUF
183 1 CD3
184 1 CD5
185 1 CD5
186 1 CD5    NOTES
187 1 CD5
188 1 CD5    USES ROUTINES
189 1 CD5
190 1 "CD5    XRMSE
191 1 CD5    XSSDL
192 1 CD5    XSNFT
193 1 CD5    XSLIS
194 1 CD5    XSNUM
195 1 CD5    XSNFT
196 1 CD5    XUDDG
197 1 CD5
198 1 C-------
BEGIN XSNPT

... 

IF PROMPT MODE IS UPDATE, THEN

IF TOKEN INPUT IS AN INTEGER, THEN

ERROR IF INTEGER < 1 :ERROR:

RETAIN INTEGER AS SEQUENCE NO. (SEGNO)

INCREMEN TO NEXT TOKEN

ERROR IF TOKEN IS NOT "#" :ERROR1:

INCREMEN TO NEXT TOKEN

START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED

EXIT IF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SEGNO)

SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY

SET INSERT FLAG TO ZERO INDICATING REPLACEMENT OF ENTRY

EXIT IF SEQUENCE NO. OF ENTRY > SEQUENCE NO. INPUT (SEGNO)

SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY

SET INSERT FLAG TO 1 INDICATING INSERT NECESSARY

ELSE

ERROR IF TOKEN IS NOT A NAME :ERROR1:

START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED

EXIT IF NAME INPUT IS DIRECTIVE

SET INDEX TO DIRECTIVE LIST ENTRY

INC. TO NEXT DIRECTIVE

ENDLOOP

ERROR :ERROR2:

ENDSEARCH

CASE (ISLIST, ISDEL, XSNPT, XSNOM, XSMRG), INDEX

ENDIF

ELSE, PROMPT MODE IS NOT UPDATE

IF TOKEN IS #, THEN

SET PROMPT MODE TO UPDATE

F AN & IS INPUT, THEN

ERROR 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

ENDIF

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

EXIT XSNPT

ERROR1: CALL XRMG - "SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD"
3 :ERROR: CALL XMSG - 'INVALID DIRECTIVE'
258 :ERROR: CALL XMSG - 'INVALID SEQUENCE NUMBER ENTERED'
259 : END XMSPT
260

5-256
ENDIF
323  ERREXIT IF NEXT TOKEN IS NOT ECS :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 XPCK 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  EDFIF
332  IF INSERT FLAG INDICATES INSERT (+1), THEN
333  IF ENTRY ABOVE INDEXED ENTRY (TABNDX) IS MARKED DELETED, THEN
334  SET TABLE ENTRY INDEX (TABNDX) 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  DO FOR ALL TABLE ENTRIES FROM BOTTOM OF TABLE TO TABNDX
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  EDFIF
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 TABNDX
349  IF INSERT FLAG DOES NOT INDICATE ENTRY REPLACEMENT, THEN
350  INCREMENT NUMBER OF ENTRIES (NUMENT) BY 1
351  ENDIF
352  EXIT XSEND
353  :ERROR: CALL XRSN = 'SYNTAX ERROR - MISSING OR EXTRANEEO FIELD'
354  :ERROR: CALL XRSN = '.... IS NOT A VALID PROCESSOR NAME'
355  :ERROR4: CALL XRSN - '.... DOES NOT USE AN INTERFACE TABLE'
356  :ERRORS: CALL XRSN - 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
357  END XSEND
359 2 C0D XSPCK
360 2 CD0 FORTRAN CALLING PROCEDURE
361 2 CD0 CALL XSPCK
362 2 CD0 C*********
363 2 CD1 XSPCK COMPACTS THE WORKING BUFFER BY REMOVING ALL SEQUENCE
364 2 CD1 TABLE ENTRIES MARKED FOR DELETION
365 2 CD1 C*********
366 2 CD2 INPUT
367 2 CD2 COMMON IE - LU
368 2 CD2 COMMON XB - DEBUG, NUMENT, TABNOD, WBUF
369 2 CD2 C*********
370 2 CD3 OUTPUT
371 2 CD3 COMMON XB - NUMENT, TABNOD, WBUF
372 2 CD3 C*********
373 2 CD5 NOTES
374 2 CD5 USES Routines
375 2 CD5 XRMN
376 2 CD5 XRMSG
377 2 CD5 XUDOS
378 2 CD5 C*********
BEGIN XSPCK
IF THE TABLE IS NOT EMPTY, THEN
DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
IF THIS ENTRY IS MARKED DELETED, THEN
SET MOVE LENGTH (MOVLEN) TO 7
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT MOVLEN BY 7
ENDIF
ENDO
MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
THE DELETED ENTRY
DECREMENT NUMENT BY MOVLEN/7
IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
ENDIF
ENDIF
ENDDO
END XSPCK
414 2  CD0     FORTRAN CALLING PROCEDURE
415 2  CD0
416 2  CD0
417 2  CD0   CALL XSLST
418 2  CD0
419 2  CD0   C**********
420 2  CD1
421 2  CD1   XSLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
422 2  CD1   CONTAINED IN THE WORKING BUFFER
423 2  CD1
424 2  CD1   C**********
425 2  CD2
426 2  CD2   INPUT
427 2  CD2
428 2  CD2   COMMON XE - LU, RERBUF, SUBSTA
429 2  CD2
430 2  CD2   COMMON XB - Bego, Debug, Endno, Listlu,
431 2  CD2
432 2  CD2
433 2  CD2   C**********
434 2  CD5
435 2  CD5
436 2  CD5     NOTES
437 2  CD5
438 2  CD5
439 2  CD5     USES ROUTINES
440 2  CD5
441 2  CD5
442 2  CD5     XRIG
443 2  CD5
444 2  CD5   C**********
445 3  BEGIN XSLST
446 4  IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
447 4  DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
448 4  SET LIMITS (BEGNO AND ENDNO) OF SEQ. ENTRIES LISTED
449 4  SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
450 3  ENDOF
451 3  WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
452 3  IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
453 4  DO FROM BEGNO TO ENDNO
454 5  MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
455 5  IF INT. TABLE NAME = 0, THEN
456 6  SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARs.)
457 5  ELSE
458 6  SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARs.)
459 5  ENDOF
460 5  CALL XRIG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
461 5  WRITE PRINT BUFFER
462 4  ENDOF
463 3  ENDD0
464 2  END XSLST
466 2 CD0 FORTRAN CALLING PROCEDURE
467 2 CD0
468 2 CD0 CALL XSCAN
469 2 CD0
470 2 CD0
471 2 CD0********
472 2 CD1 XSCAN PERFORMS SYNTACTICAL PROCESSING FOR THE LIST AND
473 2 CD1 DELETE DIRECTIVES OF THE SEQUENCE TABLE EDITOR
474 2 CD1
475 2 CD1
476 2 CD2********
477 2 CD2 INPUT
478 2 CD2
479 2 CD2
480 2 CD2 COMMON XE - COMBUF, COMPTR, LU, TOKENS
481 2 CD2
482 2 CD2 COMMON XB - DEBUG, NUMENT, TABSIZ, WRBUF
483 2 CD2
484 2 CD3********
485 2 CD3 OUTPUT
486 2 CD3
487 2 CD3 COMMON XB - BEGNO, ENDO, IRET
488 2 CD3
489 2 CD3********
490 2 CD5 NOTES
491 2 CD5
492 2 CD5 USES ROUTINES
493 2 CD5
494 2 CD5
495 2 CD5
496 2 CD5 XRMIG
497 2 CD5 XDBG
498 2 CD5
499 2 CD5********
BEGIN XSCAN

IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN

IF NEXT TOKEN IS AN INTEGER, THEN
ERREXIT IF VALUE IS < 1 :ERR06:
SET BEGIN LIMIT (BEGIN) TO THIS VALUE
INCREMENT TO NEXT TOKEN

IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERREXIT IF VALUE IS < 1 :ERR06:
SET END LIMIT (END) TO THIS VALUE
INCREMENT TO NEXT TOKEN

ENDIF

IF BEGIN LIMIT (BEGIN) = 0, THEN
SET BEGIN LIMIT (BEGIN) = 1 (BEGIN IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXITIF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGIN)
SET BEGIN LIMIT (BEGIN) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDDO

END SEARCH

IF END LIMIT (END) = 0, THEN
SET END LIMIT (END) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGIN) TO LAST TABLE ENTRY
EXITIF SEQ. NO. OF THIS ENTRY = END LIMIT (END)
SET END LIMIT (END) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDDO

ERREXIT :ERR08:
END SEARCH

SET RETURN CODE TO INDICATE NO ERROR
SET RETURN CODE TO INDICATE AN ERROR
EXIT XSL1E

:ERR01: CALL XRMSE - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERR06: CALL XRMSE - 'INVALID SEQUENCE NUMBER'
:ERR08: CALL XRMSE - 'INVALID SEQUENCE NUMBER RANGE'

END XSCAN
FORTRAN CALLING PROCEDURE

CALL XSPMT

XSPMT PROCESSES THE SEQUENCE TABLE EDITOR PROMPT DIRECTIVE

INPUT

COMMON AE - CONBUF, COMPTR, LU, TOKENS

COMMON XB - DEBUG

OUTPUT

COMMON XB - PRMTDE, TABNOX

NOTES

USES Routines

BEGIN XSPMT

ERREXIT IF TOKEN IS NOT COMMA :ERROR1:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT A NAME :ERROR1:

increment to next token

ERREXIT IF TOKEN IS NOT EOS :ERROR1:

IF NAME IS 'M', THEN

SET PROMPT MODE TO CREATE

ELSE

ERREXIT IF NAME IS NOT 'A' :ERROR9:

INCREMENT TO NEXT TOKEN

SET PROMPT MODE TO ALL

SET TABLE ENTRY INDEX (TABNOX) TO 0

ENDIF

EXIT XSPMT

:ERROR1: CALL XRMG - 'SYNTAX ERROR - MISSING CK EXTRANEUS FIELD'

:ERROR9: CALL XRMG - 'SYNTAX ERRCK - INVALID QUALIFIER'

END XSPMT
CD0  FORTRAN CALLING PROCEDURE
CD0  CALL XSLIS
661  ***********
662  CD1  XSLIS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR
664  ***********
667  CD2  INPUT
668  CD2  COMMON XE - LU
671  CD2  COMMON XB - DEBUG, TEXT
673  ***********
674  CD5  NOTES
675  CD5  USES ROUTINES
677  CD5  XICAN
679  CD5  XSLST
682  CD5  XUDBG
683  ***********
685  CD0  FORTRAN CALLING PROCEDURE
687  CD0  CALL X$NUM
689  CD0  ***********
691  CD1  X$NUM IS THE SEQUENCE TABLE EDITOR NUMBER DIRECTIVE PROCESSOR
693  ***********
695  CD2  INPUT
697  CD2  COMMON XE - COMBUF, COMPTR, LU, TOKENS
699  CD2  COMMON XB - DEBUG, NUMBER
701  CD2  ***********
703  CD3  OUTPUT
705  CD3  COMMON XB - WKBUF
707  CD3  ***********
709  CD5  NOTES
710  CD5  USES ROUTINES
712  CD5  X$MMMS
713  CD5  XUDBG
715  CD5  XUDBG
CD5
CD5
BEGIN XSNUM
ERREXIT IF TOKEN IS NOT EOS:ERROR1:
IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
SET SEQUENCE NUMBER (SENO) TO 100
DO FOR ALL ENTRIES IN TABLE
IF TABLE ENTRY IS NOT MARKED DELETED, THEN
SET SEQ. NO. FIELD OF ENTRY TO SEQUENCE NUMBER (SENO)
INCREMENT SEQUENCE NUMBER (SENO) BY 100
ENDIF
ENDIF
ENDDO
EXIT XSNUM
:ERROR1: CALL XRMESG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
END XSNUM
**FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:**

CALL XTCOM (PROMPT, PRMLEN, RETCOD)

**INPUTS FROM CALLING SEQUENCE:**

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

**OUTPUTS IN CALLING SEQUENCE:**

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

**INTERNAL VARIABLES**

- CONMSG - 6 WORD ARRAY CONTAINING THE CONTINUATION MESSAGE
- PREFIX - 4 CHARACTERS USED AS PREFIXES TO PROMPT, CORRESPONDING TO CODES IN XE(5)
- RETCEP - RETURN CODE FROM XIPRM EXTENDED PROMPTING
- RETCLA - RETURN CODE FROM XITLAN LEXICAL ANALYSIS
- SUFFIX - 5 CHARACTERS APPENDED TO END OF PROMPT

**XE COMMON USED:**

- EQUIVALENCE (XE(1), LU), (XE(2), ILASS)
- (XE(5), LFVL), (XEFF(115), LIMNIP)
- (XE(143), NOTOKS), (XE(146), NODIS)

**XS COMMON USED:**
61 1 CDS
   EQUIVALENCE (XS(1), INBUF), (XS(81), NOIN),
      (XS(82), OUTBUF), (XS(122), SSFLAG),
      (XS(1123), INLEM)
62 1 CDS
63 1 CDS
64 1 CDS
65 1 CDS
66 1 CDS
67 1 CDS
68 1 CDS
69 1 CDS
70 1 CDS
71 1 CDS
72 1 CDS
73 1 CDS
74 1 CDS
75 1 CDS
76 1 CDS
77 1 CDS
78 1 CDS
79 1 CDS
80 1 CDS
81 1 CDS
82 1 CDS
83 1 CDS
84 1 CDS
85 1 CDS

INBUF - 80 WORD ARRAY THAT CONTAINS THE USER'S RESPONSE
IN R1 FORMAT
IMLEN - LENGTH OF ORIGINAL PROMPT PASSED TO XTPRM
NOIN - NUMBER OF WORDS OF INBUF ACTUALLY USED FOR THE
USER'S RESPONSE
OUTBUF - 40 WORD AREA WHERE PROMPT IS CONSTRUCTED AND
WRITTEN FROM.
SSFLAG - SYMBOLIC STRING INDICATOR FLAG
  0 = NOT WITHIN A SYMBOLIC STRING
  NONZERO = VALUE REPRESENTING THE INDEX INTO COMBUF
WHERE LENGTH OF SS IS TO BE STORED

SUBROUTINES AND FUNCTIONS CALLED:
      EXEC, KCVT, XMOV, XMSG, XNUMP, XTPRM

PDL ROUTINES INCLUDED:
      XTCOM, READSEG
BEGIN XTCOM
IF PROMPT IS NOT TOO LONG (76 CHAR) THEN
MOVE PREFIX CHARACTER FOR EXEC LEVEL INTO OUTPUT AREA
MOVE PROMPT INTO OUTPUT AREA
MOVE SUFFIX CHARACTER INTO OUTPUT AREA
:LOOP:
ISSUE WRITE TO PROMPT USER
INITIALIZE COMMUNICATIONS BUFFER
TURN SYMBOLIC STRING FLAG OFF
INITIALIZE LA RETURN CODE TO NORMAL RETURN
PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
DO WHILE LEXICAL ANALYSIS (LA) RETURN CODE SAYS CONTINUE AND
(exec level is not interface table editor or
SYMBOLIC STRING FLAG IS ON)
CALL EXEC TO WRITE CONTINUATION MESSAGE
PERFORM READSEG TO READ INPUT AND CALL LEXICAL ANALYSIS
ENDDO
IF LA RETURN CODE SAYS ERROR IN RESPONSE THEN
CALL KCVT TO CONVERT OCTAL TO ASCII
CALL XRMG 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 XRMG 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 XRMG TO WRITE ERROR MESSAGE
ENDIF
GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF
IF LA RETURN CODE SAYS COMBUT IS FULL THEN
CALL XRMG 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
1 END XTCOM
131 1 BEGIN READSEG
132 2 CALL XRM0V TO INITIALIZE INPUT BUFFER TO BLANKS
133 3 CALL XREC TO READ RESPONSE TO PROMPT
134 4 CALL XRPK ROUTINE TO CONVERT A2 FORMAT BUFFER TO R1
135 5 IF NUMBER OF WORDS READ IS NOT ZERO THEN
136 6 CALL XTLM ROUTINE TO BUILD COMMUNICATIONS BUFFER
137 7 ELSE
138 8 IF LAST LA RETURN CODE WAS A CONTINUE THEN
139 9 REMOVE TRAILING COMMAS FROM COMBUF
140 0 SET LA RETURN CODE TO NORMAL RETURN
141 1 ELSE
142 2 SET LA RETURN CODE TO SAY USER ENTERED CR
143 3 ENDIF
144 4 ENDIF
145 5 END READSEG
CD************
CD CALLING PROCEDURE FOR LEXICAL ANALYSIS:
C
CALL XTLAM (RETC)
CD************
CD CONVERT 'INBUF' USER'S RESPONSE TO 'COMBUF' OF TOKENS
CD INDICATING CHARACTERS, INTEGERS, REALS, ETC.
CD************
CD OUTPUTS IN CALLING SEQUENCE:
CD RETC - (INTEGER, 1 WORD) IS A COMPLETION CODE PASSED
CD BACK TO CALLER AS FOLLOWS:
CD 0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE.
CD 1 - USER RESPONDED TO BUFFER CONTAINS RESPONSE
CD UP TO AND INCLUDING X.
CD 5 - USER REQUESTED A CONTINUATION.
CD 6 - EXTENDED PROMPTING REQUEST WAS RECEIVED. BUFFER
CD CONTAINS RESPONSE UP TO AND INCLUDING THE REQUEST.
CD 7 - COMMUNICATIONS BUFFER IS FULL.
CD 1XX - ERROR IN RESPONSE AT OR BEFORE CHARACTER XX.
CD 2XX - OVERFLOW/UNDERFLOW DETECTED AT OR BEFORE
CD CHARACTER XX.
CD************
CD INTERNAL VARIABLES
CD COMLEN - LENGTH IN WORDS OF COMBUF =256
CD DBLINT - DOUBLE PRECISION LOCATION TO ACCUMULATE AN
CD INTEGER VALUE
CD DBLWD - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER
CD AND FRACTIONAL VALUE FOR DOUBLE PRECISION OR REM.
CD FLGCOM - COMMA FLAG
CD 0 - LAST CHARACTER WAS NOT A COMMA
CD 1 - LAST CHARACTER WAS A COMMA
CD FLGEND - END LOOP FLAG
CD 0 - CONTINUE LOOP
CD 1 - END LOOP
CD FLGNEG - NEGATIVE EXPONENT FLAG
CD 0 - EXPONENT WAS POSITIVE
CD 1 - EXPONENT WAS NEGATIVE
CD FLGTP - TYPE OF REAL VALUE
CD 0 - SINGLE PRECISION
CD 1 - DOUBLE PRECISION
CD POWER - EXPONENT PART OF A REAL NUMBER
CD RELWD - SINGLE PRECISION LOCATION FOR REAL VALUE
CD SCCHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING
CD THE TI FORMAT REPRESENTATION FOR:
CD "?="?/0#?="?/)?"#W.XXR.; DEW
CD X IS AN EXCLAMATION POINT
CD Y IS AN OPEN BRACKET
CD Z IS A BACK SLASH
CD
XE COMMON USED:

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

XS COMMON USED

EQUIVALENCE (XS(1), INBUF), (XS(81), NCHAR),
+ (XS(122), FLAGSS), (XS(186), SCRATCH)

SUBROUTINES AND FUNCTIONS CALLED

XRPCX, XRMV

PDL ROUTINES INCLUDED:

XTLM, COMMA, ALPHA, DIGIT, DCOL, DECPT, EORD
INTGRA, REAL, DBL, REPET, INVAL, SCHARS, QUOTE
BEGIN XLAN
   SET END FLAG OFF
   IF LAST CHARACTER WAS A COMMA ON
      INITIALIZE INDEX INTO INPUT BUFFER
      INITIALIZE RETURN CODE TO NORMAL RETURN
   SET RETURN CODE TO SAY CONTINUATION REQUESTED
   ENDIF
   DO WHILE END FLAG IS OFF
      IF INPUT BUFFER HAS BEEN COMPLETELY SCRANNED THEN
         IF LAST CHARACTER WAS A COMMA OR A SYMBOLIC STRING IS STILL OPEN THEN
            EXIT 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 DIG
               ELSE
                  PERFORM SCHARS
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDDO
   STORE END OF BUFFER TOKEN IN COMBUF
   INCREMENT #TOKENS BY 1
END XLAN

BEGIN COMMA
   SET LAST CHARACTER WAS A COMMA ON
   EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
   STORE COMMA TOKEN IN COMBUF
   INCREMENT #WORDS IN COMBUF BY 1
   INCREMENT #TOKENS BY 1
   GET NEXT INPUT CHARACTER
END COMMA
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 \times 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 END DCOL
12
13 BEGIN DECP
14   CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
15   GET NEXT INPUT CHARACTER
16   IF INPUT BUFFER IS NOT EXHAUSTED THEN
17     IF INPUT CHARACTER IS A DIGIT THEN
18       PERFORM DCOL
19     ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
20     EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
21     ENDIF
22     IF INPUT CHARACTER IS AN "E" OR A "D" THEN
23       PERFORM EORD
24     ELSE
25       PERFORM REAL
26     ENDIF
27     ELSE
28       PERFORM REAL
29   ELSE
30     ENDIF
31 END DECP
BEGIN EORD
IF INPUT CHARACTER IS AN "E" THEN
SET TYPE FLAG TO "E"
ELSE
SET TYPE FLAG TO "D"
ENDIF
GET NEXT CHARACTER
ERREXIT IF INPUT BUFFER IS EXHAUSTED
SET NEGATIVE FLAG OFF
IF CHARACTER IS A - THEN
SET NEGATIVE FLAG ON
GET NEXT CHARACTER
ELSE
IF CHARACTER IS A + THEN
GET NEXT CHARACTER
ENDIF
ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
ERREXIT IF CHARACTER IS NOT A DIGIT PERFORM IVALID.
PERFORM BCOL
IF NEGATIVE FLAG IS ON THEN
SET POWER = -POWER
ENDIF
IF TYPE FLAG IS "E" THEN
PERFORM REAL
ELSE
PERFORM DBL
ENDIF
1 END EORD
1 BEGIN INTEGER
2  ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONSUM
3  CONVERT NUMBER TO INTEGER
4  ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
5  STORE INTEGER TOKEN IN COMBUF
6  INCREMENT WORDS IN COMBUF BY 2
7  INCREMENT TOKENS BY 1
8  END INTEGER
9
10 1 BEGIN REAL
11 2  ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONSUM
12 2  SET REAL = DOUBLE PRECISION * 10 ** POWER
13 2  ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
14 2  STORE REAL TOKEN IN COMBUF
15 2  INCREMENT WORDS IN COMBUF BY 3
16 2  INCREMENT TOKENS BY 1
17 2  END REAL
18
19 1 BEGIN DBC
20 2  ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONSUM
21 2  SET DOUBLE = DOUBLE PRECISION * 10 ** POWER
22 2  ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
23 2  STORE DOUBLE TOKEN IN COMBUF
24 2  INCREMENT WORDS IN COMBUF BY 4
25 2  INCREMENT TOKENS BY 1
26 2  END DBC
27
28 1 BEGIN REPET
29 2  ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONSUM
30 2  ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM IZVAL
31 2  STORE REPEAT TOKEN IN COMBUF
32 2  INCREMENT WORDS IN COMBUF BY 2
33 2  INCREMENT TOKENS BY 1
34 2  GET NEXT CHARACTER
35 2  END REPET
1 BEGIN SCHARS
  2  *  CHARACTER TABLE:
  3  "-#<>09=?:()'*%¥'; Z IS A CLOSED BRACKET
  4  X IS AN OPEN BRACKET
  5  Z IS A BACK SLASH
  6  SET J=1
  7  STARTSEARCH WHILE J<=#CHARACTERS IN TABLE
  8  EXIT IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
  9  INCREMENT J BY 1
 10  ENDCASE
 11  IF SYMBOLIC STRING FLAG IS OFF THEN
 12     SET SYMBOLIC STRING FLAG TO CURRENT COMBUF INDEX + 1
 13  ELSE
 14     SET COMBUF(SYMBOLIC STRING FLAG) = CURRENT COMBUF INDEX - SYMBOLIC STRING FLAG
 15   ENDIF
 16   IF FOLLOWING CHARACTER IS A DIGIT THEN
 17     SET NEXT FLAG OFF
 18     SET INTEGER = 0
 19     SET POWER = 0
 20     PERFORM DECPT
 21   ENDIF
 22   IF NEXT FLAG IS ON
 23     STORE TOKEN (J) IN COMBUF
 24   ENDIF
 25   INCREMENT #WORDS IN COMBUF BY 1
 26   IF TOKEN IS BEGIN SYMBOLIC STRING THEN
 27     INCREMENT #WORDS IN COMBUF BY 1
 28   ENDIF
 29   INCREMENT #TOKENS BY 1
 30   GET NEXT CHARACTER
477 1 BEGIN QUOTE
478 2 GET NEXT CHARACTER
479 2 SET #CHARACTERS = 0
480 2 DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
481 3 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
482 3 (INPUT CHARACTER IS A QUOTE AND
483 3 NEXT CHARACTER IS A QUOTE AND
484 3 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
485 3 INCREMENT #CHARACTERS BY 1
486 3 MOVE CHARACTER INTO TEMPORARY BUFFER
487 4 IF INPUT CHARACTER IS A QUOTE THEN
488 5 GET NEXT CHARACTER
489 3 ENDIF
490 3 GET NEXT CHARACTER
491 2 ENDDO
492 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
493 2 ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
494 2 ERREXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
495 2 STORE CHARACTER STRING TOKEN IN COMBUF
496 2 STORE #CHARACTERS IN COMBUF
497 2 CALL XRPROC TO PACK CHARACTERS INTO COMBUF
498 2 INCREMENT #WORDS IN COMBUF BY 2 + (#CHARACTERS +1)/2
499 2 INCREMENT #TOKENS BY 1
500 2 GET NEXT CHARACTER
501 1 END QUOTE
FORTRAN CALLING PROCEDURE

CALL XTPRM

****

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

****

INPUT

XX COMMON - CARTRG, COMBUF, FLAGS, L4, NOPROC, PRCMK, SUBSTA,

XX COMMON - AREPTR, UKBUF (FROM INTERFACE TABLE EDITOR)

XX COMMON - PRLMLE (FROM XTCOM)

VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)

OUTPUT

XX COMMON - COMBUF (USED FOR SCRATCH SPACE)

LISTING OF APPROPRIATE EXTENDED PROMPTS

****

INTERNAL VARIABLES

CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY

FILE - FILE NAME OF CURRENT TABLE ENTRY

INDEX - INDEX TO CURRENT TABLE ENTRY

L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO

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

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

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

AS FOLLOWS

ENTRY SYNTAX CONDITION FILE REC SIZE SEARCH MESSAGE CONTINUE

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

2 I R?: 1 9 I >XPRM 1 2 YES XTO6 NO I

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

4 I ,?: , $?: , $?: 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?, $?: , $?: , $?: 1 41 I >PPDC 3 128 NO XTO7 NO I

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

8 I ??: 1 57 I >PROC 1 3 NO NONE I

9 I /?: 1 65 I >XPRM 1 2 YES NONE NONE I

10 I P?: , $?: 1 73 I >PROC 1 3 YES XTO8 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 POSITION TO INDICATED STARTING RECORD AND READ
6 EXIT TO :ERROR0: IF FAILURE
7 IF TABLE SIZE FIELD < 128 (NOT A LIST RECORD), THEN
8 IF SIZE > 0 (NO LAST CHARACTER PADDING & POSSIBLE SPANNING), THEN
9 READ SECOND RECORD AND APPEND TO FIRST RECORD DATA
10 EXIT TO :ERROR0: IF FAILURE
11 ELSE
12 SET SIZE POSITIVE
13 DO FOR EACH LIST ITEM (1-M)
14 BLANK LAST CHARACTER
15 ENDDO
16 ENDIF
17 IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), THEN
18 START SEARCH WHILE LIST ITEMS REMAIN TO BE EXAMINED
19 EXIT IF TOKEN LOCATED IN LIST
20 POSITION TO APPROPRIATE RECORD (I+L-1) AND READ
21 EXIT TO :ERROR0: IF FAILURE
22 SET SIZE TO 128 (ENTIRE RECORD TO BE DISPLAYED)
23 IF TABLE MESSAGE NUMBER FIELD > 0, THEN
24 CALL XRMSG TO DISPLAY 'NOT VALID ...' MESSAGE
25 ENDIF
26 EXIT TO ENDDO
27 END SEARCH:
28 ENDIF
29 PERFORM DISPLAY
30 ELSE OPEN ERROR
31 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
32 CALL XRMSG TO DISPLAY 'NOT VALID ...' MESSAGE
33 ELSE
34 :ERROR0: CALL XRMSG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
35 CLOSE FILE
36 ENDIF
37 ENDDO
38 1 END XTPRM
1 BEGIN SETUP
2 CASE (/:, :@, :$, /:) SUBSTA
3 :Z:
4 IF FIRST TOKEN = ?, THEN
5 SET TABLE INDEX TO FIRST ENTRY
6 ELSE
7 SET TABLE INDEX TO SECOND ENTRY
8 ENDIF
9 :S:
10 IF FIRST TOKEN = ?, THEN
11 SET TABLE INDEX TO FOURTH ENTRY
12 ELSE
13 SET TABLE INDEX TO SIXTH ENTRY
14 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
15 ENDIF
16 :F:
17 IF PROMPT LENGTH = 0, THEN
18 ASSUME TABLE INDEX OF THIRD ENTRY
19 ELSE
20 ASSUME TABLE INDEX OF FOURTH ENTRY
21 ENDIF
22 IF FIRST TOKEN IS A NAME, THEN
23 IF PROMPT LENGTH IS A NAME, THEN
24 IF PROMPT LENGTH = 0, THEN
25 CHANGE TABLE INDEX BY TWO ENTRIES (NOW FIFTH OR SIXTH)
26 FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
27 ENDIF
28 IF PROMPT LENGTH = 0, THEN
29 CHANGE TABLE INDEX TO SEVENTH ENTRY
30 ELSE
31 CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY IT EDITOR
32 ENDIF
33 ELSE
34 IF PROMPT LENGTH = 0, THEN
35 CHANGE TABLE INDEX TO NINTH ENTRY
36 ENDIF
37 END CASE
38 1 EXIT SETUP
39 :ERROR2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
40 1 END SETUP
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
FORTRAN CALLING PROCEDURE

CALL XUDBG (I,U, ID)

XUDBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY
DUMP AND/OR MODIFICATION FACILITY

INPUT (CALLING SEQUENCE)

LU - LOGICAL UNIT TO INTERACT WITH IN INVOKING XUDBG OPTIONS
ID - THREE WORD ASCII ARRAY USED AS A HEADER TO IDENTIFY XUDBG CALLER

INPUT (INTERACTIVE)

OPTION (S,M,E): - ONE ASCII CHARACTER IDENTIFYING XUDBG OPTION
S - SNAP OUT (DUMP) MEMORY
M - MODIFY MEMORY
E - EXIT XUDBG
START: - OCTAL MEMORY ADDRESS IN THE 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 - "*** XUDBG FROM IDIDID"
PROMPTS - (SEE INPUT)

OUTPUT (TO 'OUTPUT UNIT')

HEADER - "*** XUDBG FROM IDIDID"
SNAP - 102 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 WORD BUFFER OF WORD TO BE SNAPPED
LUI - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET
LU - LOGICAL UNIT FOR SNAP OUTPUT
OPTN - 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
STRT - 'ORIGIN' RELATIVE BEGINNING OF MEMORY TO BE ACCESSED
RH - RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND CONVERSION'
NOTES

ORIGIN OF THE REPRODUCIBILITY OF THE ORIGINAL PAGE IS PRET
BEGIN XUG
WRITE SNAP HEADER TO TERMINAL
DO UNTIL OPTION IS EXIT (E)
PROMPT TERMINAL FOR OPTION
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 TO OUTPUT UNIT
WRITE SNAP HEADER TO OUTPUT UNIT
DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
CALL XUDP TO FORMAT LINE
OUTPUT LINE
END DO
ENDIF
END
END XUG
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 FM FILE TO BE DUMPED
IREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED
FMT - RUN TIME FORMAT FOR RECORDS (MAXIMUM OF 72 CHARACTERS) OR BLANK INDICATING THE DEFAULT OF OCTAL AND ASCII DUMP TYPE
LU - LOGICAL UNIT NUMBER OF OUTPUT DEVICE

OUTPUT
FORMATTED DUMP OF THE INDICATE PORTION OF THE INDICATED FILE

NOTES
USES EXEC, HAXO, OPEN, POINT, READF, ENUMF, XPRBS, XPRNO, XUPLF
ANY FILE WITH VARIABLE LENGTH RECORDS WILL BE DUMPED USING A RECORD BUFFER OF 1024 WORDS. LIMITING THE MAXIMUM DUMPABLE

BEGIN XUDDF
DO FOREVER
READ FILE NAME
EXIT XUDDF 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
ENDIF
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
END XUDDF
FORTRAN CALLING PROCEDURE

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)

1  C00  CALL XUDPL (ADDRES, LINE, BUFFER)
FORTRAN MAIN PROGRAM XUFMT IS SCHEDULED BY XUOMP TO PRINT

A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK

INPUT
ICTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
NUMBER OF WHERE DUMP IS ON DISK

TRACK - TRACK NUMBER OF A & TRACK GLOBALLY ALLOCATED AREA

CONTAINING THE DUMP

OUTPUT
FORMATTED DUMP TO LU 6

EXTERNAL REFERENCES
EXEC
RMPAR
XRCPR
XRMOV
XUDPL

BEGIN XUFMT
CALL RMPAR TO GET LU AND STARTING TRACK NOS.
READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
.COUNT OF ID-SEGS TO BE DUMPED
.UP TO 7 ID-SEGMENT ADDRESSES
.LOW AND HIGH BASE PAGE ADDRESSES
.LOW AND HIGH MAIN MEMORY ADDRESSES
.POINT TO 2ND SECTOR OF DUMP DATA
.DO UNTIL ALL ID-SEGS PRINTED
.DO UNTIL A EIGHT-WORD LINES PRINTED
.PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
.ENDD
_INCREMENT TO NEXT SECTOR OF DUMP DATA
.ENDDO

BEGIN XUDPL
.COUNT N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
.PERFORM COMPARE AND PRINT FUNCTION
.COMPUTE NTRKS, NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ
.DO UNTIL NTRKS ARE READ
.READ NEXT TRACK FROM DISK
.COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP
.PERFORM COMPARE AND PRINT FUNCTION
.ENDDO
.RELEASE THE GLOBALLY ALLOCATED TRACKS

EXIT XUFMT

BEGIN COMPARE AND PRINT FUNCTION
.DO UNTIL N LINES PROCESSED
IF NOT 1ST LINE, THEN
.CALL XRCPR TO COMPARE WITH PREVIOUS LINE
IF LINES ARE IDENTICAL, THEN
IF THIS IS 1ST OF A SERIES, THEN
WRITE 'DUPLICATE LINE'
.ENDIF
ELSE
.CALL XUDPL TO FORMAT THE DUMP LINE
.WRITE FORMATTED DUMP LINE
.ENDIF
ELSE
.CALL XUDPL TO FORMAT THE DUMP LINE
.WRITE FORMATTED DUMP LINE
.ENDIF
.ENDDO
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1  ENTRY XVABN
2  TYPE 1A ROUTINE TO ABORT CURRENT ID AND
3  REQUEST A DUMP FOR THE ID.
4  ENTRY XVABN
5  TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
6  THE REQUESTED DATA AREAS AND THE CURRENT
7  ID SEGMENT AND ITS FATHER CHAIN SEGMENTS
8  * 
9  BEGIN XVABN ABEND
10  * 02 INPUTS: NONE
11  * 03 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
12  * EXTERNALS: $ABRT,$ABRE,$XEN,$LIBR
13  CALL SLIBR  REENTRANT ROUTINE CALL
14  GET CURRENT ID FROM XBT (LOCATION 1717)
15  INCREMENT TO ADDRESS BOUNDARIES
16  STORE := XUDMP PARAMETER LIST
17  PERFORM XUDMP  DUMP THIS PARTITION
18  CALL $ABRT  ABORT THIS PARTITION
19  CALL $ABRE  FREE CURRENT REENTRANT DATA BLOCK
20  * 
21  EXIT TO $XEN  GO TO DISPATCHER
22  END XVABN
23  * 
24  * 
25  * 
26  * 
27  * 
28  * 
29  * 
30  BEGIN XUDMP
31  ENTRY XUDMP
32  * 02 INPUTS: ADDRESS OF DOUBLE WORD CONTAINING START AND END ADDRESSES
33  * 02 DEFINING AREA TO BE DUMPED
34  * 03 OUTPUTS: OUTPUTS TO DISK
35  * 03 HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW
36  * 03 CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS
37  * 03 MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION
38  * 03 MEMORY IMAGE AS SPECIFIED BY INPUT ADDRESS RANGE
39  * 03 SCHEDULES XFMT FOR FORMAT DATA FROM DISK TO PRINT
40  * 03 EXTERNALS: $LIBR, $LIBX, EXEC, XFMT
41  CALL SLIBR TO NOTIFY OF REENTRANCY AND BECOME PRIVILEGED
42  RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH
43  CALL SLIBX TO BECOME NON-PRIVILEGED
44  CALL EXEC FOR A GLOBAL ALLOCATION OF DISK
45  CALL EXEC TO WRITE HEADER DATA USING CLASS 1/0
46  DO UNTIL ALL ID SEGMENTS PROCESSED
47  WRITE ID SEGMENT TO DISK
48  DO ENDDO
49  WRITE BASE PAGE SEGMENT TO REMAINING PROTION OF THIS TRACK
50  DO UNTIL END ADDRESS REACHED
51  WRITE ONE SECTOR OF DATA
52  ENDDO
53  SCHEDULE XFMT WITH LU AND STARTING TRACK NO.
54  CALL SLIBX TO RETURN FROM REENTRANT ROUTINE
55  END XUDMP
58 1 D01 TYPE 14 ROUTINE TO CONTROL COMMUNICATION BETWEEN AWD
59 1 EXECUTION OF FDS MANAGER AND IT'S ASSOCIATED TASKS
60 1 (EXECUTIVE, PROCESSORS, AND UTILITIES).
61 1 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 (P3-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 ASOCIATE TASK IS ACTIVATED VIA SLIST
103   1 BEGIN XVPAM
104   2 CALL BLIBN  BECOME PRIVILEGED
105   2 SET STOP-ID FROM XERT(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   3 ENDF
114   2 * STOP-ID HAS ID-SEGMENT TO BE SUSPENDED
115   2 * AWAKE-ID HAS ID-SEGMENT TO BE ACTIVATED
116   2 CALL SLIST(SCHEDULE,AWAKE-ID)
117   2 MOVE RETURN ADDRESS TO XSUSP OF STOP-ID.
118   2 CALL SLIST(WAIT,STOP-ID)
119   1 EXIT TO :$XEN RTE DISPATCHER
120   1 END XVPAM
122   1 BEGIN XVPMAN
123   2  * DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
124   2  * FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
125   2  * SET TARGET-ID FROM CURRENT-ID-SEGMENT
126   2  * DO WHILE FATHER-ID-NUMBER .NE. 0 OR FATHER IS WAITING
127   2  * COMPUTE FATHER-ID-SEGMENT FROM FATHER-ID-NUMBER IN TARGET-ID
128   2  * PERFORM MGFRND(FATHER-ID-SEGMENT,COUNT)
129   2  * EXIT IF COUNT .GT. 0
130   2  * SET TARGET-ID TO FATHER-ID-SEGMENT
131   2  * ENDQ
132   2  * IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
133   2  * CALL $SYM (12,*XV03,SEGMENT-NAME) 'xv03,NAME' REQUESTING PROG
134   2  * PERFORM PUMP
135   2  * EXIT TO $LIBX TO ENABLE AND REDISPATCH
136   2  * ENDF
137   2  * SET AWAKEN-ID FROM FSD-ENTRY $SYM
138   2  * GET REQUEST PARTS MOVE INTO ID-SEGMENT
139   2  * SET STBAT FROM CURRENT-ID
140   1  FND XVPMAN
141   1  *
142   1  *
143   1  *
144   1  *
145   1  DEC: XVPMAN
146   2  * DETERMINE IF CALLER IS A VALID FDS MANAGER
147   2  *
148   2  * PERFORM MGFRND (CURRENT-ID,COUNT)
149   2  * COUNT WILL BE 0 FOR NO MATCH.
150   2  * COUNT NOT EQUAL ZERO IMPLIES A MATCH
151   2  * AND FSD-ENTRY HAS MATCHING FDSTAB ENTRY ADDRESS
152   2  * IF COUNT .LT. 0 THEN CALLER IS NOT A FDS MANAGER
153   2  * CALL $SYM (12,*XV01,SEGMENT-NAME) 'xv01 PROGRAM' REQUESTING PROG.
154   2  * PERFORM PUMP
155   2  * EXIT TO $SKEP THE DISPATCHER
156   2  ENDF
157   2  *
158   2  * FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
159   2  * SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
160   2  * IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
161   2  * CALL $SYM (12,*XV02,SEGMENT-NAME) 'xv02 PROGRAM' ASSOCIATED PROG.
162   2  * PERFORM PUMP
163   2  * EXIT TO $SKEP THE DISPATCHER
164   2  ENDF
165   2  *
166   2  * MOVE FDS MANAGERS INPUT PARTS TO ASSOCIATED TASK ID SEGMENT
167   2  MOVE PARTS TO ID-SEGMENT WORDS 2-6
168   1  ENDF XVPMAN
172 1 BEGIN PDUMP
173 2 * SET A CALL TO XYABN AT CALLER'S SUSPEND POINT
174 2 * END PDUMP
176 1 *
177 1 *
178 1 *
179 1 *
180 1 *
181 1 *
182 1 BEGIN MGRFND INPUT IS TARGET-ID
183 2 GET XSTB TABLE OF ACTIVE FDS MANAGERS
184 2 SET COUNT TO NUMBER IN TABLE
185 2 SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
186 2 DO WHILE COUNT .GT. 0
187 3 EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMENT
188 3 SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
189 3 SET COUNT = COUNT - 1
190 2 ENDDO
191 2 * FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
192 2 * COUNT = 0 MEANS NO MATCH
193 1 END MGRFND
194 1 *
195 1 *
196 1 *
197 1 *
198 1 BEGIN XSTB
199 2 N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XSTB
200 2 STBMM DEF N NUMBER-IN-TABLE
201 2 STBKC DEF O NUMBER-ACTIVE
202 2 STBNN DEF O STB RESOURCE NUMBER
203 2 STBES EQU * ENTRY START
204 2 STBLU DEF O LUC(OTAL)
205 2 STBLA DEF O LUC(ASCII)
206 2 STBGM DEF O A(FDS-MANAGER-ID-SEGMENT)
207 2 STBX DEF O A(FDS-EXECUTIVE-ID-SEGMENT)
208 2 STBCE DEF O A(CLASS-NUMBER-FOR-EXEC)
209 2 STBAT DEF O A(CURRENT-ASSOCIATED-TASK)
210 2 STBPC DEF O A(CLASS-NUMBER-FOR-PROCESSOR)
211 2 STBEG DEF O A(ENTRY AREA)
212 2 STBEE EQU * ENTRY "END"
213 2 STBEL EQU STBEE-STBES ENTRY LENGTH
214 2 STBNE EQU N NUMBER-OF-ENTRIES
215 2 ORG STBES
216 2 REP N DO FOR N ENTRIES
217 2 REP STBEL DEFINE STORAGE WORDS FOR AN ENTRY
218 2 DEF 0
219 1 END XSTB
FORTRAN CALLING PROCEDURE:

CALL IXAUT

IXAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE

INPUTS IN COMMON:

XE(5) MASSTA, XE(10) SENSTR, XE(11) SESEND, XE(12) SEQPTR,
XE(140) TABEND, XB(1)NOPROC, XB(2) LIBD,
XB(249) SENNO, XB(250) SENLEM, XB(251) SENTAB

OUTPUTS IN COMMON:

XE(5) MASSTA, XS(1) FLGTAB

COMMON USED:

EQUIVALENCE (XE(5), MASSTA)
+XE(10), SENSTR, XE(11), SESEND
+XE(12), SEQPTR, XE(140), TABEND,
+XB(1), NOPROC, XB(2), LIBD,
+XB(249), SENNO,
+XB(250), SENLEM, XB(251), S E N T A B,
+XS(1), FLGTAB

FDB ROUTINES CALLED:

XECPR, XEXT, XRMOV, XHSG,
XEXEE, XISTO, XITMP

RTE ROUTINES CALLED:

I09
49 1 BEGIN XXAUT
50 2 IF ENTRY IS FROM A DIRECTIVE THEN
51 3 SET MASTER STATE TO INDICATE REENTRY
52 4 DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
53 5 SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
54 6 ERREXIT IF PROCESSOR NOT FOUND TO :ERR1:
55 7 STUFF INTERFACE TABLE 6IT AND VERSION INTO SEQUENCE TABLE ENTRY
56 8 ENDIF
57 9 CALL XXSTO TO STORE REVISED SEQUENCE TABLE IN AN A AS &SERTB
58 10 ELSE (I AM BEING REENTERED FROM INT.)
59 11 CALL XXTRP TO SET UP TEMPORARY EN-ITY W:ITH $INTAB
60 12 CALL XXEXE TO EXECUTE FROM TEMPORARY ENTRY
61 13 IF RESET SEQUENCE NUMBER IS NOT REQUESTED THEN
62 14 EXIT XXAUT IF TERMINAL ENTRY WAS JUST EXECUTED
63 15 SET STARTING ENTRY TO NEXT ENTRY
64 16 ENDIF
65 17 2 EXIT XXAUT
66 18 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
67 19 CALL XXEXE TO EXECUTE REMAINDER OF TABLE
68 20 ENDDO
69 1 EXIT XXAUT
70 2 :ERR1:
71 2 CALL XXMSG TO DISPLAY INVALID PROCESSOR NAME
72 1 END XXAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELBS (XICNT)

XICNT is the main program for the execution controller. It gives control to the appropriate subroutine depending on the mode and returns to directive level.

INPUTS FROM CALLING SEQUENCE:

XICNT - (INTEGER, 3 WORDS) ARRAY CONTAINING THE NAME "XICNT" USED BY XELBS 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), XASSTA)

FBS ROUTINES USED
XERT, XEXT, XRMSC, XHMGR

NOTE: CONTAINS DUMMY CALL TO XEXEC
119 1 BEGIN XXCNT
120 2 SET MODE TO XREXT OF BITS 12 AND 13 OF WASSTA
121 2 CASE MODE (:MANU:, :SENI:, :AULT:, :AUTO:)
122 3 :MANU: CALL XXMAN
123 3 :SENI: CALL XXSEN
124 3 :AULT: CALL XENSG TO DISPLAY INVALID OPTION
125 3 :AUTO: CALL XXAUT
126 2 ENDCASE
127 2 SET MASTER STATE TO DIRECTIVE LEVEL
128 2 CALL XERTH TO RETURN TO XEXEC "AND RETURN"
129 2 DUMMY CALL XEXEC
130 1 END XXCNT
 Fortran Calling Sequence:

CALL XDEC (RETC)

XDEC Decodes a Response of Processor Name (INT Table Name)

into a sequence Table Entry.

Inputs in COMMON:

X85 Tokens, XE4145) Combuf, XB(1) Noproc, XB(2) Libd

Outputs in COMMON:

XE(16) PrCnam, XB(251) Sertab

Common Used:

Equivalent (XE(16), PrCnam),

+ (XE85), Tokels), (XE144), Tokptr1),

+ (XE145), Combuf),

+ (XB(1), Noproc), (XB2), Libd)

+ (XB251), Sertab)

FDS Routines Used:

XRCP8, XREXT, XRMOV, XMGS

RTE Routines Used:

IAND
XXDEF

**Fortran Calling Procedure:**

CALL XXDEF

**Inputs from Common:**

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

**Internal Variables:**

DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE
DEFAULT INTERFACE TABLE CAN BE READ INTO
DEFNAME - (INTEGER, 3 WORDS) ARRAY WHERE INTERFACE TABLE
NAME IS CREATED FROM PROCESSOR NAME

**Common Used:**

EQUIVALENCE (KE(5), MASTA), (KE(6), SUBSTA),
+ (KE(13), INTMAN), (KE(16), SERCH),
+ (KE(19), BSEP), (KE(20), REPBUF),
+ (KE(142), ICR ),
+ (XS(6), TPMTAB), (XS(14), DEFNAME),
+ (XS(18), LEM1 ), (XS(19), LEM2 ),
+ (XS(20), IDC), (XS(20), RRTC)

**RTE Routines Used:**

CLOSE, EXEC, KCVT, OPEN, READF

**FDS Routines Used:**

XERTM, XREQ, XDEXT, XRMOV, XRMSG, XRCK, XRUPK
1 BEGIN XXDEF
2 CREATE DEFAULT INTERFACE TABLE NAME FROM PROCESSOR NAME
3 CALL OPEN TO OPEN FILE
4 ERREXIT IF OPEN ERROR TO :FILERR:
5 CALL READ TO READ RECORD 1
6 ERREXIT IF READ ERROR TO :FILERR:
7 IF THERE IS LITERAL DATA THEN
8 CALL READ TO READ RECORD 2
9 ERREXIT IF READ ERROR TO :FILERR:
10 ENDIF
11 CALL CLOSE TO CLOSE FILE
12 ERREXIT IF CLOSE ERROR TO :FILERR:
13 CALL XRMSG TO ALLOCATE & STORE GINTAB
14 ERREXIT IF RETURN CODE IS NOT ZERO TO :MGERR:
15 IF DEFAULT TABLE IS INCOMPLETE THEN
16 SET INTERFACE TABLE NAME IN XE TO GINTAB
17 SET SUBSTATE TO INTERFACE TABLE EDITOR
18 CALL XERTN TO IMPLICITLY CALL INT EDITOR **NO RETURN**
19 ENDIF
20 1 EXIT XXDEF
21
22 :FILERR:
23 CALL CLOSE TO CLOSE FILE
24 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
25 1 EXIT XXDEF
26
27 :MGERR:
28 CALL XRMSG TO DISPLAY SPACE ERROR
29 CALL EXEC TO FREE CLASS NUMBER
30 SET MASTER STATE TO DIRECTIVE LEVEL
31 CALL XERTN TO RETURN TO EXEC **NO RETURN**
32 1 END XXDEF
Fortran Calling Procedure:

CALL XXEXEC

Inputs from Common:
- XE(10) SESTRA, XE(11) SESEND, XE(12) SESEPTR,
- XB(250) SELEN, XB(251) SEQTAB, XS(13) FLGTAB

Outputs to Common:
- XE(6) SUBSTA, XE(10) SESTRA, XE(12) SESEPTR,
- XE(13) INTNAM, XE(16) PRCHAM, XB(235) RESIND

Common Used:
- Equivalence (XE(5), MASSTA, XE(6), SUBSTA),
- (XE(10), SESTRA, XE(11), SESEND),
- (XE(12), SESEPTR, XE(13), INTNAM),
- (XE(16), PRCHAM, XE(235), RESIND),
- (XB(249), SESEND),
- (XB(250), SELEN), (XB(251), SEQTAB),
- (XS(1), IPARM), (XS(13), FLGTAB)

FDX Routines Used:
- XREQ, XREX, XRG, XROV, XROMS,
- XSSET, XVPM, XDEF, XXTMP

RTE Routines Used:
- RMPPR
1 BEGIN XXX
2 INITIALIZE RESET INDEX TO ZERO
3 IF TABLE FLAG SAYS SETTAB IN XD 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 I ERROR TRYING TO EXECUTE THE SEQUENCE THEN
10 SET SEQUENCE POINTER TO SEQUENCE # IN ERROR
11 FIND ENTRY IN ERROR AND SAVE IN XE
13
14 :INT1: INTERFACE TABLE NOT SPECIFIED
15 CALL XDEF TO READ UP DEFAULT TABLE
16 CALL XTPMP TO SET UP A TEMPORARY ENTRY
17 PERFORM XEE TO EXECUTE **NO RETURN**
18
19 :ERR1: SET MESSAGE TO INTERFACE TABLE NOT FOUND
20
21 :INT2: INTERFACE TABLE NOT COMPLETE
22 SET SUBSTATE TO INTERFACE TABLE EDITOR
23 CALL XERNT TO RETURN TO EXEC **NO RETURN**
24
25 :ERR2: SET MESSAGE TO INT TABLE NOT FOR PROCESSOR
26
27 :ERR3: SET MESSAGE TO VERSIONS DO NOT MATCH
28
29 :RESET: RESET REQUESTED ON TERMINATION
30 CONVERT SEQUENCE # TO INDEX
31 SAVE SEQUENCE INDEX AND SEQUENCE NUMBER IN COMMON
32 EXIT XE
33
34 :ERR4: SET MESSAGE TO RESET SECR NOT FOUND
35
36 :ERR5: SET MESSAGE TO PROCESSOR ABENENDED
37
38 :ERR6: SET MESSAGE TO AWA OVERFLOW
39 ENDCASE
40 CALL XMSG TO DISPLAY ERROR
41 FORMAT SEQUENCE ENTRY INTO ASCII
42 CALL XMSG TO DISPLAY SEQUENCE TABLE ENTRY IN ERROR
43 IF MODE IS SEMI-AUTO AND ENTRY IS NOT AN OVERRIDE THEN
44 RESET OLD INDEX TO RE-EXECUTE THIS ENTRY
45 ENDF
46 IF MODE IS AUTO THEN
47 SET MASTER STATE TO ZERO
48 CALL XRNT TO ABORT SEQUENCE *** NO RETURN ***
49 ENDF
50 ENDF
51 RETURN
52 END XXX
PROCEDURE FOR CALLING XIXMAN:

CALL XIXMAN

XXMAN IS THE MANUAL EXECUTION CONTROLLER

INPUTS FROM COMMON:

XE(5) MASSTA, XB(249) SEINO

OUTPUTS TO COMMON:

XE(10) SERSTR, XE(11) SEREND, XB(249) SERNO,

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

COMMON USED:

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

+ (XE(11), SEREND),

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

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

FDX ROUTINES USED:

XEXT, XMSG, XCTOM, XXEKE,

XXDEC, XXDEF, XXSTO, XXTMP

RTE ROUTINES USED:

IO
BEGIN XXMAN
IF ENTRY IS FROM A DIRECTIVE THEN
  SET HYSTA TO INDICATE RE-ENTRY
  DO UNTIL PERCNT IS ENTERED
    :PROMPT: CALL XCOMP TO PROMPT FOR PRNAME, "ITNAME"
    IF PERCENT IS NOT ENTERED THEN
      ERREXIT IF CR ENTERED TO :PROMPT:
      CALL XXDEC TO DECODE RESPONSE
      ERREXIT IF INVALID RESPONSE TO :PROMPT:
      SET SER #5 IN :E ZEROS
      SET #ENTRIES IN SEFTAB IN XE TO 1
      CALL XXSTO TO STORE SEQUENCE TABLE
      IF IT NAME IN SEFTAB IS $INTAB THEN
        PROCESSOR REQUIRES AN IT THEN
        CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
        ENDF
      CALL XXexe TO EXECUTE SEFTAB
      ENDF
    ENDIF
  ENDIF
ENDDO
ELSE AM BEING REENTERED FROM INTERFACE TABLE EDITOR
  CALL XXMP TO SET UP TO EXECUTE A TEMPORARY TABLE
  PERFORM XXMAN **NO RETURN**
END IF
RETURN
:PROMPT: CALL XRMSC TO DISPLAY ERROR
PERFORM XXMAN **NO RETURN**
END XXMAN
**FORTRAN CALLING SEQUENCE:**

CALL XXSEM

**INPUTS IN COMMON:**

XE(5) MASSTA, XE(10) SERSTA, XE(11) SEGEND, XE(12) SEQ'TRA,
XE(14)TABEND, XB(1) NPROC, XB(2) LINO,
XB(245) SERNO, XB(250) SERLEN, XB(251) SERTAB

**OUTPUTS IN COMMON:**

XE(5) MASSTA, XS(13) FLGTAB

**INTERNAL COMMON USED:**

XE(139) EXEND - ENDING SEQUENCE NUMBER USED TO
TERMINATE SEQUENCE
XE(141) CURIND - INDEX TO THE CURRENT ENTRY BEING EXECUTED
XB(235) RESIND - INDEX TO RESET CURRENT INDEX TO
XB(236) ASCENT - ASCII SEQUENCE TABLE ENTRY USED TO PRACT
THE USER
XB(246) OLDIND - INDEX TO THE LAST ENTRY EXECUTED IN THE
SEQUENCE TABLE

**COMMON USED:**

<table>
<thead>
<tr>
<th>EQUIVALENCE</th>
<th>XE(5)</th>
<th>MASSTA</th>
<th>XE(10)</th>
<th>SERSTA</th>
<th>XE(11)</th>
<th>SEGEND</th>
<th>XE(12)</th>
<th>SEQ'TRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ (XE(12), SERPR)</td>
<td>(XE(83), EOD )</td>
<td>+ (XE(86), INTGR )</td>
<td>(XE(100), AMPT)</td>
<td>+ (XE(139), EXEND )</td>
<td>(XE(140), TABEND)</td>
<td>+ (XE(141), CURIND)</td>
<td>(XE(145), COMPOS)</td>
<td>+ (XB(17), NPROC)</td>
</tr>
<tr>
<td>+ (XB(235), RESIND)</td>
<td>+ (XB(236), ASCENT)</td>
<td>(XB&quot;AB&quot;, OLDIND)</td>
<td>+ (XB(249), SERNO)</td>
<td>+ (XB(250), SERLEN)</td>
<td>+ (XS(13), FLGTAB)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FDS ROUTINES USED:**

SRCPR, XREST, XRIG, XRMOV,
XRMDG, XRSET, XDDEC, XXDEF, XXEXF,
XXSTO, XRMP, XXCM
BEGIN XXSEM:
  IF ENTRY IS FROM A DIRECTIVE THEN
  SET MASTER STATE TO SAT REENTRY
  DO FOR # ENTRY, IN SEQUENCE TABLE
  SEARCH LIB FOR PROCESSOR NAME
  ERROR IF NOT FOUND TO :ERROR:
  MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
  ENDIF
  SAVE ORIGINAL ENDING SEQUENCE #
  SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
  SET FLSTAB TO SAT EXECUTE ENTIRE SEQTAB
  CALL XXSTO TO STORE SEQTAB IN AWAY
  ELSE (I AM BEING REENTERED FROM INTF)
  CALL XXTRP TO SET UP TEMPORARY EXECUTION
  CALL XXERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQTAB
  IF RESET WAS REQUESTED THEN
  RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
  ELSE
  EXIT XXAUT IF TERMINAL ENTRY WAS JUST EXECUTED
  SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
  ENDIF
  ENDIF
DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
CALL XXCM TO PROMPT USER WITH CURRENT ENTRY
EXIT XXSEM IF RESPONSE IS X
IF RESPONSE IS CR THEN
  IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
  CALL XXDF TO READ UP DEFAULT TABLE
  ENDIF
  SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
  SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
  CALL XXERE TO EXECUTE
  SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
  ELSE
  IF RESPONSE WAS A SEQUENCE # THEN
  ERROR IF NUMBER IS ZERO TO :ERROR:
  SEARCH SEQUENCE TABLE FOR SEQUENCE #
  ERROR IF NUMBER IS NOT FOUND TO :ERROR:
  SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
  ELSE
  IF RESPONSE IS AN AMPSAND THEN
  SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
  ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
  CALL XDEC TO DECODE PROCESSOR NAME, IF NAME RESPONSE
  IF RESPONSE IS VALID THEN
  CALL XXSTO TO STORE OVERRIDDING ENTRY
  SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
  SET CURRENT SEQUENCE # TO FIRST ENTRY
  ENDIF
  ENDIF
ENDIF
ENDIF
ENDDO
EXIT XXSEM
:ERROR:
CALL XXMSG TO DISPLAY ERROR MESSAGE
END XXSEM
CALLING PROCEDURE FOR XXSTO:
CALL XXSTO
XXSTO SETS UP TO REQUEST MANAGER TO STORE GSETO AND HANDLE ANY ERRORS RETURNED BY THE MANAGER
INPUTS FROM COMMON:
XE(250), XE(251), SECTAB, XE(13) FLGTAB
OUTPUTS TO COMMON:
XE(5) MASSTA
COMMON USED:
EQUIVALENCE (XE(5), MASSTA), (XE(10), RECV), (XE(20), RENBUF), (XE(250), SECTAB), (XE(251), SECTAB), (XE(13), FLGTAB)
FDS ROUTINES USED:
XREQ, XRMOV, XRMSE, XERTN
RTE ROUTINES USED:
EXEC
BEGIN XISTO
   SET CLASS NUMBER TO ZERO
   IF TABLE FLAG SAYS STORE ENTIRE TABLE THEN
      CALL EXEC TO WRITE ENTIRE TABLE
      SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN DB
      CALL XMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER
   ELSE
      CALL EXEC TO WRITE ONLY FIRST ENTRY
      SET LENGTHS IN REQUEST BUFFER TO 7 WORDS
      CALL XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER
      ENDB
   CALL XREP TO REQUEST MANAGER TO STORE .SEEXTO
   IF RETURN CODE IS NOT ZERO THEN
      CALL XREP TO WRITE SPACE ERROR
      CALL EXEC TO RELEASE CLASS 8
      SET MASTER STATE TO ZERO
      CALL XREP TO RETURN TO EXEC **NO RETURN**
   ENDB
RETURN
1 END XISTO
ASSGN - DATA ASSIGNMENT PROCESSOR
- SCHEDULED BY FDS

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

INPUTS FROM THE MANAGER:

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

INPUTS FROM THE INTERFACE TABLE:

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

OUTPUTS TO THE AMA:

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

INTERNAL VARIABLES:

BLANK COMMON - ASGCOMMON 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(7) = 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>LASTSY</td>
<td>1</td>
<td>40</td>
<td>LAST SYMBOL TABLE ENTRY DEFINED</td>
</tr>
<tr>
<td>SYMTAB</td>
<td>12,81</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>STING</td>
<td>247</td>
<td>1013</td>
<td></td>
</tr>
<tr>
<td>RESULT</td>
<td>4,35</td>
<td>1260</td>
<td>RESULT STACK USED DURING POST-</td>
</tr>
<tr>
<td>FIX STRING EVALUATION (EACH EN-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>close word 1-3 CONTAIN VALUE;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD 4 = DATA TYPE). DATA TYPE=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-2,3 FIXED DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1 SYMBOL TABLE INDEX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSGN</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPRND$</td>
<td>5,9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>1 CD4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE DEFINITIONS:**

- **-2 DISPLACEMENT**: STACK CONTAINING OPERANDS FOR
- **-3 CHARACTER STRING INDEX**: FUNCTIONS AND ARITHMETIC OPERATIONS AND RESULTS FOR STORING
- **(EA. ENTRY: WORDS 1-3 CONTAIN VALUE; WORD 4 = DATA TYPE)**: SIZED FOR 72 CHARACTER STRING
- **SCRTCH**: SCRATCH AREA
- **REXST**: MANAGER REQUEST FOR XPREX
- **OPINFO**: CONTENTS OF FNCBL OR SYNTAX FOR FUNCTION OR MATHEMATICAL OPERATIONS BEING EVALUATED
- **NUMVDS**: NUMBER OF WORDS TO BE STORED IN OBJECT DATA ELEMENT
- **DATTPS**: DATA TYPES
- **RLTPY**: RESULT STACK POINTER
- **EXPTR**: POST-FIX STRING (POLISH) POINTER
- **CLSREN**: XPREX OPTION WORD FOR QUEUE REQUEST AND CLOSE BUFFER - NO
- **CLSTRN**: DATA TRANSFERED
- **XPREX OPTION WORD FOR QUEUE REQUEST, CLOSE BUFFER AND TRANSFER DATA**
- **MPWDS**: NUMBER OF WORDS PER LOGICAL UNIT OF DATA FOR EACH DATA TYPE
- **DECLAS**: DATA ELEMENT CLASS
- **RNSTR**: RFCHM RANGE, END RANGE, INCREMENT AND SYMBOL INDEX FOR EACH RANGE SPECIFICATION
- **POLISH**: POST-FIX REPRESENTATION OF EXPRESSION
- **SYNTAX**: SYNTAX TABLE FOR VALIDITY TESTS ON EXPRESSION (SEE BELOW)
- **FNCBL**: FUNCTION TABLE CONTAINING DATA REQUIREMENTS FOR EACH FUNCTION
- **MRBUFF**: XPREX BUFFER

**NOTE**: STACKS USED IN THE ASSGN PROCESSOR ARE SIZED FOR THE MAXIMUM POSSIBLE AND OVERFLOW IS NOT TESTED
<table>
<thead>
<tr>
<th>Tokens</th>
<th>Input</th>
<th>Output</th>
<th>Token #</th>
<th>Data Type</th>
<th>Precedent</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD4</td>
<td>CDA</td>
<td>CDA</td>
<td>(EOS)</td>
<td>-</td>
<td>4</td>
<td>OP(LBRKT)</td>
</tr>
<tr>
<td>CD4</td>
<td>INTEGER</td>
<td>REAL</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>OP(LBRKT)</td>
</tr>
<tr>
<td>CD4</td>
<td>DOUBLE</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>OP(LBRKT)</td>
</tr>
<tr>
<td>CD4</td>
<td>TNAME</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>OP(LB, MUL)</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1000B</td>
<td>=</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>5</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>5</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>400B</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>VR OP(LB, MUL)</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>400B</td>
<td>VR VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)VR VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>200B</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>200B</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>200B</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>200B</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>200B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>100B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>)</td>
<td>-</td>
<td>-</td>
<td>3000B</td>
<td>)RB VR CN</td>
</tr>
<tr>
<td>CD4</td>
<td>SUBSCRIBER</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>6</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>6</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>7</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>7</td>
<td>SAME</td>
</tr>
<tr>
<td>CD4</td>
<td>VR = VARIABLE</td>
<td>OP = OPERATOR</td>
<td>LB = LEFT BRACKET</td>
<td>CN = CONSTANT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>RB = RIGHT BRACKET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT TYPE</td>
<td>OUTPUT TYPE</td>
<td>FUNCTION</td>
<td>DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>ABS</td>
<td>Absolute Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>ACOS</td>
<td>Arccosine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>ACOT</td>
<td>Arccotangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>ATAN</td>
<td>Arctangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>ATAN2</td>
<td>Arctangent 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>COS</td>
<td>Cosine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>COT</td>
<td>Cotangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>EXP</td>
<td>Exponential</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>LOG</td>
<td>Logarithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>LOG10</td>
<td>Log10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>LOG2</td>
<td>Log2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>POW</td>
<td>Power</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SIGN</td>
<td>Sign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQR</td>
<td>Square</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SIN</td>
<td>Sine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SINH</td>
<td>Sine Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TANH</td>
<td>Tangent Hyperbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>SQRT</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>TAN</td>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

```
201 1 CO5
202 1 CO5
203 1 CO5
204 1 CO5
205 1 CO5
206 1 CO5
207 1 CO5
208 1 CO5
209 1 CO5
210 1 CO5
211 1 CO5
212 1 CO5
213 1 CO5
214 1 CO5
215 1 CO5
216 1 CO5
217 1 CO5
218 1 CO5
219 1 CO5
220 1 CO5
221 1 CO5
222 1 CO5
223 1 CO5
224 1 CO5
225 1 CO5
226 1 CO5
227 1 CO5
228 1 CO5
229 1 CO5
230 1 CO5
231 1 CO5
232 1 CO5
233 1 CO5
234 1 CO5
235 1 CO5
236 1 CO5
237 1 CO5
238 1 CO5
239 1 CO5
240 1 CO5
241 1 CO5
242 1 CO5
243 1 CO5
244 1 CO5
245 1 CO5
246 1 CO5
247 1 CO5
248 1 CO5
249 1 CO5
250 1 CO5
251 1 CO5
252 1 CO5
253 1 CO5
254 1 CO5
255 1 CO5
256 1 CO5
257 1 CO5
258 1 CO5
259 1 CO5
```

**BASCOM-Naur Language Definition**

- `<assignment> ::= <replacement> <range> //
  <evaluation> <range>`
- `<replacement> ::= <non-numeric de> = <non-numeric de> //
  <non-numeric de> = "character string" //
  <variable> = free data element`
- `<non-numeric de> ::= free data element //
  free data element <subscript list> //
  character data element //
  character data element <subscript list>`
- `<evaluation> ::= <variable> = <expression> //
  free data element = <expression> //
  free data element <subscript list> =
  <expression>`
- `<variable> ::= fixed data element //
  fixed data element <subscript list>`
- `<subscript list> ::= <subscript list>, <expression> //
  <expression>`
- `<expression> ::= <expression> <additive operator> <term> //
  <unary operator> <term>`
- `<term> ::= <term> <multiplicative operator> <factor> //
  <factor>`
- `<factor> ::= <power> <power> //
  <power>`
- `<power> ::= <expression> //
  <operand>`
- `<unary operator> ::= <additive operator> //
  0`
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Code Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>260</td>
<td>1 C05</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>261</td>
<td>1 C05</td>
<td></td>
<td>SINGLE PRECISION REAL</td>
</tr>
<tr>
<td>262</td>
<td>1 C05</td>
<td></td>
<td>DOUBLE PRECISION REAL</td>
</tr>
<tr>
<td>263</td>
<td>1 C05</td>
<td>RANGE</td>
<td>RANGE</td>
</tr>
<tr>
<td>264</td>
<td>1 C05</td>
<td>RANGE</td>
<td>LIMITS</td>
</tr>
<tr>
<td>265</td>
<td>1 C05</td>
<td>RANGE</td>
<td>LIMITS</td>
</tr>
<tr>
<td>266</td>
<td>1 C05</td>
<td></td>
<td>LIMITS</td>
</tr>
<tr>
<td>267</td>
<td>1 C05</td>
<td>LIMITS</td>
<td>LIMITS</td>
</tr>
<tr>
<td>268</td>
<td>1 C05</td>
<td>LIMITS</td>
<td>LIMITS</td>
</tr>
<tr>
<td>269</td>
<td>1 C05</td>
<td>LIMITS</td>
<td>LIMITS</td>
</tr>
<tr>
<td>270</td>
<td>1 C05</td>
<td>LIMITS</td>
<td>LIMITS</td>
</tr>
</tbody>
</table>
DDDSP - DATA BOX DISPLAY PROCESSOR

- SCHEDULED BY FDS

DDDSP PROCESSES THE DATA PRODUCED BY SCANNING FUNCTIONAL
PROCESSES. 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:

XIDM - READS AND EDITS INTERFACE TABLE

XIDP1 - READS OR LIST NAMES IN
INTERFACE TABLE AGAINST NAMES IN
DATABASE 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 LU DEVICE
WITH SUBROUTINE XIDOT

DDDSP MERELY CALLS THESE OVERLAYS IN THE PROPER ORDER AND
EXITS

INPUTS TO DDOSP FROM INTERFACE TABLE

DATBOX - DATA BOX FILE NAME

NVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER

KEIP - DISPLAY VARIABLE SCALE LIST SET UP BY USER

VIDEF - CONSTRAINT VARIABLE DEFINITION LIST

INPUTS TO DDOSP 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

(1) - NO OF ENTRIES IN SUMMARY TABLE
(2) - X SCAN VARIABLE (6 CHAR)
(3) - X SCMS SUBSCRIPT (INT OR ZERO)
(4) - XSECOND SUBSCRIPT (INT OR ZERO)
(7) - X UNITS (6 CHAR)
(10) - X CENTROID (REAL)
(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)
(22) - Y CENTROID (REAL)
(23) - YX INCREMENT (REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPENDANT VARIABLE NAME AND UNITS IN RECORDS 36 48 AND 5

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

OUTPUT FROM DBDSP
- DISPLAY IS OUTPUT TO LUM IDENTIFIED IN THE INTERFACE TABLE
397  CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
398  CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)
399  CD 4 YSCANN - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
400  CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
401  CD 4 ZTABLE - TABLE IN COMMON FOR ZTAB Variable NAMES AND UNITS
402  CD 4 ZMVUL - UNITS LIST FOR VARIABLES SCANNED BY ZMVUL ENDSCAN
403  CD 4 ZTAB - VALUES FOR ZMVUL VARIABLE(S) - 1 TO 32 VALUES/RECORD
404  CD 4 PU2NS - COMMUNICATION BUFFER FOR RMPAR - LU, USER ID, FLAGS
405  CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATN
406  CD 4 LUSNP - RMPAR WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
407  CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRM
408  CD 4 DEBUG
409  CD 4 SELECT - SELECT =0 PROMPT = SELECT NOT 0 RUN ALL DISPLAYS TO O/P
410  CD 4 CARTAG - CARTAG USED TO LOCATE DATA BOX
411  CD 4 CYVALU -
412  CD 4 INCD -
413  CD 4 ***USE ROUTINES
414  CD 5
415  CD 5 XPRM, XELBS, XPIIT, RMPAR
416  CD 5
417  CD 5
418  CD 5
419  CD 5
420  CD 5
421  CD 5
DEFIN is an FDS processor scheduled by the Manager.

DEFIN allocates data elements in the AMA that were specified by the parameter keyword DEFINE. If the data element already exists, it is deleted and reallocated. Data elements are initialized to zeros (character strings to blanks).

Inputs from the Manager:
- LU - Logical unit of the user's terminal
- DEBUG - Flags for debug

Inputs from the interface table:
- DEFINE - String containing data element name(s)
- Optional I and J dimensions and a required type

Outputs to the AMA:
- Set of data element(s) requested

Internal variables:
- INTBUF - Interface table header
- ILEN - Length of symbolic string
- ITOKEN - Position within the symbolic string
- NAME - Buffer for valid names to be allocated
- BUFFER - Buffer area for FGET and XPRINT
- NAME - Number of names in symbolic string
- NEXTNAME - Token position for next name
- STRING - Symbolic string input to DEFINE

External routines used:
- EXEC, IAMS, KCVT, KMPAR, XPIET
- XPREQ, XPAR,XR06G, XIRDFT, XRMSG
527 1 BEGIN XDPH
528 2 SET DIM AND JDIM TO 1
529 3 IF TOKEN IS NOT A NAME THEN
530 4 SET ERROR CODE
531 5 ELSE
532 6 MOVE NAME INTO REQUEST
533 7 INCREMENT TO NEXT TOKEN
534 8 IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREN) THEN
535 9 INCREMENT TO NEXT TOKEN
536 10 IF TOKEN IS NOT AN INTEGER OR
537 11 TOKEN IS NOT ZERO THEN
538 12 CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
539 13 EXIT TO :PERR1:
540 14 ENDIF
541 15 SET IDIM TO THIS TOKEN
542 16 INCREMENT TO NEXT TOKEN
543 17 IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
544 18 INCREMENT TO NEXT TOKEN
545 19 IF TOKEN IS NOT AN INTEGER OR
546 20 TOKEN IS NOT ZERO THEN
547 21 CALL XIMSG TO DISPLAY ERROR "INVALID JDIM"
548 22 EXIT TO :PERR1:
549 23 ENDIF
550 24 SET JDIM TO THIS TOKEN
551 25 INCREMENT TO NEXT TOKEN
552 26 ENIF
553 27 IF TOKEN IS NOT A RIGHT PAREN THEN
554 28 CALL XIMSG TO DISPLAY ERROR "INVALID SUBSCRIPT DELIMETER"
555 29 EXIT TO :PERR1:
556 30 ENDIF
557 31 INCREMENT TO NEXT TOKEN
558 32 IF TOKEN IS NOT A BEGIN TYPE FIELD SLASH THEN
559 33 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
560 34 EXIT TO :PERR1:
561 35 ENDIF
562 36 INCREMENT TO NEXT TOKEN
563 37 IF TOKEN IS NOT A NAME THEN
564 38 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
565 39 EXIT TO :PERR1:
566 40 ENDIF
567 41 INCREMENT TO NEXT TOKEN
568 42 STARTSEARCH FOR ALL VALID TYPES
569 43 EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
570 44 SET TYPE AND CLASS IN REQUEST
571 45 COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
572 46 IF SIZE IS TOO LARGE (>1200 WORDS) THEN
573 47 CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
574 48 EXIT TO :PERR1:
575 49 ENDIF
576 50 ENLOOP
577 51 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
578 52 EXIT TO :PERR1:
579 53 ENDOSEARCH
580 54 INCREMENT TO NEXT TOKEN
581 55 IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
582 56 NEXT TOKEN IS NOT A COMMA THEN
583 57 CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
**ENDSC**

ENDSC is an FDS processor scheduled by the manager.

ENDSC processor is used in conjunction with scan only. It marks the end of the series of processors to be scanned. It has no interface table.

**INPUTS FROM THE MANAGER:**

**LU** - Logical unit of the user's terminal

**USEID** - User ID code

**FLAGS** - Flags for debug

**OUTPUTS FROM AWA:**

**&SCANB** - 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, WRTIF**

**FDS ROUTINES USED:**

**XPXIT, XREX, XRMOV, XUDBS, XVPAM, XIFCL, XIMSG**
641 1 BEGIN ENDC
642 2 CALL RMFAE TO GET INPUTS FROM MANAGER
643 3 SAVE REQUEST TO RETRIEVE $SCNTH FROM ANA
644 4 CALL XVPAW TO REQUEST ANA MANAGEMENT
645 5 ERREXIT IF $SCNTH NOT FOUND TO :ERR4:
646 6 CALL EXEC TO READ IN $SCNTH
647 7 SAVE REQUEST TO RETRIEVE SUNITAB IN REQBUF
648 8 CALL XVPAW TO REQUEST ANA MANAGEMENT
649 9 ERREXIT IF NOT FOUND TO :ERR4:
650 10 CALL EXEC TO READ IN SUNITAB
651 11 IF SUMMARY TABLE IS LARGER THAN 32 ENTRIES THEN
652 12 SET SIZE OF SUNITAB TO 32 ENTRIES
653 13 ENDF
654 14 CALL WRITF TO WRITE SUNITAB TO DATBOX
655 15 ERREXIT IF WRITF ERROR TO :ERR4
656 16 IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
657 17 THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
658 18 CALL READ TO READ HEADER RECORD
659 19 ERREXIT IF READF ERROR TO :ERR4:
660 20 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
661 21 CALL WRITF TO WRITE UPDATED HEADER
662 22 ERREXIT IF WRITF ERROR TO :ERR4:
663 23 CALL CLOSE TO CLOSE DATBOX
664 24 SAVE REQUEST TO DEL/VER ABS $SCNTH IN REQBUF
665 25 IF THERE ARE REMAINING SCANS I: $SCNTH THEN
666 26 CALL EXEC TO WRITE REMAINING $SCNTH
667 27 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCNTH
668 28 ENDF
669 29 SET RETURN PARAMETER TO NORMAL RETURN
670 30 ELSE
671 31 PERFORM SETXY
672 32 ENDF
673 33 CALL XVPAW TO REQUEST ANA MANAGEMENT
674 34 CALL XPSIT TO TERMINATE WITH RETURN PARAMETERS
675 35 EXIT ENDC
676 36 :ERR4:
677 37 CALL XIMSG TO DISPLAY ERROR
678 38 CALL XPSIT TO ABEND PROCESSOR
679 39 END ENDC
1 BEGIN SETXY
2 IF XCUR IS END STEP THEN
3 IF THERE IS 1 VARIABLE THEN
4 SET X TO CENTROID
5 CALL POSNT TO POSITION FILE TO CENTROID RECORD
6 ERREXIT IF POSNT ERROR TO :ERR4:
7 ELSE
8 IF YCUR IS END STEP THEN
9 SET YCUR TO ZERO
10 SET X TO CENTROID
11 SET Y TO CENTROID
12 CALL POSNT TO POSITION TO CENTROID RECORD
13 ERREXIT IF POSNT ERROR TO :ERR4:
14 ELSE
15 SET XCUR TO (-XSTEP)
16 IF XSTEP IS ZERO AND Y IS CENTROID THEN
17 INCREMENT YCUR BY 1
18 CALL WRIF TO WRITE DUMMY AS CENTROID RECORD
19 ERREXIT IF WRIF ERROR TO :ERR4:
20 ENDIF
21 COMPUTE X AS (XCENT + XINCR + FLOAT (XCUR))
22 COMPUTE Y AS (YCENT + YINCR + FLOAT (YCUR))
23 ENDIF
24 ELSE
25 INCREMENT XCUR BY 1
26 IF THERE IS 1 VARIABLE AND X IS THE CENTROID OR
27 THERE ARE 2 VARIABLES AND X IS THE CENTROID AND Y IS THE CENTROID THEN
28 INCREMENT YCUR BY 1
29 CALL WRIF TO WRITE DUMMY AS CENTROID RECORD
30 ERREXIT IF WRIF ERROR TO :ERR4:
31 ENDIF
32 COMPUTE X AS (XCENT + XINCR + FLGAT (XCUR))
33 IF THERE ARE 2 VARIABLES THEN
34 COMPUTE Y AS (YCENT + YINCR + FLOAT (YCUR))
35 ENDIF
36 ENDIF
37 DJ FOR # SCAN VARIABLES
38 CALL EXEC TO WRITE VARIABLE
39 SAVE REQUEST TO STORE VALUES FOR VARIABLE IN RERBUF
40 END
41 CALL EXEC TO WRITE $SCMTB
42 SAVE REQUEST TO STORE NEW $SCMTB
43 SET RETURN PARAMETERS TO RESET SEQUENCE NUMBER
44 END SETXY
730   1 CD************
731   1 CD0  SCAN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER
732   1 CD0
733   1 CD0
734   1 CD0
735   1 CD0
736   1 CD0
737   1 CD0  SCAN PROCESSOR EXECUTES A SERIES OF PROCESSORS ITERATING ON XSCAN AND
738   1 CD0  YSCAN (IF ENTERED) VALUES COMPUTED USING THE CENTROID, THE INCREMENT,
739   1 CD0  AND THE CURRENT STEP NUMBER, CREATING A DATA BOX FILE.
740   1 CD0
741   1 CD0
742   1 CD0
743   1 CD0
744   1 CD0
745   1 CD0
746   1 CD0
747   1 CD0
748   1 CD0
749   1 CD0
750   1 CD0
751   1 CD0
752   1 CD0
753   1 CD0
754   1 CD0
755   1 CD0
756   1 CD0
757   1 CD0
758   1 CD0
759   1 CD0
760   1 CD0
761   1 CD0
762   1 CD0
763   1 CD0
764   1 CD0
765   1 CD0
766   1 CD0
767   1 CD0
768   1 CD0
769   1 CD0
770   1 CD0
771   1 CD0
772   1 CD0
773   1 CD0
774   1 CD0
775   1 CD0
776   1 CD0
777   1 CD0
778   1 CD0
779   1 CD0
780   1 CD0
781   1 CD0
782   1 CD0
783   1 CD0
784   1 CD0
785   1 CD0
786   1 CD0
787   1 CD0
788   1 CD0
789   1 CD0
790   1 CD0
791   1 CD0
792   1 CD0
793   1 CD0
794   1 CD0
795   1 CD0
796   1 CD0
797   1 CD0
798   1 CD0
799   1 CD0
800   1 CD0
801   1 CD0
802   1 CD0
803   1 CD0
804   1 CD0
805   1 CD0
806   1 CD0
807   1 CD0
808   1 CD0
809   1 CD0
810   1 CD0
811   1 CD0
812   1 CD0
BEGIN SCAN
CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XPSET TO GET PROCON AND # SCAN VARIABLES
EXIT IF # SCAN VARIABLES < 1 OR > 2 TO ERR3:
GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
EXIT IF SUMTAB IS A LITERAL TO ERR3:
EXIT IF DISPLACEMENT IS NOT AN ELEMENT BOUNDARY TO ERR3:
GET DATA BOX NAME FROM INTERFACE TABLE
DO FOR # SCAN VARIABLES
CALL XPATH TO GET NAME AND DISPLACEMENT
COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND IDIM
CALL XPGET TO GET UNITS, CENTROID, IMER, # STEPS
EXIT IF STEPS < 0 OR > 2 TO ERR3:
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR BSETB AND BSCNTB
CALL XVPAM TO REQUEST AWA MANAGEMENT
CALL EXEC TO READ IN BSETB
IF BSCNTB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF BSCNTB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN BSCNTB
EXIT IF # SCANS = MAXIMUM ALLOWED (4) TO ERR3:
EXIT IF THIS DATABASE NAME IS ALREADY IN USE TO ERR3:
ENDIF
EXIT IF THIS IS THE LAST ENTRY IN BSETB TO ERR3:
GET THE SEQUENCE NUMBER OF THIS SCAN FROM BSETB
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH BSETB FOR THE SEQUENCE NUMBER
EXIT IF THIS SCAN IS THE LAST ENTRY IN BSETB TO ERR3:
IF THIS IS A SEMI-OVERRIDE (2 PROCESSOR NAMES NOT EQUAL) THEN
SET RESET NUMBER TO THIS ENTRY SEQUENCE NUMBER
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
ENDF
END

BEGIN XCHRN

SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT

CASE (RESULT DATA TYPE) :
CHSTR; :CHRFRE; :CHRFRE; :FIXERR;

CASE (RESULT DATA TYPE) :
FIXERR; :FIXERR;

DETERMINE # WORDS IN CHARACTER STRING
MOVE CHARACTER STRING TO RESULT LOCATION
IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
BLANK FILL AFTER CHARACTER STRING

:CHRFRE:
IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
POP DISPLACEMENT FROM RESULT STACK
ELSE
SET DISPLACEMENT = 0
ENDIF
POP RESULT OPERAND FROM STACK
IF RESULT OPERAND IS FREE, THEN
SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
ELSE CHARACTER = CHARACTER
SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
ENDIF
CALL XPREQ TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
BLANK FILL REMAINING DATA
ENDIF
ENDCASE
EXIT XCHRN

SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO"
NUMERICAL DATA"
CALL XING TO OUTPUT MESSAGE TO USER
CALL XLSIS TO LIST SYMBOLIC STRING
CALL XEXIT TO EXIT PROCESSOR
EXIT XCHRN
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 = SYMBOLIC 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 SYMBOLIC STRING IF FOUND,
  OR 'END' IF NOT FOUND
1083 1 BEGIN XZDFT
1084 2 SET INDXX TO START
1085 3 DO UNTIL INDEX .GE. END
1086 4 IF ARRAY(INDEX) .EQ. TOKEN, THEN
1087 ELSE
1088 CASE ARRAY(INDEX)+1 ( SIMPLIFY)
1089 5 :ADD1: 0 END OF STRING
1090 :ADD2: 1 INTEGER
1091 :ADD3: 2 REAL
1092 :ADD4: 3 DOUBLE
1093 :ADD1: 4 UNKNOWN
1094 :ADD4: 5 NAME
1095 :ADD1: 6 UNKNOWN
1096 :ADD1: 7 UNKNOWN
1097 :CALST: 8 CHARACTER STRING
1098 :ADD1: 9 UNKNOWN
1099 :ADD1: 10 +
1100 :ADD1: 11 -
1101 :ADD1: 12 *
1102 :ADD1: 13 /
1103 :ADD1: 14 <
1104 :ADD1: 15 >
1105 :ADD1: 16 #
1106 :ADD1: 17 @
1107 :ADD1: 18 =
1108 :ADD1: 19 %
1109 :ADD1: 20 ?
1110 :ADD1: 21 !
1111 :ADD1: 22 )
1112 :ADD2: 23 START OF SYMBOLIC STRING
1113 :ADD1: 24 END OF STRING
1114 :ADD1: 25 %
1116  5  :ADD1:  26 BACK SLASH
1117  5  :ADD1:  27 $ 
1118  5  :ADD1:  28 .
1119  5  :ADD1:  29 OPEN BRACKET
1120  5  :ADD1:  30 CLOSE BRACKET
1121  5  :ADD2:  31 REPEAT
1122  5  :ADD1:  32 ,
1123  5  :ADD1:  33 ;
1124  5  :ADD1:  INDEX=INDEX+1
1125  5  :ADD2:  INDEX=INDEX+2
1126  5  :ADD3:  INDEX=INDEX+3
1127  5  :ADD4:  INDEX=INDEX+4
1128  5  :CALST:  INDEX=INDEX+2+(ARRAY(INDEX+1)+1)/2
1129  5  ENDCASE
1130  5  ENDF
1131  5  ENDO
1132  5  END
DEPANANT VARIABLE SCANNED (UP T( 32 VALUES
INCLUDING ERROR FLAG WHICH IS FIRST VALUE
IN SUMMARY TABLE)

**OUTPUT FROM XZDP1**

- MASK TABLES CONTAINING THE CONSTRAINT MAILS FOR EACH
- CONSTRAINT WHICH WAS VIOLATED AND I(NICATION 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

**DATA**

- NAME OF DATA BOX TO BE DISPLAYED BY DBSP
- NAME LIST FOR VARIABLES SCANNED BY SCANNED/ENDS
- 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 PAG ED OUT PUT (1-16 PR)
- NUMBER OF DEP DISP VARIABLE PAIR CALCULATORS FOR 0/P
- SET OF INDICATORS FOR CONSTRAINTS VIOLATED/NOT 0
- NUMBER OF STEPS ON EITHER SIDE OF S CENTROID (0 TO 5)
- NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
- NUMBER OF 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)
- NAME OF X VAR NAME AND UNITS
- NAME LIST FOR VARIABLES SCANNED BY SCANNED/ENDS
- VALUES FOR SCAN VARIABLES (1 - 32 VALUES/RECORD
- COMMUNICATION BUFFER FOR RMAP = LU, USER ID, FLAG
- LOGICAL UNIT # FOR XROM CALLING SEQUENCE - USER LOCATIONS
- DBSP - DBSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
- TABLE IN COMMON TO COMMUNICATE WITH XPRD
1402 1 CD ******
1403 1 CD 0  XZDP2 - DATA BOX DISPLAY OVERLAY - PROMPTS USER FOR DISPLAY
1404 1 CD 0  DESIRED, THEN FORMATS AND DISPLAYS DATA ACCORDINGLY
1405 1 CD 0  SCHEDULED BY DBDSP
1410 1 CD 1  XZDP2 ISSUES PROMPTS TO THE USER REQUESTING PAGE NUMBER
1411 1 CD 1  (STARTING ROW NUMBER, NUMBER OF ROWS, AND DESIRED
1412 1 CD 1  LU). SELECT OPTN CAN BE SET WHEN ENTERING
1414 1 CD 1  DBDSP IN THE INTERFACE TABLE TO OUTPUT ALL PAGES
1415 1 CD 1  TO NAMED LU.
1416 1 CD 1  XZDP2 UTILIZES SUBROUTINE XDOT TO ACTUALLY FORMAT
1417 1 CD 1  AND DISPLAY THE SCAN RESULTS.
1438 1 CD 3  ******
1440 1 CD 3  INPUTS TO XZDP2 FROM XZDP1
1442 1 CD 2  MASK TABLES CONTAINING THE CONSTRAINT MASKS FOR EACH
1444 1 CD 2  CONSTRAINT WHICH WAS VIOLATED AND INDICATION OF
1445 1 CD 2  WHETHER OR NOT ANY CONSTRAINT WAS VIOLATED FOR EACH
1446 1 CD 2  ARRAY COORDINATE
1449 1 CD 2  LIST OF VALID DISPLAY DEPENDANT VARIABLES FOR EACH
1450 1 CD 2  PAGE (UP TO 16 PAGES)
1451 1 CD 2  DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES
1435 1 CD 2  ******
1436 1 CD 3  OUTPUT FROM XZDP2
1437 1 CD 3  DISPLAY FORMAT SHOWN IN DOCUMENTATION IS SENT TO
1439 1 CD 3  NAME LU DEVICE
1442 1 CD 3  ******
1444 1 CD 4  XBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPARM
1445 1 CD 4  DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
1446 1 CD 4  DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
1447 1 CD 4  IDV1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
1448 1 CD 4  IDV2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.
1449 1 CD 4  ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XIMSK (MAX OF 8)
1451 1 CD 4  IXSCN1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O)
1452 1 CD 4  IXSCN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O)
1453 1 CD 4  IXSCN3 - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
1454 1 CD 4  IXSCN4 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
1455 1 CD 4  LEMBF - LENGTH OF COMBUF IN 16-BIT WORDS - XPARM
1456 1 CD 4  LENDT - LENGTH IN WDS OF CHAR STRING USED FOR USER PROMPT - XPARM
1457 1 CD 4  MBK - ARRAY CONTAINING MASKS FOR UP TO 8 CONSTRAINTS X 121 SIZE
1458 1 CD 4  NAME - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN
1459 1 CD 4  NCON - NUMBER OF CONSTRAINTS INPUT BY USER (INT/GE2)
1460 1 CD 4  NCOREL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
1461 1 CD 4  NCONVRL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
1461 1 CD 4 NDVARL - LIST OF DEP LISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
1462 1 CD 4 NDVRC - NUMBER OF DEP DISP VAR IN NDVARL LIST (INTEGER)
1463 1 CD 4 SSDVRL - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P
1464 1 CD 4 SSDVRC - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT 0
1465 1 CD 4 NSSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1466 1 CD 4 MSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1467 1 CD 4 RETC - RTM CODE FM XPRDM; 0=NRML, 1=RTM TO EXEC, 2=NULL BFR, 3=ERR
1468 1 CD 4 STRING - CHAR STRING CONTAINING USER PROMPT MESSAGE
1469 1 CD 4 SCORD - LIST OF X VAR VALUES FOR X COORDINATES (1-11 REAL)
1470 1 CD 4 SCOND - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1471 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1472 1 CD 4 TCOND - LIST OF X VAR VALUES FOR Y COORDINATES (1-11 REAL)
1473 1 CD 4 TCOND - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1474 1 CD 4 TUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1475 1 CD 4 ITABLE - TABLE IN COMMON FOR SUTAB VARIABLE NAMES AND UNITS
1476 1 CD 4 MARVUL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN
1477 1 CD 4 SUNTAB - VALUES FOR SCAN VARIABLES - 1 TO 32 VALUES/RECORD
1478 1 CD 4 PARMS - COMMUNICATION BUFFER FOR RMPAR - LU, USER ID, FLAGS
1479 1 CD 4 LU - LOGICAL UNIT # FOR XPRDM CALLING SEQUENCE - USER LOCATM
1480 1 CD 4 LUDP - DDBSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
1481 1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRDM
1482 1 CD 4 SELECT - SELECT =0 PROMPT; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
1483 1 CD 4 WITHOUT PROMPT
1484 1 CD 4 CARTBG - CARTRIDGE USED TO LOCATE DATA BOX
1485 1 CD 4 CARTBG - CHARTRGE USED TO LOCATE DATA BOX
1486 1 CD 5
1487 1 CD 5
1488 1 CD 5
1489 1 CD 5
1490 1 CD 5
1491 1 CD 5
1492 1 CD 5

**USERS ROUTINES**

- XPRDM, EXEC, XZD0T.
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1591 1 CD000000
1592 1 CD09 FORTRAN CALLING PROCEDURE
1593 1 CD01 CALL XZ1SP (STRING, LEN)
1594 1 CD02 1 CD1
1595 1 CD1 XZ1SP REMOVES DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1596 1 CD2 A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1597 1 CD3 WITH BLANKS
1598 1 CD4

1601 1 CD8 INPUT
1602 1 CD9 CALLING SEQUENCE
1603 1 CD2 STRING - INPUT CHARACTER STRING
1604 1 CD2 LEN - NUMBER OF WORDS IN STRING

1609 1 CD2 OUTPUT
1610 1 CD2 CALLING SEQUENCE
1611 1 CD3 STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
1612 1 CD3 BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILL
1613 1 CD3 LEN - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL

1620 1 CD2 NOTES
1621 1 CD2 USES ROUTINES
1622 1 CD2 XR1SP

1629 1 CD2 BEGIN XZ1SP
1630 2 CALL XR1SP TO REMOVE DUPLICATE BLANKS FROM STRING
1631 2 DO WHILE THERE ARE TRAILING WORDS IN STRING
1632 2 SET THIS TRAILING WORD TO BLANKS
1633 2 ENDDO
1634 2 END XZ1SP
1635 1 END XZ1SP
1674 1 CDO FORTRAN CALLING PROCEDURE:
1675 1 CDO CALL XIFNC(ENTRY)
1676 1 CDO
1677 1 CDO
1678 1 CDO
1679 1 CDO************
1680 1 CDO
1681 1 CDO XIFNC IS USED BY THE ASSGN ROUTINE XZPS2 TO EVALUATE FUNCTION
1682 1 CDO OPERATIONS
1683 1 CDO
1684 1 CDO************
1685 1 CDO2 INPUTS
1686 1 CDO2
1687 1 CDO2 ENTRY - FUNCTION TOKEN CURRENTLY BEING PROCESSED
1688 1 CDO2
1689 1 CDO2 FROM ASSGN = LU,SSTRNG,OPRNDs,OPINFO
1690 1 CDO2
1691 1 CDO2
1692 1 CDO************
1693 1 CDO3 OUTPUTS TO ASSGN
1694 1 CDO3
1695 1 CDO3 RESULT,RLTPT,OPRNDs
1696 1 CDO3
1697 1 CDO3
1698 1 CDO************
1699 1 CDO5 EXTERNAL REFERENCES
1700 1 CDO5
1701 1 CDO5 FDS - XPXIT, XRM0V, XILSS, XZMSG
1702 1 CDO5
1703 1 CDO5
1704 1 CDO5 RTE - ABS, AMT, ALOG, ALG0T, AMOD, ATAN, ATAN2, COS, DBS, DATAN,
1705 1 CDO5 DATN2, DCEL, DCOS, DINT, DEEP, DLOG, DLOGT, DMOD, DSI-W, DSI-M, DSQRT,
1706 1 CDO5 DIP, FLOAT, IABS, IDINT, IFIX, ISIGN, MOD, OV0, SIGN, SIM, SINGLE, SGR1,
1707 1 CDO5 TAN, TANH
1708 1 CDO5
1709 1 CDO5
FORTRAN CALLING PROCEDURE:

CALL XIFRE

XIFRE IS USED BY THE ASSGN ROUTINE XIAP2 TO PROCESS DATA ASSIGNMENTS
FOR FREE-TYPE OBJECT DATA ELEMENTS

INPUTS FROM ASGCOM

SYMTAB, SSTRNG, RESULT, RSLTPT, CLSTM, MAPWDS

OUTPUTS TO ASGCOM

CPNDS, RERST, NIIMWDS, RSLTPT

EXTERNAL REFERENCES

FDS - XPREQ, XRMO, XIPCS

RTE - IAMD

-----------------------------------------------------------------------------------
BEGIN XZFR

CASE (RESULT DATA TYPE)
  :FRESTR:, :FREFR:, :FREFR:, :FREFR:
  :FREXR:, :FREFR:

SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
MOVE CHARACTER STRING TO RESULT LOCATION

IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
  POP DISPLACEMENT FROM RESULT STACK
ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPTED
  SET DISPLACEMENT = 0
ENDIF

POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
IF RESULT OPERAND IS FREE, THEN
  CALL XPREG TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
SET # WORDS TO BE STORED IN OBJECT = 1
ELSE FREE = CHARACTER DATA ELEMENT
  CALL XPREG TO RETRIEVE LOGICAL ELEMENT OF CHARACTER DATA FROM RESULT OPERAND
  AT DISPLACEMENT DETERMINED
SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR RESULT OPERAND
ENDIF

CALL XPBCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
SET # WORDS TO BE STORED = RESULT DATA TYPE
END CASE
END XZFR
FORTRAN CALLING PROCEDURE:

CALL XIFXDI

XIFXDI is used by the ASGM routine XZPS2 to process data assignments
for fixed-type object data elements.

INPUTS FROM ASGOM

LU, SYMTAB, SSTRNG, RESULT, RSLTPT, DATYPS, CLSTAN

OUTPUTS TO ASGOM

RSLTPT, REQST, NUMWDS, OPRNDS

FDS - XPREQ, XPARIT, XRMV, XLSS, XMSG, XIPCS

RTE - IAND

ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
ZFXFD
1844 1 CDO    FORTRAN CALLING PROCEDURE
1845 1 CCG
1846 1 CDD    CALL XILSS (LU, STRING, INDEX)
1847 1 CDO
1848 1 CDO
1849 1 C*******
1850 1 CDO
1851 1 CD1    XILSS IS CALLED TO LIST A SYMBOLIC STRING AND AN INDICATOR TO A
1852 1 CD1    PARTICULAR TOKEN IN THAT STRING
1853 1 CD1
1854 1 C*******
1855 1 CDO
1856 1 CDO    INPUT
1857 1 CDO    LU - LOGICAL UNIT NO. FOR OUTPUT OF STRING
1858 1 CDO    STRING - SYMBOLIC STRING TO BE LISTED
1859 1 CDO    INDEX - SUBSCRIPT INTO STRING OF THE TOKEN TO BE INDICATED
1860 1 CDO
1861 1 CDO
1862 1 C*******
1863 1 CDO    OUTPUT
1864 1 CDO
1865 1 CDO
1866 1 CDO    THE SYMBOLIC STRING IS OUTPUT TO THE LU FOLLOWED BY A LINE CONTAINING
1867 1 CDO    AN INDICATOR (UP ARROW) TO THE DESIGNATED TOKEN.
### INTERNAL VARIABLES

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>TOKEN</th>
<th>WORD 1</th>
<th>WORD 2</th>
<th>WORD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CD4</td>
<td>INTERNAL VARIABLES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>CONTROL = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>FOR EACH OF THE TOKEN VALUES 1-32; EACH CONTROL TABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>ENTRY IS 3 WORDS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>WORD 1 (*SIZE) = NO. OF WORDS IN PRINT BUFFER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>OR FLAG DESCRIBING HOW TO COMPUTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>1 CD4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>NAME</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td>WORD 3 (TOKSIZ) = NO. OF WORDS IN SYMBOLIC STRINGS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 C*******
2 NOTES
3 USES ROUTINES
4 EXEC
5 XRI6
6 XRE14
7 XRD18
8 XMSG
9 XRMOV
10 C*******
11 BEGIN XILSS
12 MOVE A 'CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
13 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
14 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
15 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
16 EXIT TO ERROR 2 IF FIELD = 0
17 IF SIZE < 0, THEN
18 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
19 ENDIF
20 IF TOKSIZ < 0, THEN
21 SET TOKSIZ TO SIZE + 2
22 ENDIF
23 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
24 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
25 SET BUFFER POINTER TO 1ST POSITION FOR DATA
26 IF INDICATED TOKEN PROCESSED, THEN
27 OUTPUT LINE WITH INDICATOR
28 ENDIF
29 IF FIELD > 0, THEN
30 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
31 ELSE
32 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
33 SOME: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING
34 AND PUT RESULTS INTO PRINT BUFFER
35 TWO: CALL XRE14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
36 AND PUT RESULTS INTO PRINT BUFFER
37 THREE: CALL XRD18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
38 AND PUT RESULTS INTO PRINT BUFFER
39 FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
40 FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
41 SIX: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
42 INTO PRINT BUFFER FOLLOWED BY AN "R"
43 EXIT: PUT A 'CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
44 INDEX BY 1
45 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
46 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
47 INDICATOR LINE NOT YET OUTPUT, THEN
FORTRAN CALLING PROCEDURE

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

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

NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
A = AREA INDICATOR AS FOLLOWS
  1 - AS
  2 - XB
  3 - EX
  4 - XI
  5 - XS
  6 - XT
  7 - XX
  8 - XL
  9 - DF
10 - SC
NN = MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'
LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED
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 "AANN* MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE"

NOTES:
USES FOS SYSTEM MESSAGE FILE JXRMSG
USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XRMOV, XUGD
2035 1 BEGIN XZMSG
2036 2 BEGIN XZMSG
2037 3 SEPERATE NUMBER INTO AREA AND MESSAGE NUMBER
2038 4 SET NUMBER IN PREFIX
2039 5 READ MESSAGE DIRECTORY RECORD
2040 6 IF AREA VALID
2041 7 THEN
2042 8 SET AREA CODE IN PREFIX
2043 9 IF MESSAGE NUMBER > 0
2044 10 THEN
2045 11 IF VALID MESSAGE NUMBER
2046 12 THEN
2047 13 COMPUTE MESSAGE RECORD NUMBER
2048 14 READ RECORD
2049 15 CALL XRMV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
2050 16 ELSE
2051 17 EXIT TO :ERROR:
2052 18 ENDIF
2053 19 CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
2054 20 IF MESSAGE NUMBER > 0
2055 21 THEN
2056 22 CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
2057 23 ENDIF
2058 24 ELSE
2059 25 SET AREA IN PREFIX
2060 26 ERROR: CALL XRMV TO MOVE 'XZMSG ERROR' INTO BUFFER
2061 27 CALL XRMV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
2062 28 ENDIF
2063 29 OUTPUT BUFFER TO USER'S TERMINAL
2064 30 IF DEBUG IS REQUESTED
2065 31 THEN
2066 32 CALL XUDBG
2067 33 ENDIF
2068 34 END XZMSG
FORTRAN CALLING PROCEDURE:

CALL XIOPR(ENTRY)

XIOPR IS USED BY THE ASSGN ROUTINE XIPS2 TO EVALUATE MATH OPERATIONS

INPUTS:
ENTRY - OPERATOR TOKEN CURRENTLY BEING PROCESSED
FROM ASSGN - LU, SSTRNG, SYNTAX, DATYP, RSLTP, CSLSTR, MAPWDS, RESULT

OUTPUTS TO ASSGN
RESULT, OPRNDS, REGST, RSLTP

INTERNAL VARIABLES
MAPOP - MAPS OPERATOR TOKENS FOR EXECUTION

EXTERNAL REFERENCES
FDS - XPAR, XPAR, XPAR, XPAR, XPAR, XPAR
RTE - IANC, OWF
1 BEGIN XOPR
2 CASE OPERATOR
3 :ADD; :SUBTR; :MULT; :DIVIDE; :INDEX; :SUBSCR;
4 :EXPO; :UNIMINUS;
5 END CASE
6
7 :ADD:
8 PERFORM ADDITION
9 SET RESULT'S DATA TYPE AS APPROPRIATE
10
11 :SUBTR:
12 PERFORM SUBTRACTION
13 SET RESULT'S DATA TYPE AS APPROPRIATE
14
15 :MULT:
16 PERFORM MULTIPLICATION
17 SET RESULT'S DATA TYPE AS APPROPRIATE
18
19 :DIVIDE:
20 PERFORM DIVISION
21 SET RESULT'S DATA TYPE AS APPROPRIATE
22
23 :INDEX:
24 IF RESULT STACK IS NOT EMPTY, THEN
25 GET # OF TOP OPERAND IN RESULT STACK
26 SET # OF WORDS PER ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK
27 ELSE
28 IF EQUABLE SUBSCRIPT FOR OBJECT IS BEING EVALUATED
29 GET # OF TOP ENTRY IN SYMBOL TABLE
30 SET # OF WORDS PER ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
31 ENDF
32
33 :SUBSCR:
34 IF RESULT STACK IS NOT EMPTY, THEN
35 GET # OF TOP ENTRY OF RESULT STACK ENTRY
36 SET # OF WORDS/ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK ENTRY
37 ELSE
38 IF OBJECT IS BEING SUBSCRIPTED
39 SET # OF WORDS/ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
40 ENDF
41
42 :EXPO:
43 PERFORM EXPONENTIATION
44 SET RESULT'S DATA TYPE AS APPROPRIATE
45
46 :UNIMINUS:
47 CHANGE SIGN OF OPERAND FOR RESULT
48 ENDCASE
49
50 IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
51 PUSH RESULT AND TYPE ONTO RESULT STACK
52 EXIT XOPR
53
54 :OVER:
55 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
FORTRAN CALLING PROCEDURE:

CALL XPCS(TARGET,OPNUM)

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

INPUTS

TARGET - DESIRED FDS FIXED DATA TYPE
OPNUM - OPERAND NUMBER FOR ENTRY CURRENTLY BEING SET UP
FROM ASGCOM - LU, SSTANG, RESULT, RSLTP, DATYPS

OUTPUTS TO ASGCOM
RSLTP, OPRING

EXTERNAL REFERENCES
FDS - XPXIT, XRMOV, XLSS, XMSG
RTE - COLE, FLOST, IFIX, OVF, SNGL
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2200</td>
<td>BEGIN XIPCS</td>
</tr>
<tr>
<td>2201</td>
<td>POP OPERAND AND DATA TYPE FROM TOP ENTRY OF RESULT STACK</td>
</tr>
<tr>
<td>2202</td>
<td>IF TARGET TYPE AND DATA TYPE ARE NOT EQUAL, THEN</td>
</tr>
</tbody>
</table>
| 2203 | CASE (TARGET TYPE) :
| 2204 | :INTS: CONVERT OPERAND TO INTEGER |
| 2205 | :REAL: CONVERT OPERAND TO SINGLE PRECISION REAL |
| 2206 | :DOUBLE: CONVERT OPERAND TO DOUBLE PRECISION REAL |
| 2207 | ENDCASE |
| 2208 | ENDF IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER: |
| 2209 | STORE OPERAND AND CURRENT DATA TYPE FOR USE BY FUNCTION OP ARITHMETIC OPER. |
| 2210 | 1 EXIT XIPCS |
| 2211 | 2 :OVER: |
| 2212 | 2 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED" |
| 2213 | 2 CALL XMSG TO OUTPUT MESSAGE TO USER |
| 2214 | 2 CALL XLIST TO LIST SYMBOLIC STRING |
| 2215 | 2 CALL XEXIT TO EXIT PROCESSOR |
| 2216 | 1 END XIPCS |
FORTRAN CALLING PROCEDURE:

CALL XIPS1

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

INPUTS FROM ASGCOM

LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTBL

OUTPUTS TO ASGCOM

EXPTR, RNGSTK, POLISH, SYNTAX

INTERNAL VARIABLES

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

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

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

TKNPT - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING

TOKEN - TOKEN CURRENTLY BEING PROCESSED

EXTERNAL REFERENCES

RTE - IAMO, IMPO

FDS - XPXIT, XLS, XMS, XSYM
1 BEGIN XIPS1
2 DO FOR EACH TOKEN UNTIL ; OR ESS IS REACHED
3 PERFORM SETUP FOR SPECIAL HANDLING FOR NUMBERS, NAMES, +, -, (, AND 
4 PERFORM STRING FOR SYNTAX CHECKING AND POLISH STRING BUILD
5 END DO
6 PERFORM RANGE TO PROCESS RANGE DEFINITION, IF NECESSARY
7 END XIPS1
BEGIN SETUP

CASE TOKEN

CASE SYMBOL:

CASE CHARACTER:

CASE PLUS:

CASE MINUS:

CASE ASTERISK:

CASE LPAREN:

CASE RPAREN:

CASE END:

BEGIN SETUP

CASE SYMBOL:

CASE CHARACTER:

CASE PLUS:

CASE MINUS:

CASE ASTERISK:

CASE LPAREN:

CASE RPAREN:

CASE END:

BEGIN SETUP

CASE SYMBOL:

CASE CHARACTER:

CASE PLUS:

CASE MINUS:

CASE ASTERISK:

CASE LPAREN:

CASE RPAREN:

CASE END:
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
INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK
ERREXIT TO :EXCEPTION: IF COUNT > COMMA LIMIT FOR (TOP ENTRY IN GROUPING)STK
IF COMMA LIMIT SHOWS SUBSCRIBING IN FUNCTION LIST 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 :BADOQL:
ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTAX2:
DISCARD TOKEN (2 OR LESS)
ENDIF
POP OPERATOR STACK
POP OPERATOR ONTO EXPRESSION STACK
ELSE BRACKETS OR PARENTHESES HAVE BEEN MATCHED
IF TOP ENTRY OF OPERATOR STACK IS ( THEN
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 :EXIT: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GROUPING STK COUNT=0)
ENDIF
POP OPERATOR STACK
POP GROUPING STACK
DISCARD CURRENT TOKEN
ENDIF
END IF
END DO
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 "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY"
:COMERR:
SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
1 BEGIN RANGE
2 INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
3 IF LAST TOKEN INPUT WAS ; THEN
4 DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
5 ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
6 CALL XSYM TO INSERT NAME IN SYMBOL TABLE, IF NECESSARY, FLAG ENTRY AS
7 AN INDEX, AND RETURN SYMBOL TABLE INDEX
8 ERREXIT IF NEXT TOKENS ARE NOT "INTEGER,INTEGER" TO :RNGSYM:
9 IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
10 SET INCREMENT TC -1
11 ENDF
12 PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
13 ENDF DO
14 ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
15 ENDF
16 EXIT RANGE
17 :RNGSYM:
18 SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ":NAME=
19 * INTEGER, INTEGER"
20 CALL XMSG TO OUTPUT MESSAGE TO USER
21 CALL ZILSS TO LIST SYMBOLIC STRING
22 CALL XEXIT TO EXIT PROCESSOR
23 END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

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

** INPUTS FROM ASGCOM **
LU, TOKENS, LASTY, SYMTAB, SSTRNG, RESULT, OPRNDS
OPINFO, NUMNDS, DATTPS, RSLTPT, EXPR, CLSREQ, CLSTRM, DECLAS,
RNGSTR, POLISH, SYNTAX, FMCTBL

** OUTPUTS TO ASGCOM **
SYMTAB, RESULT, RERST, RSLTPT, OPINFO

** INTERNAL VARIABLES **
ENTRY - ENTRY FROM POST-FIX STRING WHICH IS CURRENTLY BEING PROCESSED

** EXTERNAL REFERENCES **
FDS - XPROR, XPART, XRMOP, XICHR, XIFMC, XIFRE, XIFX0, XILSS, XIMS6,
XIOPR, XIPCS, XIRET

RTE - IABS, IAND, MAXR
BEGIN XIPS2

2465   1 BEGIN XIPS2
2466   2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
2467   3 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
2468   4 INITIALIZE RANGE VALUES FOR ITERATION
2469   5 DO UNTIL ALL RANGES ARE FINISHED
2470   6 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
2471   7 POP ENTRY FROM POLISH STRING
2472   8 IF ENTRY IS AN OPERAND, THEN
2473   9 PERFORM PUSHS TO RETRIEVE DATA IF AVAILABLE AND PUSH INTO RESULT STACK
2474  10 ELSE ENTRY IS AN OPERATOR
2475  11 IF OPERATOR IS NOT "=", THEN
2476  12 PERFORM EVALVAL TO EVALUATE POLISH STRING
2477  13 ELSE OPERATOR IS =
2478  14 PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
2479  15 ENDEF
2480  16 ENDF
2481  17 END DO
2482  18 END DO
2483  19 PERFORM RNGSET TO DETERMINE CORRECT RANGE VALUES
2484  20 END XIPS2
BEGIN TOC
DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
IF SYMBOL NOT = 0 (SYMBOL IS DE OR INDEX), THEN
IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT = 1), THEN
CALL XPREQ TO RETRIEVE TOC ENTRY
ELSE SYMBOL IS A RANGE INDEX
SET DATA TYPE TO INTEGER
ENDIF
ENDIF
END DO
CALL XPREQ WITH A CLOSE BUFFER REQUEST
END TOC
2499 1 BEGIN DATA1
2500 2 DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
2501 3 IF SYMBOL IS A NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENT, THEN
2502 4 CALL XPRESS TO QUEUE REQUEST FOR DATA RETRIEVAL
2503 5 ENDIF
2504 6 END DO
2505 7 CALL XPRESS WITH A CLOSE BUFFER REQUEST
2506 8 END DATA1
1 BEGIN
2 PUSH
3 IF operand is a character string pointer THEN
4 OPENUP < C1,254,354> RESULT STACK
5 ELSE
6 IF operand is a symbol table index THEN
7 CALL GET-ARG VALUE AND DATA TYPE
8 IF operand is a symbol table index THEN
9 ELSE push
10 END IF
11 END IF
12 END IF
13 END
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2517</td>
<td>BEGIN EVAL</td>
</tr>
<tr>
<td>2518</td>
<td>IF OPERATOR IS A FUNCTION, THEN</td>
</tr>
<tr>
<td>2519</td>
<td>DETERMINE &amp; OPERANDS FROM FUNCTION TABLE</td>
</tr>
<tr>
<td>2520</td>
<td>ELSE</td>
</tr>
<tr>
<td>2521</td>
<td>DETERMINE &amp; OPERANDS FROM SYNTAX TABLE</td>
</tr>
<tr>
<td>2522</td>
<td>ENDIF</td>
</tr>
<tr>
<td>2523</td>
<td>DO FOR EACH OPERAND</td>
</tr>
<tr>
<td>2524</td>
<td>IF RESULT STACK CONTAINS A SYMBOL INDEX, THEN</td>
</tr>
<tr>
<td>2525</td>
<td>EREREVE IF DATA TYPE IS FREE OR CHARACTER TO :ERRER:</td>
</tr>
<tr>
<td>2526</td>
<td>CALL XPES TO RETRIEVE DATA AND &quot;PE FOR ELEMENT OF ARRAY</td>
</tr>
<tr>
<td>2527</td>
<td>STORE DATA AND TYPE IN RESULT STACK AT THIS ENTRY</td>
</tr>
<tr>
<td>2528</td>
<td>ELSE</td>
</tr>
<tr>
<td>2529</td>
<td>EREREVE IF DATA TYPE IS FREE OR CHARACTER TO :ERRER:</td>
</tr>
<tr>
<td>2530</td>
<td>ENDIF</td>
</tr>
<tr>
<td>2531</td>
<td>END DO</td>
</tr>
<tr>
<td>2532</td>
<td>IF THE OPERATOR IS A FUNCTION, THEN</td>
</tr>
<tr>
<td>2533</td>
<td>DO FOR EACH FUNCTION OPERAND</td>
</tr>
<tr>
<td>2534</td>
<td>SET TARGET TYPE FROM FUNCTION TABLE</td>
</tr>
<tr>
<td>2535</td>
<td>CALL XIPCS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP</td>
</tr>
<tr>
<td>2536</td>
<td>END DO</td>
</tr>
<tr>
<td>2537</td>
<td>CALL XIPMC TO PERFORM FUNCTION FOR RESULTS</td>
</tr>
<tr>
<td>2538</td>
<td>SET DATA TYPE FOR RESULTS FROM FUNCTION TABLE</td>
</tr>
<tr>
<td>2539</td>
<td>ELSE THIS IS AN OPERATOR OTHER THAN A FUNCTION OR &quot;=&quot;</td>
</tr>
<tr>
<td>2540</td>
<td>DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE</td>
</tr>
<tr>
<td>2541</td>
<td>CASE (TYPE REQUIREMENT+1) :SAME:, :INTER:</td>
</tr>
<tr>
<td>2542</td>
<td>:SAME:</td>
</tr>
<tr>
<td>2543</td>
<td>IF # OPERANDS &gt; 1, THEN</td>
</tr>
<tr>
<td>2544</td>
<td>SET TARGET TYPE TO MAX OF TWO DATA TYPES</td>
</tr>
<tr>
<td>2545</td>
<td>ELSE # OPERANDS = 1</td>
</tr>
<tr>
<td>2546</td>
<td>SET TARGET TYPE TO OPERAND'S DATA TYPE</td>
</tr>
<tr>
<td>2547</td>
<td>ELSE</td>
</tr>
<tr>
<td>2548</td>
<td>:INTER:</td>
</tr>
<tr>
<td>2549</td>
<td>SET TARGET TYPE TO INTEGER</td>
</tr>
<tr>
<td>2550</td>
<td>END CASE</td>
</tr>
<tr>
<td>2551</td>
<td>END</td>
</tr>
<tr>
<td>2552</td>
<td>DO FOR EACH OPERAND</td>
</tr>
<tr>
<td>2553</td>
<td>CALL XIPCS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP</td>
</tr>
<tr>
<td>2554</td>
<td>END DO</td>
</tr>
<tr>
<td>2555</td>
<td>CALL XIPOR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK</td>
</tr>
<tr>
<td>2556</td>
<td>1 EXIT EVAL</td>
</tr>
<tr>
<td>2557</td>
<td>2 :ERRER:</td>
</tr>
<tr>
<td>2558</td>
<td>SET MESSAGE TO BE OUTPUT TO &quot;FREE OR CHARACTER DATA ELEMENT FOUND IN AN</td>
</tr>
<tr>
<td>2559</td>
<td>* EXPRESSION&quot;</td>
</tr>
<tr>
<td>2560</td>
<td>CALL XPMSG TO OUTPUT MESSAGE TO USER</td>
</tr>
<tr>
<td>2561</td>
<td>CALL XLSLS TO LIST SYMBOLIC STRING</td>
</tr>
<tr>
<td>2562</td>
<td>CALL XPRT TO EXIT PROCESSOR</td>
</tr>
<tr>
<td>2563</td>
<td>1 END EVAL</td>
</tr>
</tbody>
</table>
2565 1 BEGIN REPLAC
2566 2 IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
2567 3 IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
2568 4 CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
2569 5 STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
2570 6 ENDIF
2571 7 CASE (OBJECT DATA TYPE )
2572 8 FREE: ; :FIXED: ; :FIXED: ; :CHAR: ; :CHAR: ;
2573 9 FIX: ; ; :FIXED: ; :CHAR: ; :CHAR: ;
2574 10 :CHAR: ; :CHAR: ; :CHAR: ;
2575 11 :CHAR: ; :CHAR: ; :CHAR: ;
2576 12 :FREE: ;
2577 13 CALL XPREQ TO RETRIEVE DATA AND SET UP FOR STORE
2578 14 :FIXED:
2579 15 CALL FIXOBJ TO RETRIEVE DATA, CONVERSE IF NECESSARY, AND SET UP FOR STORE
2580 16 :CHAR:
2581 17 CALL CHROBJ TO RETRIEVE DATA AND SET UP FOR STORE
2582 18 END CASE
2583 19 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT, THEN
2584 20 POP OBJECT'S DISPLACEMENT FROM RESULT STACK
2585 21 ELSE OBJECT HAS NOT BEEN SUBSCRIPED
2586 22 SET OBJECT'S DISPLACEMENT TO 0
2587 23 ENDIF
2588 24 BUILD XPREQ REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S
2589 25 DISPLACEMENT
2590 26 CALL XPREQ TO STORE DATA IN OBJECT
2591 27 END REPLAC
2593 1 BEGIN RNGSET
2594 2  DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
2595 3   * RANGES ARE PROCESSED
2596 4   IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
2597 4   INCREMENT RANGE VALUE
2598 3   ELSE
2599 4   SET RANGE INDEX VALUE TO BEGIN VALUE
2600 3   ENDIF
2601 2   END DO
2602 1   END RNGSET
`FORTRAN CALLING PROCEDURE:
CALL XIRET(ENTRY)

XIRET IS USED BY THE ASSGN ROUTINE XIPS2 TO RETRIEVE THE DATA VALUE FOR A GIVEN SYMBOL INDEX FROM THE SYMBOL TABLE AND PUSH IT ONTO THE RESULT STACK ALONG WITH ITS DATA TYPE.

INPUTS
ENTRY - SYMBOL TABLE INDEX CURRENTLY BEING PROCESSED
ASGCON - SYMTAB, RSLTPT
OUTPUTS TO ASGCON
RESULT, RSLTPT
EXTERNAL REFERENCES
FDS - XRMOV
RTE - IAMD

END OF XIRET`
1 BEGIN XIRET
2  IF SYMBOL ENTRY IS FOR NON-SUBSCRIPTED FIXED-TYPE DATA (DATA ELEMENT OR
3  RANGE INDEX), THEN
4  SET RETURN VALUE = VALUE IN SYMBOL TABLE FOR THIS ENTRY
5  SET RETURN DATA TYPE = DATA TYPE IN SYMBOL TABLE FOR THIS ENTRY
6  ELSE  SYMBOL IS CHARACTER OR FREE DATA OR SUBSCRIPTED FIXED DATA
7  SET RETURN VALUE = SYMBOL TABLE INDEX
8  SET RETURN DATA TYPE = -1
9  ENDF
10  END XIRET
FORTAN CALLING PROCEDURE

CALL XISTM (TKNPTR, FLAG, SYMIND)

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

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

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

ASCOM COMMON - LU, SSTRING, STWIDE, SYMTAB, TOKENS,

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

ASCOM COMMON - SYMTAB

EXTERNAL REFERENCES
XPIIT, XILSS, XIMSG, XISYM
BEGIN XSYM
   CALL XSYM TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY
   IF TOKEN IS A NAME, THEN
      IF SPECIAL PROCESSING FLAG IS SET (1=RANGE INDEX, 2=SUBSCRIPTED), THEN
         IF FLAG INDICATES RANGE INDEX, THEN
            ENEXIT TO :BADRNG: WITH ERROR AS01 IF OBJECT (FIRST ENTRY IN TABLE)
         ELSE IF BADRNG: WITH ERROR AS02 ENTRY IS ALREADY SUBSCRIPTED
         ELSEIF BADRNG: WITH ERROR AS03 ENTRY IS ALREADY A DEFINED RANGE
      ENDIF
      ELSEIF SET ENTRY FLAG WORD TO FLAG VALUE
      ENDEF
   ENDF
   ENDF
   ADD BIAS OF 256 TO SYMBOL INDEX
2700 1 EXIT XSYM

2701 2 :BADRNG:
2702 2 CALL XMSG TO OUTPUT ERROR DESCRIPTION
2703 2 CALL XLLSS TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR
2704 2 CALL SQUIT TO TERMINATE PROCESSOR
2705 1 END XSYM

REPEATED CLARITY OF 1.
ORIGIN PAGE IS POOR.
2753 1 BEGIN XISTT
2754 2 CLEAR BUFFER TO BE USED IN MOVING TOKEN
2755 3 IF TOKEN IS A NAME, THEN
2756 4 SET COMPARISON DISPLACEMENT IN TABLE TO 1 (NAME FIELD)
2757 5 ELSE
2758 6 SET COMPARISON DISPLACEMENT IN TABLE TO 8 (VALUE FIELD)
2759 7 EMDIF
2760 8 MOVE TOKEN INTO BUFFER
2761 9 START SEARCH UNTIL ALL ALLOCATED SYMBOL TABLE ENTRIES EXAMINED
2762 10 EXIT IF ENTRY MATCHES BUFFER CONTENTS AND TYPE FIELD MATCHES TOKEN CODE
2763 11 END LOOP
2764 12 STORE TOKEN CODE IN TYPE FIELD OF NEXT ENTRY
2765 13 STORE BUFFER CONTENTS INTO APPROPRIATE FIELD OF ENTRY (NAME OR \VALUE)
2766 14 INCREMENT NUMBER OF ALLOCATED ENTRIES
2767 15 END SEARCH
2768 16 SET SYMIND TO ENTRY NUMBER
2769 1 END XISTT
(27-n) 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

Words 8-14: interface table entry for opnum1 (same as

in original table except for possible new

values pointing into literal area)

Words 15-21: interface table entry for opnum2 (see above)

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

opnum1 & 2

Word 26: relation id code as follows

0 - =
1 -> OR =
3 <= OR <=

Word 27: sequence number of top of loop

---

Internal variables

Code: Array of eight acceptable relation mnemonics and

corresponding internal code

Doent: Index into bostk for new 27 word entry

Prduff: 64 word manager communications buffer. eight word

tables of the form

Rerset: Ama management request code

Class: Class and type of data

Name: six character data name

Size: Size of data

Disp: Displacement into data of transaction origin

Clasho: Class i/o number through which data is

transmitted

Top: Index into bsetb of top of loop

Xpcls: Class i/o number for manager communications (set by

xpget)

---

Referenced routines

Exec, iand, mod, xpar, xpget, xpreg(xpget), xpixt, xrcpr, xmov,

Xrset, xudgs, xvpam, xirsg

Notes

Do and enddo must be used in pairs

Bostk is allocated in the ama for interprocessor communications

Zesch cleans up any residual bostk after execution controller

Termination

The maximum number of nested loops is 4

All nesting (structure) errors are left for execution time

Detection
124 1 BEGIN DO
125 2 CALL XGET TO INITIALIZE ACCESS TO XPRGR AND TO RETRIEVE DOTYPE AND RELATN
126 3 CALL XVPAM TO RETRIEVE EOSTK INTO BUFFER
127 4 IF RETRIEVAL FAILED, THEN
128 5 INITIALIZE BUFFER FOR BUILDING FIRST EOSTK ENTRY
129 6 ENDIF
130 7 IF EOSTK IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
131 8 IF XGET INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXIST, THEN
132 9 CALL XPRGR TO RETRIEVE LITERALS
133 10 EMDIF
134 11 IF RELATN IS A VALID RELATION OPERATOR, THEN
135 12 SET RELATION CODE IN NEW ENTRY IN BUFFER
136 13 SET INTERFACE TABLE HEADER WITH NAME OF EINTAB AND NUMBER OF PARAMETERs OF 2
137 14 DO FOR EACH OPAM
138 15 MOVE OPAM ENTRY INTO NEW INTERFACE TABLE BUFFER
139 16 IF OPAM HAS LITERAL VALUE OR DOUBLE SUBSCRIPTS, THEN
140 17 MOVE LITERAL DATA
141 18 ADJUST LITERAL POINTERS
142 19 EMDIF
143 20 EMDIF
144 21 CALL XPRGR TO RETRIEVE $SERTB (EXECUTING SEQUENCE TABLE)
145 22 IF $SERTB DISPLACEMENT ($SEQSP) > 0, THEN
146 23 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (0 IF END OF TABLE)
147 24 ELSE INSERTED COMMAND
148 25 EXIT TO :ERROR2: IF SEQUENCE NUMBER IS ZERO (MANUAL)
149 26 LOCATE ORIGINAL SEQUENCE ENTRY
150 27 IF ORIGINAL ENTRY WAS ALSO A DO (OVERRIDE CONDITION), THEN
151 28 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (OR ZERO)
152 29 ELSE (INSERT)
153 30 SET TOP OF LOOP TO CURRENT NUMBER
154 31 ENDIF
155 32 EMDIF
156 33 CASE (:WHILE, :UNTIL, :OTHER) DOTYPE
157 34 :WHILE:
158 35 INVERT RELATION CODE
159 36 INITIALIZE NEST COUNTER TO 1
160 37 START SEARCH FROM TOP OF LOOP ENTRY UNTIL ALL ENTRIES HAVE BEEN EXAMINED
161 38 IF COMMAND IS EMDO, THEN
162 39 DECREMENT NEST COUNTER
163 40 ELSE
164 41 IF COMMAND IS ANOTHER DO, THEN
165 42 INCREMENT NEST COUNTER
166 43 ENDIF
167 44 ENDIF
168 45 EXIT IF NEST COUNTER IS ZERO
169 46 SET RESET NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
170 47 END LOOP
171 48 EXIT TO :ERROR4: FOR NO MATCHING EMDO
172 49 END SEARCH
173 50 :UNTIL:
174 51 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
175 52 :OTHER:
176 53 TERMINATE WITH ERROR FOR UNRECOGNIZED DOTYPE
177 54 END CASE
178 55 CALL XPRGR TO OUTPUT NEW EXPANDED EOSTK
179 56 ELSE INVALID RELATION
CD********
CD0
1 CD0 FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
CD1
1 CD1 THE ELSE UTILITY PROCESSOR LOCATES THE ENDIF CORRESPONDING TO
CD2
1 CD2 THIS ELSE AND SETS THE SEQUENCE RESET NUMBER (VIA XPXIT) SO
CD3
1 CD3 THE PROCESSOR FOLLOWING ENDIF IS EXECUTED NEXT IN THE SEQUENCE.
CD4
1 CD4
CD5
1 CD5
CD6
1 CD6 INPUT
CD7
1 CD7 SCHEDULING PARAMETERS
CD8
1 CD8 LU - LOGICAL UNIT NO. OF THIS FDS USER
CD9
1 CD9 FLAGS - DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT
CD10
1 CD10
CD11
1 CD11 OUTPUT
CD12
1 CD12 RPARGS - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPXIT
CD13
1 CD13 (1) = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE
CD14
1 CD14 NO. GIVEN IN RPARGS(2)
CD15
1 CD15 (2) = 8 => ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE
CD16
1 CD16 (2) = SEQUENCE NO. TO BE EXECUTED NEXT WHEN RPARGS(1) = 3
CD17
1 CD17
CD18
1 CD18
CD19
1 CD19 REFERENCE Routines
CD20
1 CD20
CD21
1 CD21 RMPAR
CD22
1 CD22 XPXIT
CD23
1 CD23 XREX
CD24
1 CD24 XUDGR
CD25
1 CD25 XIFCL
CD26
1 CD26 XISCN
CD27
1 CD27
CD28
1 CD28
ENDDO - LOOP TERMINATION PROCESSOR
SCHEDULED BY FD

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

INPUT FROM MANAGER VIA SCHEDULING PARAMETERS
LU - LOGICAL UNIT OF USER'S TERMINAL
FLAGS - DEBUG FLAGS FROM USER SIGN-ON
INPUT/OUTPUT FROM/TO AWU
BDOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27,N) WHERE N IS THE CURRENT NUMBER OF ACTIVE LOOPS.
SEE DO PROCESSOR FOR CONTENTS
THE FD'S SYSTEM STATUS TABLE, XVSST, IS ACCESSSED FOR THE MANAGER/
PROCESSOR CLASS I/O COMMUNICATIONS NUMBER

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

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

REFERENCES ROUTINES
EXEC, IAND, IMPAR, XPGET, XPREG(XPGET), XPXIT, XRLDG, XRMV,
XUDBG, XVPFW, XIMSG
298 1 BEGIN ENDDO
299 2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XUSTB
300 2 CALL XUPAN TO RETRIEVE BDSTK
301 2 IF RETRIEVAL WAS SUCCESSFUL, THEN
302 3 WRITE BINTAB TO CLASS I/O FROM LAST BDSTK ENTRY
303 3 WRITE NON-LITERAL PORTION OF BINTAB TO CLASS I/O (LEAVE FOR XPGT)
304 3 CALL XUPAN TO RESTORE BINTAB FROM FIRST CLASS BUFFER INTO AWA
305 3 IF RESTORE SUCCESSFUL, THEN
306 4 CALL XPGT TO RETRIEVE OPRMDS
307 4 CALL XVEVL TO EVALUATE RELATION
308 4 IF RELATION IS TRUE, THEN
309 5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
310 5 IF THIS IS LAST BDSTK ENTRY, THEN
311 6 DELETE BDSTK FROM AWA
312 6 ELSE
313 7 CALL XPREQ TO REPLACE BDSTK LESS LAST ENTRY
314 7 ENDF
315 4 ELSE
316 5 SET RESET NUMBER TO TOP OF LOOP
317 5 ENDF
318 3 ELSE
319 4 TERMINATE WITH ERR06 FOR AWA OVERFLOW
320 4 ENDF
321 2 ELSE
322 3 TERMINATE WITH ERR03 FOR ENDDO WITH OUT DO
323 3 ENDF
324 1 ENDDO
```fortran
     366  1 CD*********
     367  1 CD0
     368  1 CD0 FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
     369  1 CD0
     370  1 CD0
     371  1 CD0
     372  1 CD0
     373  1 CD0 THE IF UTILITY PROCESSOR COMPARES TWO INPUT QUANTITIES ACCORDING TO A
     374  1 CD0 SPECIFIED RELATION. WHEN THE COMPARISON IS FALSE, THE SEQUENCE RESET
     375  1 CD0 NUMBER IS SET (VIA XPAR) TO THE PROCESSOR FOLLOWING THE CORRESPONDING
     376  1 CD0 ELSE (OR ENDIF). IF THE CONDITION IS TRUE THE NORMAL PROCESSOR
     377  1 CD0 EXECUTION SEQUENCE IS FOLLOWED.
     378  1 CD0
     379  1 CD0
     380  1 CD0
     381  1 CD0 INPUT
     382  1 CD0 SCHEDULING PARAMETERS
     383  1 CD0
     384  1 CD0
     385  1 CD0
     386  1 CD0 INTERFACE TABLE PARAMETERS
     387  1 CD0
     388  1 CD0
     389  1 CD0
     390  1 CD0
     391  1 CD0
     392  1 CD0
     393  1 CD0
     394  1 CD0 OUTPUT
     395  1 CD0
     396  1 CD0
     397  1 CD0
     398  1 CD0
     399  1 CD0
     400  1 CD0
     401  1 CD0
     402  1 CD0
     403  1 CD0
     404  1 CD0
     405  1 CD0
     406  1 CD0
     407  1 CD0
     408  1 CD0
     409  1 CD0
     410  1 CD0
     411  1 CD0
     412  1 CD0
     413  1 CD0
```
BEGIN IF
CALL RNPAR TO GET INPUT (SHEMATING) PARAMETERS
CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSLATE IT TO A CODE
ERREXIT IF RELATIONAL OPERATOR INVALID :ERROR:
CALL XIEVL TO EVALUATE THE RELATIONAL EXPRESSION
IF THE EXPRESSION IS FALSE, THEN
CALL XISSC TO LOCATE THE ELSE OR ENL: CORRESPONDING TO THIS IF
AND SET SEQUENCE RESET NUMBER
ELSE
CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE
ENDIF
EXIT IF
:ERROR: CALL XIMSG 'INVALID RELATIONAL OPERATOR - MUST BE $,>,>,>,<,<=,OR =>
END IF
**FORTRAN CALLING PROCEDURE**

LVALUE = XIEVL (OPRND1, RELAT, OPRND2)

**INPUT**

OPRND1 - FIRST REAL VALUE TO COMPARE

RELAT - RELATION CODE AS FOLLOWS

0 - NOT EQUAL
1 - GREATER THAN
2 - GREATER THAN OR EQUAL
3 - EQUAL
4 - LESS THAN OR EQUAL
5 - LESS THAN

OPRND2 - SECOND REAL VALUE TO COMPARE

**OUTPUT**

XIEVL - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

**INTERNAL VARIABLES**

TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS FOLLOWS

OPRND1 - OPRND2

<table>
<thead>
<tr>
<th>CODE RELATION + O -</th>
<th>OPRND1 - OPRND2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T F F</td>
</tr>
<tr>
<td>1</td>
<td>&gt; T F</td>
</tr>
<tr>
<td>2</td>
<td>&gt;= T T F</td>
</tr>
<tr>
<td>3</td>
<td>= T F</td>
</tr>
<tr>
<td>4</td>
<td>&lt;= T F F</td>
</tr>
<tr>
<td>5</td>
<td>&lt; T F</td>
</tr>
</tbody>
</table>

(BY COMPLEMENTING CODE > 2)

**REFERENCED Routines**

XEXT

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

**CALL ZISCH (SRCFG, PPARMS)**

**INPUTS**

**SRCFG** - FLAG INDICATING ORIGINATION OF THIS CALL
  
  = 0 => CALLED BY IF TO FIND MATCHING ELSE OR ENDF
  
  = 1 => CALLED BY ELSE TO FIND MATCHING ENDF

**OUTPUTS**

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

**SEESP**, INDEX INTO SEQUENCE TABLE (SERTAB) OF THE CURRENT COMMAND

**INPUTS**

**PARMS** - PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XFRIT

**SEQUENCE NO. TO BE EXECUTED NEXT IF PAPAMS(1) = 3**

**Routines Used**

1. EXEC
2. RMPAR
3. XVFN
4. XUSER
5. XCPBR
6. XRESX
7. XDRG
1 BEGIN XISCH
2 RETRIEVE &SETA3 FROM THE AWA USING XVPAM
3 STARTSEARCH UNTIL ALL COMMANDS IN &SEATAB
4 EXIT IF CURRENT COMMAND IS FOUND
5 SET NUMBER OF IF NESTS TO 1
6 STARTSEARCH FROM NEXT COMMAND IN &SEATAB UNTIL ALL FOLLOWING COMMANDS
7 IF COMMAND IS FOR ENDIF PROCESSOR, THEN
8 DECREMENT NUMBER OF IF NESTS BY 1
9 ELSE
10 IF COMMAND IS FOR IF PROCESSOR, THEN
11 INCREMENT NUMBER OF IF NESTS BY 1
12 ELSE
13 IF CALLED BY IF PROCESSOR, AND
14 COMMAND IS FOR ELSE PROCESSOR, THEN
15 ERREXIT IF THIS IS THE END OF &SEATAB :ERROR1:
16 IF NUMBER OF IF NESTS IS 1, THEN
17 DECREMENT NUMBER OF IF NESTS TO 0
18 ENDIF
19 ENDIF
20 EXIT IF NUMBER OF IF NESTS IS 0
21 SET SEQUENCE RESET NUMBER (RPARMS(2)) TO BE SEQUENCE NUMBER OF THE
22 NEXT COMMAND IN THE TABLE
23 ENDDO
24 ERREXIT :ERROR1:
25 ENDSEARCH
26 ENDDO COPI
27 ERREXIT :ERROR4:
28 ENDSERCH
29 1 EXIT XISCH
30 :ERROR1: CALL XISMSG - 'IF CANNOT "E EXECUTED WITHOUT MATCHING ENDIF'
31 :ERROR4: CALL X.ISMSG - 'SYSTEM ERROR - NO &SEATAB'
32 1 END XISCH
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO                     : 124</td>
</tr>
<tr>
<td>ELSE                   : 230</td>
</tr>
<tr>
<td>ENDDO                  : 298</td>
</tr>
<tr>
<td>ENDF                   : 361</td>
</tr>
<tr>
<td>ERROR1                 : 591</td>
</tr>
<tr>
<td>ERROR2                 : 156</td>
</tr>
<tr>
<td>ERROR3                 : 428</td>
</tr>
<tr>
<td>ERROR4                 : 187</td>
</tr>
<tr>
<td>IF                     : 415</td>
</tr>
<tr>
<td>:OTHER                 : 175</td>
</tr>
<tr>
<td>:UNTIL                 : 173</td>
</tr>
<tr>
<td>:WHILE                 : 137</td>
</tr>
<tr>
<td>XIEVL                  : 490</td>
</tr>
<tr>
<td>Y1                       : 494</td>
</tr>
<tr>
<td>:D                       : 493</td>
</tr>
</tbody>
</table>

Q FIN
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ #</th>
<th>SIZE-PRE TEXT (CYCLE WORD)</th>
<th>PSRNODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:35:40</td>
<td>1</td>
<td>55</td>
<td>3 0</td>
<td>1</td>
</tr>
<tr>
<td>FSTURD</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:35:07</td>
<td>2</td>
<td>30</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>SORT2</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:35:40</td>
<td>3</td>
<td>25</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>XUPDL</td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>03:37:22</td>
<td>4</td>
<td>84</td>
<td>2 1</td>
<td>2</td>
</tr>
<tr>
<td>PDLIST</td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>2 2</td>
<td>1</td>
</tr>
<tr>
<td>MAP</td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>5 1</td>
<td>2</td>
</tr>
<tr>
<td>POLIST</td>
<td>ABSOLUTE</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>280</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>15:47:05</td>
<td>8</td>
<td>109</td>
<td>5 6</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>15:47:05</td>
<td>9</td>
<td>394</td>
<td>5 2</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>00:40:28</td>
<td>10</td>
<td>55</td>
<td>6 12</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>195</td>
<td>8 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>12</td>
<td>324</td>
<td>9 5</td>
<td>5</td>
</tr>
<tr>
<td>XI</td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>13</td>
<td>627</td>
<td>14 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>14</td>
<td>320</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>COMM:N</td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>22:00:04</td>
<td>15</td>
<td>200</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>POLIST</td>
<td>RELOCATABLE</td>
<td>03 MAR 78</td>
<td>22:00:04</td>
<td>16</td>
<td>200</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>17 MAR 78</td>
<td>10:54:29</td>
<td>17</td>
<td>101</td>
<td>10 5</td>
<td>5</td>
</tr>
<tr>
<td>XP</td>
<td>ELT SYMB</td>
<td>17 MAR 78</td>
<td>10:54:29</td>
<td>18</td>
<td>534</td>
<td>22 5</td>
<td>5</td>
</tr>
<tr>
<td>XT</td>
<td>ELT SYMB</td>
<td>17 MAR 78</td>
<td>10:54:29</td>
<td>19</td>
<td>384</td>
<td>22 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>20</td>
<td>154</td>
<td>4 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>21</td>
<td>504</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>XR</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>22</td>
<td>403</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>23</td>
<td>504</td>
<td>20 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>24</td>
<td>151</td>
<td>5 5</td>
<td>5</td>
</tr>
<tr>
<td>XI</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>25</td>
<td>118</td>
<td>13 5</td>
<td>5</td>
</tr>
<tr>
<td>XD</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>26</td>
<td>1201</td>
<td>21 5</td>
<td>5</td>
</tr>
<tr>
<td>XN</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>27</td>
<td>600</td>
<td>29 5</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAR 78</td>
<td>12:00:25</td>
<td>28</td>
<td>326</td>
<td>2 3</td>
<td>3</td>
</tr>
</tbody>
</table>

NEXT AVAILABLE LOCATION:

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

EXIT F.POLIST