NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE
Flight Design System-1
System Design Document

Executive Logic Flow - Program Design Language

Mission Planning and Analysis Division
December 1979

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

FLIGHT DESIGN SYSTEM-I
SYSTEM DESIGN DOCUMENT

EXECUTIVE LOGIC FLOW - PROGRAM DESIGN LANGUAGE

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

JSC Task Monitor: Software Development Branch

Approved: Elric N. McHenry, Chief
Software Development Branch

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

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

The System Design Document (SDD) for FDS-1 is the specification for and description of this hardware/software facility. The SDD is logically organized into 10 published volumes. This organization is presented in the accompanying table. The material in the early volumes is primarily presented from the user's point of view, whereas the latter material is software-developer oriented. The SDD will be published by volumes over a period of time, and various volumes will be updated and republished during the development of FDS-1.
<table>
<thead>
<tr>
<th>Volume</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume I</td>
<td>Introduction, Overview, and User Interface</td>
</tr>
<tr>
<td>Volume II</td>
<td>Utility Processor Library</td>
</tr>
<tr>
<td>Volume III</td>
<td>FDS-1 Processor Library</td>
</tr>
<tr>
<td>Volume IV</td>
<td>System Architecture and Executive</td>
</tr>
<tr>
<td>Volume V</td>
<td>Data Management and Data Base Documentation Support System</td>
</tr>
<tr>
<td>Volume VI</td>
<td>Standards</td>
</tr>
<tr>
<td>Volume VII</td>
<td>Utility Support Software</td>
</tr>
<tr>
<td>Volume VIII</td>
<td>Build and Delivery Procedures, Software Development, Debug, and System Build Aids</td>
</tr>
<tr>
<td>Volume IX</td>
<td>Executive Logic Flow - Program Design Language</td>
</tr>
<tr>
<td>Volume X</td>
<td>Document Change Request Procedure and Submittal Form</td>
</tr>
</tbody>
</table>

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

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

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

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

TO FOS ROUTINES EXCEPT XR....

********

XE CONTENTS

INTEGER
# CERTIFG
# CLASNO
# COMPTL
# FLAGS
# PRCNAM(3)
# REBUF(64)
# SEREND
# STUBTD
# TKMLNG
# TOKENS(32)
# XE

DIMENSION
# INTNAM(3)

EQUIVALENCE
# XE(1), LU
# XE(3), DUAL
# XE(5), MAVSTA
# XE(7), SEGNAM(1)
# XE(11), SEGNAM(2)
# XE(13), INTNAM(1)
# XE(19), REBUF(2)
# XE(85), TKMLNG
# XE(139), EXEDEX
# XE(141), OLDIND
# XE(143), NOPROC
# XE(144), COMPTL
# XE(145), COMBUF(3)

CARTAG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

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

COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER

(1) - NUMBER OF TOKENS IN BUFFER
(2) - NUMBER OF USED WORDS IN BUFFER

(3-256) - TOKENS REPRESENTING USER'S RESPONSE

COMPTL - POINTER TO FIRST CURRENTLY BEING PROCESSED FROM COMBUF

EXED - SEQUENCE # WHERE EXECUTION IS TO END (RETURN TO X)

FLGCRE - EXECUTIVE FLAG WORD (0-OFF, I-ON)

BITS 0-10 UNUSED

11 PRINTER 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, SENI OR
AUTO-WITH-TRACE MODE (FIRST WORD = 0 IF DEFAULT INTERFACE
TABLE)

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

HASSTA - EXECUTIVE MASTER STATE FLAG (LEVEL LAST PASSED CONTROL BY
EXEC). 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 - TCE
2 - SAVE
3 - RECALL
4 - DELETE
5 - RENAME
6 - COPY
7 - CLEAR
8 - OFF
9 - ISTORE
10 - RESTORE
11 - UNLOAD
12 - LOAD
13 - BATCH

BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR

IF BITS 14-15 = 1

0 - INITIALIZATION FROM DIRECTIVE
1 - INITIALIZATION FOR REENTRY

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

0 - MANUAL
1 - SENI-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

HOPROC - NUMBER OF PROCESSES IN LIBRARY

OLDIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

PROC - NAME OF PROCESSOR FOR WHICH INTERFACE TABLE EDITOR WAS

INCLUDED OR BEING EXECUTED IN MANUAL, SENI OR AUTO-WITH-
TRACE MODE

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

REDBUF - BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SDD 6.2.7)

REDEPTH - POINT TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUFF

(R INDICATES REBUFF EMPTY) OR RETURN CODE FORM XREX

SELEND - TERMINATING SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED

IN SENI OR AUTO MODE AS PASSED TO THE MANAGER

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

EXECUTED IN SENI OR AUTO MODE

SEQPTS - POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SENI

SERSTA - INITIAL SEQUENCE NUMBER OF SEQUENCE TABLE EXECUTED IN

SENII 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.
IMME:

162  CO  XB CONTENTS (EXECUTION CONTROL LEVEL)
163  CO  INTEGER
164  CO  * ASCENT
165  CO  * RESIND
166  CO  * SENO
167  CO  DIMENSION
168  CO  * ASCENT(10)
169  CO  * SENO(150)
170  CO  * SENDER(150)
171  CO  EQUIVALENCE
172  CO  * (TB(2), LIIQ(1))
173  CO  * (TB(36), ASCENT(1))
174  CO  * (TB(249)), SENO)
175  CO  * (TB(250)), SENDER)
176  CO  ASCENT - SEQUENCE TABLE ENTRY IN ASCII TO PROMPT USER
177  CO  IN SEMI MODE ONLY
178  CO  CURIND - CURRENT INDEX TO EXECUTING SEQUENCE ENTRY
179  CO  LIBD - LIBRARY DIRECTORY PROCESSOR NAME TABLE
180  CO  NPPC2 - NUMBER OF PROCESSORS IN LIBD (SAME AS XE(143))
181  CO  RESIND - INDEX OF RESET ENTRY WHEN RESET SEQUENCE 0 IS REQUESTED
182  CO  SERLEN - LENGTH OF SEQUENCE TABLE
183  CO  SENO - NUMBER OF ENTRIES IN SEQUENCE TABLE
184  CO  SENDER - SEQUENCE TABLE CURRENTLY BEING EXECUTED
185  CO  --------
<table>
<thead>
<tr>
<th>Line</th>
<th>XC CONTENTS (INTERFACE TABLE EDIT LEVEL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>INTEGER BITNO BITM BITNUM DIRECT(6) DFLAG EQUAT(3) PRMTND SFLAG</td>
</tr>
<tr>
<td>252</td>
<td>ARGNO BITM DRGTH UML</td>
</tr>
<tr>
<td>253</td>
<td>VERSION DIMENSION ISIZE(9) NEWTAG(3)</td>
</tr>
<tr>
<td>254</td>
<td>EQUIVALENC (XB(23), COM Foley) (XB(25), LISTG) (XB(27), DEBUG)</td>
</tr>
<tr>
<td>255</td>
<td>(XB(29), MDSAV) (XB(30), IARG4)</td>
</tr>
<tr>
<td>256</td>
<td>(XB(31), IDIM) (XB(32), ISIZE)</td>
</tr>
<tr>
<td>257</td>
<td>(XB(33), SFLAG) (XB(34), ISUB)</td>
</tr>
<tr>
<td>258</td>
<td>(XB(35), CFLAG) (XB(36), DFLAG)</td>
</tr>
<tr>
<td>259</td>
<td>(XB(37), ARGPH) (XB(38), ARGPTR)</td>
</tr>
<tr>
<td>260</td>
<td>(XB(39), ISUB) (XB(40), IOFLAG)</td>
</tr>
<tr>
<td>261</td>
<td>(XB(41), PRMTND) (XB(42), PMLLEN)</td>
</tr>
<tr>
<td>262</td>
<td>(XB(43), PROMPT(1))</td>
</tr>
<tr>
<td>263</td>
<td>(XB(44), LEagneF)</td>
</tr>
<tr>
<td>264</td>
<td>(XB(45), ITEMP) (XB(46), BITSO)</td>
</tr>
<tr>
<td>265</td>
<td>(XB(47), MOBiT) (XB(48), LITPS)</td>
</tr>
<tr>
<td>266</td>
<td>(XB(49), LITIONS) (XB(50), LENG)</td>
</tr>
<tr>
<td>267</td>
<td>(XB(51), LITL) (XB(52), LITR)</td>
</tr>
<tr>
<td>268</td>
<td>(XB(53), LITV) (XB(54), IRET)</td>
</tr>
<tr>
<td>269</td>
<td>(XB(55), IARG) (XB(56), NUMARG)</td>
</tr>
<tr>
<td>270</td>
<td>(XB(57), NEWTAG) (XB(10), XBUF)</td>
</tr>
<tr>
<td>271</td>
<td>(XB(101), XBUF)</td>
</tr>
<tr>
<td>272</td>
<td>ARGNO - CURRENT ARGUMENT'S NO. (I.E. 1 TO 64)</td>
</tr>
<tr>
<td>273</td>
<td>ARGPTR - INDEX IN XBUF TO ARGUMENT PROMPT FOR CURRENT ARGUMENT</td>
</tr>
<tr>
<td>274</td>
<td>BITNO - BIT NO. IN BIT MASK CURRENTLY BEING PROCESSED</td>
</tr>
<tr>
<td>275</td>
<td>BITM - BIT NO. IN BIT MASK OF NEXT DIFFERING BIT</td>
</tr>
<tr>
<td>276</td>
<td>COMFLE - FLAG SET TO 1 IF A CONTINUATION PROMPT OF CURRENT ARGUMENT</td>
</tr>
<tr>
<td>277</td>
<td>HAS BEEN EXPLICITLY REQUESTED VIA A TRAILING COMA OR IS</td>
</tr>
<tr>
<td>278</td>
<td>IMPLICITLY REQUIRED BECAUSE INCOMPLETE ELEMENTS REMAIN BEYOND</td>
</tr>
<tr>
<td>279</td>
<td>LAST ENTERED ELEMENT OF THIS ARGUMENT</td>
</tr>
<tr>
<td>280</td>
<td>DEBUG - FLAG SET TO VALUE OF DEBUG FLAG (BITS 13-15 OF XE(4))</td>
</tr>
<tr>
<td>281</td>
<td>DIRECT - ASCII ARRAY OF VALID INTERFACE TABLE EDIT DIRECTIVES</td>
</tr>
<tr>
<td>282</td>
<td>DEBUG - FLAG SET TO VALUE OF DATA EDIT FOR THIS ARGUMENT</td>
</tr>
<tr>
<td>283</td>
<td>ARG - INDEX IN XBUF TO CURRENT ARG</td>
</tr>
<tr>
<td>284</td>
<td>ICLASS - CLASS OF THIS ARGUMENT</td>
</tr>
<tr>
<td>285</td>
<td>IOFLAG - FLAG SET TO I/O BITS FOR THIS ARGUMENT</td>
</tr>
<tr>
<td>286</td>
<td>IRETC - GENERAL RETURN FLAG (0=OK, -1=ERROR, 5='EXIT')</td>
</tr>
<tr>
<td>287</td>
<td>ISIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT</td>
</tr>
<tr>
<td>288</td>
<td>ISIZE - NO. OF WORDS/ELEMENT FOR EACH ALLOWED ITYPE VALUE</td>
</tr>
<tr>
<td>289</td>
<td>ITYPE - CURRENT EFFECTIVE SUBSCRIPT FOR THIS ARGUMENT</td>
</tr>
</tbody>
</table>
1 i' JSUB F - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
1 CD LENEFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
1 CD LISTLU - LU TO WHICH PRINT SHOULD GO (USED WHEN
1 CD LIST DIRECTIVE CALLS XILSD OR XICHRS)
1 CD LITDSP - DISPL. TO LITERAL DATA FOR THIS ARGUMENT
1 CD LITDW - INDEX IN WKBUF TO END OF LITERAL DATA
1 CD LITLEN - LENGTH OF LITERAL DATA AREA OF WKBUF
1 CD LITPRT - 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 XICHRS.
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, PARAMETER 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 PRMTMD WHILE PRMTMD = 4 (CONTINUE)
1 CD MARG - INDEX IN WKBUF TO START OF SHORT PROMPTS
1 CD MDTBM - INDEX IN WKBUF TO BIT MASK WORD(S) FOR THIS ARGUMENT
1 CD WENAT - ASCII NAME OF TABLE BEING GENERATED
1 CD WODIM - NO. OF BIT MASTS 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 PRMTMD - CURRENT PROMPTING MODE
1 CD = 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
1 CD = 2, PROMPT WITH A LIST OF CURRENT VALUES
1 CD = 3, CONTINUATION OF SAME ARGUMENT
1 CD = 4, CONTINUATION OF SAME ARGUMENT
1 CD = 5, PROMPT WITH "/" (CREATE MODE)
1 CD PRTOMP - 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 WKBFLG - LENGTH OF WKBUF
1 CD WBKBUF - WORKING BUFFER FOR INTERFACE TABLE BEING EDITED
1 CD ORGANIZED AS :
1 CD HEADER (7 WORDS)
1 CD ARGUMENT CHARACTERISTICS (NUMARG+7 WORDS)
1 CD ARGUMENT PROMPTS (NUMARG+3 WORDS)
1 CD LITERAL DATA AND PARAMETER SUBSCRIPTS (LITLEN WORDS)
1 CD
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
FDS ERROR MESSAGES

ATTENTION FUNCTION

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

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

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

**XA04** FDS ATTENTION FUNCTION TERMINATING

**XA05** FDS MANAGER SIGNALED TO TERMINATE BOTTOM ASSOC TASK

**XA06** FDS EXECUTIVE ACTIVE - NO ACTION TAKEN

**XQ04** FDS PROCESSOR 'NAME' SCHEDULED TO ABORT

**XQ05** MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN

**XQ06** BATCH JOB CREATION

**XQ07** CONFIGURATION PROGRAMS

**XQ08** LU 'NN' SIGNED ON TO FDS

**XQ10** LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED

**XQ12** -LU 'LU' IS AN INVALID LU

**XQ14** FDS CURRENTLY AT MAX USER'S, CANNOT SIGN ON

**XQ16** 'LU' IS ALREADY SIGNED ON TO FDS

**XQ18** CANNOT FIND 'NAME' ID - SIGN ON TERMINATED

**XQ19** ENTER VALID USER ID (A-Z)

**XQ21** MANAGER

**XQ23** INVALID REQUEST II FROM 'NAME'

**XQ25** FDS SIGN OFF FOR LU 'LU'

**XQ27** INVALID BACK CHAIN FOR CURRENT PROCESSOR

**XQ29** 'H' TRACKS NOT AVAILABLE FOR DWA

**XQ31** NO DWA SPACE FOR 'NAME'

**XQ33** NO AWA FOR DWA DIRECTORY

**XQ35** I/O ERROR FOR DWA, STATUS = "NNNNNN"

**XQ37** TOC SEARCH ERROR, PHASE 3 COMPRESS

**XP** PROCESSOR SERVICES

**XP01** TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED

**XP02** DATA AREA OVERFLOW

**XP03** SUBSCRIPT OUT OF RANGE

**XP04** INVALID ENTRY AFTER SUBSCRIPT

**XP05** INVALID REPEAT ENTRY

**XP06** TOO MANY NESTED REPEATS

**XP07** INVALID SUBSCRIPT

**XP08** RESPONSE IS TOO LONG FOR BUFFER

**XP09** INVALID RESPONSE ON OR AFTER COLUMN NN

**XP10** PROCESSOR INITIALIZATION ERROR

**XP11** AWA ACCES FAILURE FOR ...

**XP12** PARAMETER I/O INCONSISTENCY

**XP13** INVALID PARAMETER REQUEST

**XP14** INVALID OR DFE FILE ATTRIBUTE(S)

**XP15** RETRIEVAL OF TOO MUCH DATA REQUESTED

**XV** SYSTEM SERVICES

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

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

**XV03** 'ID-NAME' NAMLO PROGRAM HAS INVALID BACK CHAIN TO MANAGER
4.0  PDL LISTING PROGRAM

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

INPUT

80 COLUMN PDL IMAGES SUBJECT TO THE FOLLOWING CONVENTIONS:

- In Column 1 indicates Page Eject and the first token (6 or less
characters) is reproduced in columns 127-132 of output listing

UNTIL NEXT RECORD IS DETECTED. IF RECORD CONTAINS ONLY THE 

TOKEN FROM THE PREVIOUS RECORD CONTINUES TO APPEAR IN THE

IDENTIFICATION COLUMNS OF THE OUTPUT

* In Column 1 indicates a comment to be copied to the output listing.

OTHER IN COLUMN 1 indicates PDL record to be structured based on key

WORDS APPEARING AS FIRST NON-BLANK CHARACTERS (SEE LOGIC).

OUTPUT

132 COLUMN LISTING AS FOLLOWS

2-6 SEQUENCE NUMBER

8-11 STRUCTURE LEVEL NUMBER

13-125 STRUCTURED LISTING OF 80 COLUMN INPUT RECORDS

FOLLOWING THE LISTING IS A SYMBOL DEFINITION TABLE INDICATING THE

SEQUENCE NUMBER OF THE LINE CONTAINING EACH "BEGIN NAME" AND

"LABEL:"

NOTES

USES FSTWAD & SORT1
1 BEGIN PDLIST
2 INITIALIZE SEQUENCE NUMBER, LEVEL AND DEFINITION TABLE COUNTER
3 DO UNTIL END-OF-FILE INPUT
4 READ RECORD
5 INCREMENT SEQUENCE NUMBER
6 IF COLUMN 1 = 
7 THEN
8  SET PAGE EJECT IN IMAGE
9  IF REMAINDER OF IMAGE IS NOT BLANK
10   THEN
11   SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
12   ENDF
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 FSTUND TO GET FIRST WORD OF PDL
23   LOOKUP FIRST WORD IN KEY WORD TABLE
24          - KEY WORD TABLE CONTAINS
25          1 BEGIN       - BEGIN SECTION INDICATOR
26          2 IF          - SECTION INDICATOR
27          3 DO          - SECTION INDICATOR
28          4 DFOR         - SECTION INDICATOR
29          5 DOUNTI       - SECTION INDICATOR
30          6 DOWNMIL     - SECTION INDICATOR
31          7 CASE         - SECTION INDICATOR
32          8 START        - SECTION INDICATOR
33          5 STARTS       - SECTION INDICATOR
34          6 ELSE         - SECTION SEPARATOR
35          7 THEN         - SECTION SEPARATOR
36          8 END          - END OR END LOOP?
37          17 END          - END OF LOOP?
38          18 ENDF         - SECTION TERMINATOR
39          19 ENUDO        - SECTION TERMINATOR
40          20 ENOCAS       - SECTION TERMINATOR
41          21 ENDOSEA      - SECTION TERMINATOR
42          80 ENDSE
43 IF KEY WORD LOCATED
44 THEN
45 CASE LOCATION :BEGIN; :SECON; :SECON; :SECON; :SECON; :SECON;
46 (:SECON; :SECON; :SECON; :SECON; :SEP; :SEP; :SEP; :SEP; :SEP; :SEP;
47 (:SEP; :TERM; :TERM; :TERM; :TERM; :TERM; :TERM; :TERM; :TERM;
48 86 8
49 BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
50 SET LEVEL INCREMENT = 1
51 87 8
52 :SECON: SET LEVEL INCREMENT = 1
53 PDLIST
:SEP:  SET LEVEL INCREMENT = 1
       DECREMENT LEVEL
:END?:  CALL FSTWRD TO GET NEXT WORD OF PDL
       DECREMENT LEVEL
       IF WORD = LOOP
       THEN
       SET LEVEL INCREMENT = 1
       ENDIF
:TERM:  DECREMENT LEVEL
ENDCASE
ENDIF
ELSE
COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
SET INDENTATION FACTOR = 1
ENDIF
CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
INPUT RECORD AND ID FIELD
OUTPUT IMAGE
APPLY LEVEL INCREMENT
CLEAR PAGE EJECT
ENDIF
ENDDO
CALL SORT1 TO ORDER DEFINITION TABLE
OUTPUT DEFINITION TABLE
END POLIST
EXTRACT THE FIRST TOKEN FROM A PDL RECORD

116 1  CD1
117 1  CD1
118 1  CD1
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  *  
130 1  *  
131 1 *
132 1* 
133 1 BEGIN FSTWRD 
134 2 BLANK OUTPUT WORD 
135 2 LOCATE FIRST NON-BLANK CHARACTER 
136 2 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 3 ELSE 
141 3 EXIT DO 
142 3 EDI 
143 2 END IF 
144 2 RETURN LOCATION 
145 1 END FSTWRD
5.0 FDS EXECUTIVE DETAILED LOGIC FLOW

A directory listing the major programs and subroutines in alphabetical order is presented initially. The detailed logic flow of each then follows in alphabetical order.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td><strong>XBIT</strong> MULTIPLE WORD BIT STRING BIT CLEAR/SET</td>
</tr>
<tr>
<td>121</td>
<td><strong>XRCPR</strong> COMPARE ARRAYS</td>
</tr>
<tr>
<td>122</td>
<td><strong>XR18</strong> DOUBLE PRECISION TO ASCII CONVERSION</td>
</tr>
<tr>
<td>123</td>
<td><strong>XREG</strong> ASCII MANAGEMENT REQUEST ROUTINE</td>
</tr>
<tr>
<td>124</td>
<td><strong>XREX</strong> EXTRACTS A VARIABLE LENGTH FIELD FROM A WORD</td>
</tr>
<tr>
<td>125</td>
<td><strong>XRE14</strong> CONVERT A WORD TO ASCII IN 14 OFFSET FORM</td>
</tr>
<tr>
<td>126</td>
<td><strong>XRI6</strong> BINARY INTEGER TO ASCII CONVERSION ROUTINE</td>
</tr>
<tr>
<td>127</td>
<td><strong>XRLC</strong> XSTB RM LOCK</td>
</tr>
<tr>
<td>128</td>
<td><strong>XRLDC</strong> RETURN 16-BIT ADDRESS OF ARGUMENT</td>
</tr>
<tr>
<td>129</td>
<td><strong>XMOV</strong> MOVES WORDS FROM ARRAY1 TO ARRAY2</td>
</tr>
<tr>
<td>130</td>
<td><strong>XMS6</strong> FDS EXECUTIVE MESSAGE ROUTINE</td>
</tr>
<tr>
<td>131</td>
<td><strong>XRX6</strong> MULTIPLE WORD BIT STRING BIT SEARCH</td>
</tr>
<tr>
<td>132</td>
<td><strong>XBD</strong> CONVERT A WORD TO ASCII IN 04 FORMAT</td>
</tr>
<tr>
<td>133</td>
<td><strong>XRPC</strong> PACKS CHARACTERS FROM R1 TO A2 FORMAT</td>
</tr>
<tr>
<td>134</td>
<td><strong>XR6</strong> FILE NAME QUALIFICATION</td>
</tr>
<tr>
<td>135</td>
<td><strong>XR6E</strong> SETS A VARIABLE LENGTH FIELD INTO A WORD</td>
</tr>
<tr>
<td>136</td>
<td><strong>XRE</strong> SHIFT A WORD LEFT LOGICALLY</td>
</tr>
<tr>
<td>137</td>
<td><strong>XR6R</strong> SHIFT A WORD RIGHT LOGICALLY (XRSFL EP)</td>
</tr>
<tr>
<td>138</td>
<td><strong>XRLK</strong> XSTB RM UNLOCK (XRCLK EP)</td>
</tr>
<tr>
<td>139</td>
<td><strong>XRUI</strong> FILE NAME QUALIFICATION</td>
</tr>
<tr>
<td>140</td>
<td><strong>XPUPK</strong> REMOVES BLANKS AND UNPACKS FROM A2 TO R1 FORMAT</td>
</tr>
<tr>
<td>141</td>
<td><strong>XRS</strong> REMOVE DUPLICATE BLANKS FROM A2 STRING</td>
</tr>
<tr>
<td>142</td>
<td><strong>XS</strong> SEQUENCE TABLE EDITOR</td>
</tr>
<tr>
<td>143</td>
<td><strong>XSCAN</strong> SEQUENCE TABLE EDITOR (STE) DIRECTIVE SCANNER</td>
</tr>
<tr>
<td>144</td>
<td><strong>XDE</strong> DELETE DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>145</td>
<td><strong>XEN</strong> ENTRYS PROCESSOR</td>
</tr>
<tr>
<td>146</td>
<td><strong>XEN</strong> SEQUENCE TABLE EDITOR (STE) DIRECTIVE SCANNER</td>
</tr>
<tr>
<td>147</td>
<td><strong>XEN</strong> SEQUENCE TABLE EDITOR MAIN ROUTINE</td>
</tr>
<tr>
<td>148</td>
<td><strong>XLRS</strong> LIST DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>149</td>
<td><strong>XLS</strong> SEQUENCE TABLE LIST ROUTINE</td>
</tr>
<tr>
<td>150</td>
<td><strong>XMP</strong> INPUT PROCESSOR</td>
</tr>
<tr>
<td>151</td>
<td><strong>XJMP</strong> NUMBER DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>152</td>
<td><strong>XJPC</strong> STE TABLE COMPACTER</td>
</tr>
<tr>
<td>153</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>154</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>155</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>156</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>157</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>158</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>159</td>
<td><strong>XJPT</strong> STE PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>160</td>
<td><strong>XT</strong> TERMINAL Communications</td>
</tr>
<tr>
<td>161</td>
<td><strong>XCOM</strong> PROMPTS USER, READS RESPONSE, CALLS XTLAN AND XTPRM</td>
</tr>
<tr>
<td>162</td>
<td><strong>XTLN</strong> CONVERTS ASCII USER'S RESPONSE TO TOKENS</td>
</tr>
<tr>
<td>163</td>
<td><strong>XTPM</strong> HANDLES EXTENDED PROMPTING REQUESTS</td>
</tr>
<tr>
<td>164</td>
<td><strong>XU</strong> UTILITY (SOFTWARE AIDS)</td>
</tr>
<tr>
<td>165</td>
<td><strong>XUG</strong> ON-LINE SNAP AND MEMORY MODIFICATION ROUTINE</td>
</tr>
<tr>
<td>166</td>
<td><strong>XUMP</strong> SYSTEM RESIDENT PARTITION DUMP (XVARN EP)</td>
</tr>
<tr>
<td>167</td>
<td><strong>XUPF</strong> FILE MANAGER FILE DUMP PROGRAM</td>
</tr>
<tr>
<td>168</td>
<td><strong>XUPL</strong> OCTAL AND ASCII DUMP LINE FORMAT</td>
</tr>
<tr>
<td>169</td>
<td><strong>XUINF</strong> DUMP FORMATTER</td>
</tr>
<tr>
<td>170</td>
<td><strong>XV</strong> SYSTEM SERVICES</td>
</tr>
<tr>
<td>171</td>
<td><strong>XVABN</strong> FDS ABEND (SEE XHDMP)</td>
</tr>
<tr>
<td>172</td>
<td><strong>XVPAW</strong> FDS COMMUNICATIONS SERVICES (POST AND WAIT)</td>
</tr>
<tr>
<td>173</td>
<td><strong>XX</strong> EXECUTION CONTROL</td>
</tr>
<tr>
<td>174</td>
<td><strong>XXAU</strong> AUTOMATIC MODE</td>
</tr>
<tr>
<td>175</td>
<td><strong>XXNT</strong> EXECUTION CONTROL MAIN PROGRAM</td>
</tr>
<tr>
<td>176</td>
<td><strong>XXDC</strong> DECIDES USER RESPONSE IN MNU AND SENI</td>
</tr>
<tr>
<td>177</td>
<td><strong>XXDF</strong> READS IN DEFAULT INTERFACE TABLE IF NEEDED</td>
</tr>
<tr>
<td>178</td>
<td><strong>XXER</strong> EXECUTES ASSEMBLE AND HANDLES ERROR CONDITIONS</td>
</tr>
<tr>
<td>179</td>
<td><strong>XXMAN</strong> MANUAL MODE</td>
</tr>
</tbody>
</table>
179 1  XXSNH  SERN - AUTOMATIC NODE  DIRECT
180 1  XXSTO  STORE SEQUENCE TABLE IN BERTAB  DIRECT
181 1  XXTIM  TEMPORARY EXECUTION OF ONE ENTRY WITH BERTAB  DIRECT
182 1  UTILITY PROCESSORS  DIRECT
183 1  ASSGN  ASSIGN PROCESSOR  DIRECT
184 1  DBDSP  DATA BOX DISPLAY PROCESSOR  DIRECT
185 1  DEFIN  DEFINE PROCESSOR  DIRECT
186 1  DO  CONDITIONAL LOOP IN SEQUENCE TABLE  DIRECT
187 1  ELSE  EXECUTION POINT FOR FALSE IF CONDITION  DIRECT
188 1  ENDF  TERMINATES AN IF STRUCTURE  DIRECT
189 1  ENDFD  TERMINATES A DO LOOP STRUCTURE  DIRECT
190 1  ENDS C END SCAN PROCESSOR  DIRECT
191 1  IF  CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES  DIRECT
192 1  SCAN  SCAN PROCESSOR  DIRECT
193 1  XICHR  CHARACTER OBJECT STORE FOR ASSGN  DIRECT
194 1  XIDFT  FIND ANY TOKEN IN A SYMBOLIC STRING  DIRECT
195 1  XIDIN  DATA BOX DISPLAY INPUT PROCESSOR  DIRECT
196 1  XIDMK  DATA BOX DISPLAY CONSTRAINT MASKER  DIRECT
197 1  XIDOT  DATA BOX DISPLAY OUTPUT ROUTINE  DIRECT
198 1  XIDP1  DATA BOX DISPLAY PASS 1 PROCESSOR  DIRECT
199 1  XIDP2  DATA BOX DISPLAY PASS 2 PROCESSOR  DIRECT
200 1  XEEVL  PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS  DIRECT
201 1  XEFCL  FIND PROCESSOR CLASS NUMBER  DIRECT
202 1  XEFMC  FUNCTIONAL OPERATIONS FOR ASSGN  DIRECT
203 1  XFFRE  FREE OBJECT STORE FOR ASSGN  DIRECT
204 1  XFFRD  FIXED OBJECT STORE FOR ASSGN  DIRECT
205 1  XZISP  REMOVE DUPL. BLANKS & BLANK FILL  DIRECT
206 1  XZLSS  SYMBOLIC STRING SYNTAX ERROR LISTER  DIRECT
207 1  XZMG  FRU PROCESSOR MESSAGE ROUTINE  DIRECT
208 1  XZPKR  MATH OPERATIONS FOR ASSGN  DIRECT
209 1  XZPCS  DATA CONVERSION AND STORAGE FOR ASSGN  DIRECT
210 1  XZPS1  PASS 1 SUBROUTINE FOR ASSGN PROCESSOR  DIRECT
211 1  XZPS2  PASS 2 SUBROUTINE FOR ASSGN PROCESSOR  DIRECT
212 1  XZRET  DATA RETRIEVAL FOR ASSGN  DIRECT
213 1  XZSCH  SEARCHES SEQUENCE TABLE FOR IF STRUCTURES  DIRECT
214 1  XZSYM  SYMBOL TABLE INTERFACE FOR ASSGN  DIRECT
215 1  XZSTT  SYMBOL TABLE MAINTENANCE  DIRECT
216 1  *
217 1  *
SAVE EGR ADDRESS(IN BREG ON ENTRY)
CALL ENLU(BREG) GET LU IN ASCII & BINARY
STARTSEARCH UNTIL LAST STATUS TABLE ENTRY
EXITIF STBU EN LU
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STREG)
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 SLIRST DISABLE
IF STBX(EXECUTIVE ADDRESS).NE. 0, THEN
IF STAT(CURRENT).NE. STBX, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XHCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STREG(MANGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARM DME(P1)
CALL SLIRD ENABLE
CALL NMRAL 'OFF,BOTTOM'
CALL SLIRG DISABLE
SET BOTTOM TO NEXT-AT
ENDO
ENDIF
ENDIF
CALL SLIST MAKE EXEC DORMANT
CLEAR EXEC'S ID & STER
ENDIF
DECREMENT NUMBER ACTIVE(STBA)
GET EGR ADDRESS
RESTORE INTERRUPT HANDLER(FROM STER)
CLEAR STER
CLEAR MANAGER'S ID, STER & STBL
ENABLE... (VIA A JMP TO EEC(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 XASTAT
ENDIF
ENDIF
ELSE
WRITE '***XAO= ERROR LU IS NOT SIGNED-ON TO FDS'
ENDIF
WRITE '***XAO= BDS ATTENTION FUNCTION TERMINATING'
END
179 1 BEGIN XABTN    FIND BOTTOM AT
180 2 DO WHILE CURRENT IS IN GENERAL WAIT,
181 3 AND WAIT POINTER(P) HAS A SON ADDRESS,
182 3 AND SONS FATHER ID POINTS TO CURRENT
183 3 SET SON AS CURRENT
184 2 ENDPO
185 2 SET BOTTOM AS CURRENT
186 1 END XABTN
BEGIN XCNFG

* 1 FDS CONFIGURATION MANAGER
  2 *D1 INITIATES AN FDS SYSTEM FOR A REQUESTED TERMINAL OR
  3 *D1 TERMINATES AN FDS SYSTEM FOR A REQUESTED TERMINAL
  4 *D1 INITIATED VIA
  5 *D1 RU,FDS,LU_ID,CNA SIZE,PARMS,OPTIONS
  6 *D2 INPUTS
  7 *D2 LOGICAL UNIT(LU) FOR THE REQUESTED TERMINAL,
  8 *D2 A PARM TO DENOTE INITIATION OR TERMINATION
  9 *D2 A ONE CHARACTER USERID
 10 *D2 A DEBUG OPTION INDICATOR
 11 *D2 THE NUMBER OF DMA TRACKS
 12 *D3 OUTPUTS
 13 *D3 INITIATION-
 14 *D3 A BLANK ID-SEGMENT WILL BE CONSTRUCTED FOR THE FDS MANAGER,
 15 *D3 AND FDS EXECUTIVE
 16 *D3 THE ENT FOR THE REQUESTED LU WILL BE CONNECTED
 17 *D3 TO THE FDS ATTENTION TASK
 18 *D3 THE FDS TABLE(FDSAB) IN RESIDENT LIBRARY ROUTINE
 19 *D3 WILL BE UPDATED TO REFLECT THE INITIATION
 20 *D3 TERMINATION-
 21 *D3 THE IDS SEGMENTS WILL BE RETURNED TO BLANK STATUS
 22 *D3 THE ENT FOR THE LU WILL BE REINSTATED
 23 *D3 THE FDS TABLE(FDSAB) IN RESIDENT LIBRARY ROUTINE
 24 *D3 WILL BE UPDATED FOR THE TERMINATION
29  2  * PARMS=LU,ID,DWA SIZE,PARM(ON OR OFF),OPTIONS
30  3  * SWITCH INPUT PARM AROUND SO THAT
31  4  * NOW PARMS=LU,P2(ON OR OFF),ID,OPTS,DWA SIZE
32  5  * FOR COMPATABILITY TO BUILD 1.
33  6  CALL RMPAR(PARMS)
34  7  2  IF LU IS .LT. 0, OR
35  8  3  .GT. LUMP(1553), OR
36  9  3  .LE. 6(PRINTER), OR
37  10  3  THE DRIVER IS .NE. 0 OR 5, THEN
38  11  3  ISSUE MESSAGE "**XCONF 'LU' IS AN INVALID LU"
39  12  ELSE
40  13  2  IF PARM P2 IS OFF THEN
41  14  3  PERFORM XCONF SIGN OFF
42  15  3  ELSE
43  16  3  PERFORM XCONF SIGN ON
44  17  2  ENDIF
45  18  2  ENDIF
46  2  :XCEXT
47  2  CALL EXEC PROGRAM TERMINATION
48  1  END XCONF
1 BEGIN XCOM
2  * SIGN ON A USER TO FDS
3    DO UNTIL VALID USER ID (P3)
4      IF ID NOT A - 2, THEN
5          WRITE 'XCOM ENTER VALID ID (A - Z)'
6          HEAD RESPONSE
7          ENDF
8    ENDO
9    IF USER ID IS BEING USED, THEN
10       WRITE '***XCOM LU 'LU' IS CURRENTLY USING ID 'ID'- SIGN ON REJECTED'
11    EXIT :XCETA
12    ENDF
13    IF FDS RESOURCE NUMBER NOT DEFINED, THEN
14       CALL RNAG (GLOBAL ALLOCATE, LOCAL SET)
15    ELSE
16       CALL RNAG (LOCAL SET)
17    ENDF
18    IF NUMBER SIGNED ON (STBAR) .EQ. MAXIMUM USERS (STNUM), THEN
19       ISSUE MESSAGE '***XCOM FDS CURRENTLY AT MAX USER'S.
20    ELSE
21       DO FOR STNUM (NUMBER OF FDS ENTRIES)
22          IF ENTRY'S LU (STBULU) .EQ. REQUESTING LU(P2) THEN
23             ISSUE MESSAGE '***XCOM LU 'LU' IS ALREADY SIGNED ON TO FDS'
24             EXIT :XCETA
25          ELSE
26             IF THIS ENTRY IS AVAILABLE, THEN
27                SET AS CURRENT-ENTRY-ADDRESS
28                ENDF
29          ENDIF
30      ENDO
31    ENDO
32    BECOME PRIVILEGED & DISABLED
33    CALL SLIR
34    STARTSEARCH WHILE NUMBER-FOUND .LT. NUMBER-NEEDED
35    SEARCH ID-SEGMENTS USING KEYB(1657)
36      IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
37          SET ID ADDRESS OF XEXEC
38          INCREMENT NUMBER-FOUND
39    ELSE
40      IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
41          SET ID ADDRESS OF XEXEC
42          INCREMENT NUMBER-FOUND
43    ELSE
44      IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
45          SET ID ADDRESS OF FIRST-BLANK
46          INCREMENT NUMBER-FOUND
47    ELSE
48      IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
49          SET ID ADDRESS OF SECOND-BLANK
50          INCREMENT COUNT
51    ENDF
52    ENDF
53    ENDF
54    WRITE 'EXITS THERE ARE NO MORE IDS'
100 4 CALL BLNX  ENABLE
101 4 ISSUE MESSAGES "**XCO? CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
102 4 ENDL00
103 4 * BUILD FNTRY IN FNSTB
104 4 * SET LU INQ STBLU
105 4 * SET LU IN ASCII INTO STBLA
106 4 * SET USER'S ID INTO STBR
107 4 * SET ADDRESS OF FIRST-BLANK INTO STBD
108 4 * SET ADDRESS OF SECOND-BLANK INTO STBE
109 4 * INCREMENT ACTIVE COUNT(STBAC)
110 4 * BUILD XPGNN & XEXN
111 4 * MOVE PRIORITY THRU DISC ADDRESS FROM XMGF TO FIRST-BLANK
112 4 * TURN ON TH BIT
113 4 * SET NAME TO XPENN
114 4 * MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
115 4 * TURN ON TH BIT
116 4 * SET NAME TO XEXN
117 4 * LINK ATTENTION FUNCTION TO THE USER
118 4 * DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
119 4 * CALCULATE ERT OVERLAY
120 4 * SAVE ERT VALUE IN STBR
121 4 * SET ID ADDRESS OF XRTN INTO ERT
122 4 * ENDF
123 4 * SET INPUT PARMS INTO ID OF XMGNN
124 4 * SCHEDULE XMGNN VIA BLIST
125 4 * CALL BLIST
126 4 * CALL BLNX  ENABLE
127 4 * IF FPS HAS A FATHER, THEN
128 4 * CALL MESS 'OFF,FATHER'
129 4 * ENDF
130 4 * ISSUE MESSAGE "**XCO? LU "MN" SIGNED ON TO FPS"
131 4 ENDSRCH
132 3 X:CTA
133 3 CALL RMNR (LOCAL CLEAR)
134 3 ENDF
135 2 CALL EXEC TERMINATE
136 1 END XCON
FORTAN CALLING PROCEDURE

CALL XDLD (XDCLD)

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

LIST

INPUT

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

INTERNAL VARIABLES

LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

NOTES

USES .ENTR, XDLS, XERTM

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

BEGIN XDCLD

EXTRACT DIRECTIVE INDEX FROM MASSTA

CASE (LIST: INDEX)

LIST: CALL XDLS

ENDCASE

CALL XERTM TO RETURN FROM SEGMENT

END XDCLD
40 1 #00 FORTRAN CALLING PROCEDURE
41 1 #00 CALL XELDS (XDFCL)
42 1 #00
43 1 #00
44 1 #00 XDFCL DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
45 1 #00 REQUESTED AND Calls THE APPROPRIATE HANDLER SUBROUTINE
46 1 #00 TGC
47 1 #00 TOC
48 1 #00 SAVE
49 1 #00 RECALL
50 1 #00 DELETE
51 1 #00 RENAME
52 1 #00 COPY
53 1 #00 CLEAR
54 1 #00 OFF
55 1 #00
56 1 #00
57 1 #02 INPUT
58 1 #02 XE COMMON - MASSTA (BITS 10-13 CONTAIN A 1-8 INDEX INTO A LIST OF
59 1 #02 DIRECTIVES)
60 1 #02
61 1 #00
62 1 #04 INTERNAL VARIABLES
63 1 #04 LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
64 1 #04
65 1 #00
66 1 #05 NOTES
67 1 #05 USES :ENTR, XDELE, XDPOP, XDELE, XOFF, XDREC, XDREM, XDSEY.
68 1 #05 XTOC, XETRN
69 1 #05 XDCLF IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
70 1 #05 CONTAINING THE REFERENCED DIRECTIVES
71 1 #05
72 1 #00
73 1 
74 1 
75 1 
76 1 
77 1 BEGIN XDCLF
78 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
80 3 :TOC: CALL XTOC
81 3 :SAVE: CALL XDSEY
82 3 :RECA: CALL XDREC
83 3 :DELE: CALL XDELE
84 3 :REMA: CALL XDREM
85 3 :COPY: CALL XDPOP
86 3 :CLEA: CALL XDCLF
87 3 :OFF: CALL XDFF
88 2 ENDCASE
89 2 CALL XETRN TO RETURN FROM SEGMENT
90 1 END XDCLF
* Fortran Calling Procedure *

** XDCLU **

1. **CALL XELDS (XDCLU)**

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

- **STORE**
- **RESTORE**
- **UNLOAD**
- **LOAD**
- **BATCH**

** XDCLU input: **

- **X COMMON - MASSTA (Bits 10-13 contain a 9-13 index into a list of directives)**

** XDCLU internal variables: **

- **XDCLU is designed to be the main routine for the overlay segment containing the referenced directives.**

** XDCLU extract directive index from MASSTA and decrement by 7. **

** XDCLU case:**

- **:STOR: CALL XSTOR**
- **:REST: CALL XRES**
- **:UNLO: CALL XDUNL**
- **:LOAD: CALL XDLOA**
- **:BATCH: CALL XBAT**
- **ENDCASE**
- **CALL XERN TO RETURN FROM SEGMENT**

** XDCLU end XDCLU **
FORTRAN CALLING PROCEDURE

CALL XDCLF

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

**Input:**
- XE COMMON - CARTES, FLAGS, LU
- MANAGER - AWA HEADER AND TOC (SEE XMANA)

**Output:**
- XE COMMON - REBUF, REOPTN, COMBUF-SCRATCH
- XB COMMON - SCRATCH BEYOND XB(200)
- XS COMMON - SCRATCH
- MANAGER - COMMANDS TO PURGE AWA AND RESTORE CHAINS 0 AND 8

**Local variables:**
- AWA - AWA HEADER AND TOC RECEIVED FROM MANAGER
- HD1 - TOC CHAIN 1 HEAD
- HD2 - TOC CHAIN 2 HEAD
- HD3 - TOC CHAIN 3 HEAD
- HD4 - TOC CHAIN 4 HEAD
- NEXT - INDEX TO NEXT ALLOCATABLE EIGHT WORD ENTRY IN QUEUE
- NMAX - MAXIMUM SIZE OF QUEUE
- ORG - BASE REFERENCE ADDRESS FOR AWA ADDRESSES
- QUEUE - BUFFER FOR CONSTRUCTING AWA RESTORE REQUESTS FOR NON-DELETED ITEMS
- TOCMAX - MAXIMUM SIZE OF TOC ACCOMODABLE BY XDCLF

**Notes:**
- USES EXEC, IAND, PRTN, PURGE, XREQ, XRIS, XRNV, XRMSG, XR06, XRWF, XUDRG, XVABN
- IN THE EVENT THAT A COMPACTED AWA TOC WILL NOT FIT IN THE ALLOCATED SPACE AND ELEMENTS OF CHAIN 1 OR 8 EXTEND BEYOND IT, THE CLEAR FUNCTION WILL NOT BE PERFORMED.
- SHOULD THE RESTORATION OF CHAIN 1 OR 8 FAIL AFTER THE CLEAR
- REQUEST HAS BEEN COMPLETED BY THE MANAGER, FBS WILL BE TERMINATED
- IN ORDER TO PROTECT USER DISK FILES.
188 1 BEGIN XDCEL
189 2 RETRIEVE TOC
190 2 BUILD REQUEST TO CLEAR AWAY
191 2 DO UNTIL END OF PERMANENT SYSTEM TABLES CHAIN (CHAIN 1)
192 2 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
193 2 EXIT TO :ERR48: IF RESTORATION REQUEST QUEUE IS FULL
194 3 BUILD REQUEST TO REALLOCATE TABLE
195 3 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
196 3 BUILD REQUEST TO STORE TABLE INTO AWAY FROM SAM
197 2 ENDDO
198 2 DO UNTIL END OF DATABASE FILES CHAIN (CHAIN 8)
199 2 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
200 2 EXIT TO :ERR48: IF RESTORATION REQUEST QUEUE IS FULL
201 2 BUILD REQUEST TO REALLOCATE TOC ENTRY
202 2 ENDDO
203 2 BUILD REQUEST TO TERMINATE LIST
204 2 DO UNTIL END OF DRDE CHAIN (CHAIN 3)
205 3 IF CHAIN POINTS BEYOND TOC BUFFER
206 3 THEN
207 4 OUTPUT XD13 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
208 4 EXIT PURGE LOOP
209 3 ENDIF
210 3 PURGE FILE
211 2 ENDDO
212 2 DO FOR EACH BLOCK OF EIGHT REQUESTS
213 2 TRANSMIT BLOCK TO MANAGER
214 2 EXIT TO :ERR23: IF REQUESTS FAILED
215 2 ENDDO
216 1 EXIT XDCEL

217 2 :ERR23:
218 2 DO FROM FAILING REQUEST TO END OF LIST
219 3 IF REQUEST TO STORE
220 4 THEN
221 5 READ SAM TO FREE BUFFER AND CLASS NUMBER
222 5 ENDIF
223 2 ENDDO
224 2 DO UNTIL END OF DATABASE FILE CHAIN (CHAIN 8)
225 3 IF FILE IS UTDB (TYPE 1)
226 3 THEN
227 4 CALL PURGE TO DELETE FILE
228 4 ENDIF
229 2 ENDDO
230 2 PURGE ALL UTDB FILES
231 2 TERMINATE FDS WITH CLEAR FAILURE MESSAGE

232 2 :ERR48:
233 2 :ERR24:
234 2 DO FOR ALL STORE REQUESTS BUILT
235 2 READ SAM TO FREE BUFFER AND CLASS NUMBER
236 2 ENDIF
237 1 EXIT XDCEL WITH CLEAR FAILURE MESSAGE
238 1 END XDCEL
**FORTRAN CALLING PROCEDURE**

**CALL XDCOP**

**XDCOP PROCESSES A USER REQUEST TO COPY A SEQUENCE TABLE,**
**INTERFACE TABLE, DATA ELEMENT, O R D E R, U T O B, OR FOR. ONLY***
**THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S AMA CAN BE COPIED.**

**INPUT**

**COMMON XE - CARTAG, COMBUF, QUA L, REPTR, TOKENS**

**OUTPUT**

**COMMON XE - COMPR TR, REBUF**

**INTERNAL VARIABLES**

**COMMON XS - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE)**

** (2) DTYP E: TYPE CODE FOR DATA BASE FILES**

** (3) I: INDEX**

** (4) IDCB: EXEC BUFFER**

** (148) IFRR: FILE MANAGER ERROR RETURN**

** (149) IMSG: ERROR MESSAGE NUMBERS**

** (150) NAMEN E: NEW FMGR FILE NAME**

** (153) NAMPTR: POINTER TO NEW NAME IN COMBUF**

** (154) OFNAME: OLD FMGR FILE NAME**

** (157) OMPTR: POINTER TO OLD NAME IN COMBUF**

** (158) TOSCN: TOC ENTRY FOR DATA BASE RENAME**

** (166) EOF: EOF RETURN FROM FMGR**

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

** (124A) FTYP E: FMGR FILE TYPE**

** (1245) FAM1: FIRST BYTE OF REPTR**

** (1246) FAM2: SECOND BYTE OF REPTR**

** (1247) PREFIX: PREFIX FOR FILE NAME**

** (1248) SCRIT Y: FILE SECURITY CODE (0 OR 88)**

** (1249) SIZE: FILE SIZE IN BLOCKS**

** (1251) IDCBY: EXEC BUFFER**

**ROUTINES USED -**

**CLOSE, CREATE, EXEC, OPEN, PURGE, READF, WRITF, XDOBA, XDOBO, XREE, XRMV, XRMG, XRFM, XR SFR**

************
1 BEGIN XDCOP
2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
3 SAVE POINTER TO CURRENT NAME
4 IF NEXT TOKEN IS HYPHEN, THEN
5 DECODE CLASS NAME
6 EXIT TO :CLASS: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,B)
7 ELSE
8 SET CLASS TO DATA ELEMENT
9 ENDIF
10 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
11 SAVE POINTER TO NEW NAME
12 EXIT TO :SYNTAX: IF NEXT TOKEN IS END-OF-MESSAGE
13 IF CLASS IS DATA BASE OR ORD, THEN
14 EXIT IF NEW NAME IS MORE THAN 4 CHARACTERS ::NAMERR:
15 IF CLASS IS DATA BASE, THEN
16 EXIT TO :NAMERR: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
17 ENDIF
18 BUILD AVA MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
19 CALL XDCP
20 EXIT TO :TOCERR: IF RETURN INDICATES ERROR ON FIRST REQUEST
21 CALL EXEC TO GET TOC ENTRY
22 IF CLASS IS DATA BASE THEN
23 IF TYPE IS POB, THEN
24 CALL XDBA TO ADD NEW POB TO LOG FILE (XDBA)
25 EXIT TO :FILE: IF FMGR ERROR RETURNED
26 EXIT TO :TOCERR: IF NEW NAME IS DUPLICATE
27 EXIT TO :MAXERR: IF POB MAX IS EXCEEDED
28 SET FILE PREFIX TO RIGHT BRACKET
29 ELSE
30 EXIT TO :INVLD: IF FILE IS NOT N
31 SET FILE PREFIX TO *
32 ENDIF
33 SET FMGR FILE TYPE TO 1
34 SET SECURITY CODE TO 0
35 ELSE
36 SET FILE PREFIX TO /
37 SET SECURITY CODE = 0
38 SET FMGR FILE TYPE FROM TOC ENTRY
39 IF FILE IS TYPE 2, THEN
40 STORE RECORD LENGTH FROM TOC ENTRY
41 ENDIF
42 CALL XRCF
43 CALL XRCF TO FORMAT NEW FILE NAME
44 CALL XRCF TO CREATE NEW FILE
45 EXIT TO :FILE: IF FMGR ERROR RETURNED
46 CALL XRCF TO FORMAT OLD FILE NAME
47 CALL XRCF TO CREATE OLD FILE
48 EXIT TO :FILE: IF FMGR ERROR RETURNED
49 EXIT TO :TYPERR: IF TYPE IS NOT SAME AS IN TOC
50 DO UNTIL END-OF-FILE IS READ ON OLD FILE
51 CALL READ TO READ RECORD FROM OLD FILE
52 EXIT IF ERROR DETECTED TO :FILE:
53 CALL WRITE TO WRITE RECORD TO NEW FILE
54 EXIT IF ERROR DETECTED TO :FILE:
55 END DO
56 CALL CLOSE FOR OLD FILE
57 CALL CLOSE FOR NEW FILE
BUILD AND ISSUE AWA MANAGER REQUEST TO ALLOCATE TOC ENTRY

EXIT TO :TOCERR: IF ERROR IS INDICATED
ELSE
BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
BUILD MANAGER REQUEST FOR DATA RETRIEVAL
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
CALL EXEC TO GET TOC ENTRY
BUILD MANAGER REQUEST TO ALLOCATE NEW TABLE
BUILD MANAGER REQUEST TO STORE TABLE
IF CLASS IS INTERFACE TABLE, THEN
CALL EXEC TO WRITE NEW TABLE NAME TO SAM
BUILD MANAGER REQUEST TO STORE NEW NAME IN TABLE
ENDIF
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
ENDIF
1 EXIT XDCOP

:SYNTAX: CALL XRMGS -"SYNTAX ERROR ..." AND EXIT
:CLASER: CALL XRMGS -"INVALID CLASS DESIGNATOR ..." AND EXIT
:NAMEERR: CALL XRMGS -"NEW NAME IS INVALID ..." AND EXIT
:MAEXER: CALL XRMGS -"AUTHORIZED LIMIT ..." AND EXIT
:VALID: CALL XRMGS -"NOB CANNOT BE ..." AND EXIT
:FILER: CALL XRMGS -"FILE ACCESS ERROR ..." AND EXIT TO :END:
:TOCERR: CALL XRMGS TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
:TYPEERR: CALL XRMGS -"INCONSISTENT FILE TYPE ..."

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

CALL XDBD(NAME, IERR, ISIZE)

XDBDA, XDBOV, AND XDBD PROVIDE THE ABILITY TO ADD AN MDB OR
 MDB 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

NAME - 2 WORD INTEGER ARRAY CONTAINING THE UNQUALIFIED ASCII
 MDB/PDB NAME

ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
 XDBDA ONLY)

IERR - INTEGER WORD CONTAINING RETURN CODE

=0 NO ERRORS

=1 DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE

=2 MDB/PDB LIMIT EXCEEDED

=3 ISIZE INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
 XDBOV ONLY)

COMMON USED

XE - RUAL

CARTAG

XS - DCO (WORDS 1-144)

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

NOTES

XRULK, XRLCK, .ENTR, OPEN, CLOSE, READF, AND WRITF ARE USED
WHEN WORKING WITH MDB'S, QUAL SHOULD BE SET TO 77 OCTAL
1 BEGIN XDOBA
2  DO RETURN ADDRESS
3   CALL :ENTR TO SET UP CALLING ARGUMENTS
4   CALL XLCK FOR EXCLUSIVE USE OF XPD
5   CALL OPEN FOR EXCLUSIVE USE OF XPD
6   ELSE IF ERRR: IF ERROR RETURNED
7   EXIT TO :FILE:
8   EXIT TO :FILE:
9   EXIT TO :FILE:
10  DETERMINE PART OF RECORD TO BE USED
11  IF REQUEST IS FOR ADD, THEN
12  EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
13  START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
14  EXIT TO :NAME:
15  EXIT TO :NAME:
16  EXIT TO :NAME:
17  EXIT TO :NAME:
18  EXIT TO :NAME:
19  IF REQUEST IS FOR VERIFY, THEN
20  STORE WORD 3 OF ENTRY IN ISIZE
21  ELSE
22  REPLACE ENTRY WITH LAST ENTRY
23  STORE IDRES IN LAST ENTRY
24  DECREMENT # OF CURRENT ENTRIES
25  ENDIF
26  ENDIF
27  CALL WRIT TO WRITE RECORD TO XPD
28  EXIT TO :RETURN;
29  ELSE IF ERRR: SET IERR = 1 AND EXIT TO :RETURN;
30  ELSE IF ERRR: SET IERR = 2 AND EXIT TO :RETURN;
31  EXIT TO :FILE:
32  EXIT TO :FILE:
33  EXIT TO :FILE:
34  EXIT TO :FILE:
35  EXIT TO :FILE:
36  EXIT TO :FILE:
37  EXIT TO :FILE:
38  EXIT TO :FILE:
39  EXIT TO :FILE:
40  EXIT TO :FILE:
41  END XDOBA
1 BEGIN XDELETE
2 DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED
3 IF CLASS DESIGNATOR IS SPECIFIED, THEN
4 SET REQUESTED CLASS APPROPRIATELY (D, S, I, O, OR F)
5 ELSE
6 SET REQUESTED CLASS TO BE (O)
7 ENDIF
8 IF DATA BASE TO BE DELETED, THEN
9 BUILD AND ISSUE DMA MANAGER REQUEST FOR TOC ENTRY
10 ENDIF
11 IF ELEMENT IS NOT A MASTER DATA BASE, THEN
12 IF ELEMENT IS A PERSONAL DATA BASE, THEN
13 CALL EDROD TO DELETE THIS PDB FROM XPDO
14 IF ERROR IS RETURNED, THEN
15 CALL XRMG - "FILE ACCESS ERROR 0... XPDO"
16 SET ERROR FLAG
17 ENDIF
18 ENDIF
19 BUILD AND ISSUE DMA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
20 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
21 CALL XRMG - "XXXX NOT FOUND"
22 SET ERROR FLAG
23 ENDIF
24 IF CLASS IS DATA BASE (C), OR
25 CLASS IS DROD (F), THEN
26 IF ERROR FLAG IS NOT SET, THEN
27 CALL XRMG TO CONSTRUCT FILE NAME
28 ISSUE RTE PURGE FOR THE FILE
29 IF RETURN CODE FROM PURGE, THEN
30 CALL XRMG - "FMTA ERROR XXX XXXXXX"
31 ENDIF
32 ENDIF
33 ELSE
34 CALL XRMG - "... IS A MBD. NOT DELETED."
35 ENDIF
36 ENDDO
37 1 EXIT TO :RETURN;
38 2 :ERROR: CALL XRMG - "SYNTAX ERROR"
39 2 :RETURN;
40 1 END XDELETE
CALL XMIS (TClSt)

XMIS PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECORING EACH ELEMENT AND FLAGGING IT IN TClSt AS AN ELEMENT
TO BE STORED/RESTORED.

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XB(151) ABLG, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

TClSt - (INTEGER, 1200 WORDS) LIST OF TCl ENTRIES; THOSE
THAT ARE TO BE STORED/RESTORED ARE FLAGGED.

OUTPUTS IN COMMON:

XB(151) ABLG, XB(157) TTSTZ, XB(158) TTWPD

INTERNAL XB COMMON USED:

XB(151) ABLG - (INTEGER, 1 WORD) ABORT FLAG
XB(152) SRFLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TTSTZ - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
BE STORED/RESTORED
XB(158) TTWPD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
BE STORED/RESTORED
XB(198) REDT - (INTEGER, 1 WORD) XCOM RETURN CODE
XB(199) BATCLS - (INTEGER, 1 WORD) CLASS OF DATA BEING SEARCHED
FOR IN LIST OF TCl ENTRIES
XB(200) NAMPT - (INTEGER, 1 WORD) POINTER TO NAME IN COMBUF
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TClST

COMMON USED:

EQUIVALENCE
(1) LU
(48) Eos
(85) Name
(113) Comma
(144) Compto
(145) Combuf
647 1 BEGIN XDIS
648 2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
649 3 TURN ERRFLG OFF
650 3 :RTM1:
651 3 DO UNTIL ERR IS SENSED IN COMBUS
652 4 ERREXIT IF TOKEN IS NOT "NAME" TO :ERR1:
653 4 SAVE INDEX TO NAME FIELD
654 4 INCREMENT TO NEXT TOKEN
655 4 IF TOKEN IS A HYPHEN THEN
656 5 ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:
657 5 INCREMENT TO NEXT TOKEN
658 5 DECODE CLASS NAME (I, S, D, F)
659 5 ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:
660 5 SET CLASS TO CLASS SPECIFIED
661 4 ELSE
662 5 SET CLASS TO DATA ELEMENT
663 4 ENDF
664 4 IF XDIS CALLED FROM STORE THEN
665 5 ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:
666 4 ENDF
667 4 ERREXIT IF NAME/CLASS ENTRY NOT FOUND IN TOC TO :ERR2:
668 4 CALL ARSET TO TURN STORE/RESTORE BIT ON
669 4 INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
670 3 ENDO
671 3 :RTN2:
672 3 IF ERROR FLAG IS ON THEN
673 4 CALL XCOM TO REPROMPT USER TO CONTINUE
674 4 ERREXIT IF RESPONSE IS X TO :ERR3:
675 4 ENDF
676 2 ENDO
677 1 EXIT XDIS
678 2 :ERR1:
679 2 SET ERROR FLAG ON
680 2 CALL XRMSG TO DISPLAY SYNTAX ERROR
681 2 GO TO :RTM2:
682 2 :ERR2:
683 2 IF ERROR FLAG IS OFF THEN
684 3 TURN ERROR FLAG ON
685 2 CALL XRMSG TO DISPLAY NOT STORED/RESTORED MESSAGE
686 2 ENDF
687 2 CALL EXEC TO DISPLAY ELEMENT NAME
688 2 GO TO :RTM1:
689 2 :ERR3:
690 2 SET ABFLG TO ABORT STORE/RESTORE OPERATION
691 1 END XDIS
736 1 BEGIN XDLST
737 2   IF DEVICE ID FIELD SPECIFIED, THEN
738 3      SET LU FOR LISTING AS INDICATED ON INJECTIVE
739 4   ELSE
740 5      SET LU FOR LISTING TO BE TERMINAL LU
741 6   ENDIF
742 7   DO UNTIL END-OF-STATEMENT IS REACHED
743 8      ERREXIT IF COMMA IS NOT SPECIFIED :ERROR9:
744 9      ERREXIT IF A NAME DOES NOT FOLLOW THE COLON :ERROR9:
745 10     RETAIN NAME FOR XREX CALL
746 11     IF A CLASS DESIGNATOR IS SPECIFIED, THEN
747 12        SET CLASS (I, S, OR D) FOR XREX CALL
748 13     ELSE
749 14        USE DATA (0) CLASS IN XREX CALL
750 15     ENDIF
751 16     CALL XREX TO RETRIEVE THIS TABLE OR DATA ELEMENT
752 17      ERREXIT IF NOT FOUND :ERROR9:
753 18      ERREXIT IF AWA SPACE NOT AVAILABLE FOR TABLE IN DNA :ERR11:
754 19     CALL EXEC TO PERFORM CLASS READ OF DATA OR TABLE INTO
755 20     BOTTOM OF WORKING BUFFER
756 21     IF INTERFACE TABLE TO BE LISTED, THEN
757 22        CALL XRMOV TO MOVE CHARACTERISTICS TO TOP OF WORKING BUFFER
758 23        READ SHORT PROMPTS FOR THIS PROCESSOR INTO WORKING BUFFER
759 24        CALL XEINT TO INITIALIZE LITERAL ENTRIES
760 25        INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE ITS LIST RTN.
761 26        INITIALIZE 'LSTFLG' TO INDICATE ENTIRE TABLE TO BE LISTED
762 27     CALL XILST TO LIST THE INTERFACE TABLE
763 28     ELSE
764 29      IF SEQUENCE TABLE TO BE LISTED, THEN
765 30        CALL XRMOV TO MOVE TABLE TO TOP OF WORKING BUFFER
766 31        CALL XILST TO LIST SEQUENCE TABLE
767 32     ELSE
768 33        INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE ITS LIST RTN.
769 34        SET 'LSTFLG' TO INDICATE ONLY 1 DATA ELEMENT BEING LISTED
770 35        INITIALIZE PRINT BUFFER WITH NAME OF ELEMENT
771 36        CALL XILST TO LIST THE DATA
772 37     ENDIF
773 38     ENDIF
774 39     INCREMENT TO NEXT TOKEN IN OPERAND LIST OF THE DIRECTIVE IMAGE
775 40     :ERROR10: CALL XRMSG -- '... NOT FOUND'
776 41     :ERROR11: CALL XRMSP -- '... CANNOT BE MOVE FROM DNA TO AWA'
777 42     ENDODO
778 43     1 EXIT TO :RETURN:
779 44     2 :ERROR9: CALL XRMSP -- 'SYNTAX ERROR'
780 45     2 :RETURN:
781 46     1 END XDLST
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>783</td>
<td>1 CD</td>
<td>FORTRAN CALLING PROCEDURE</td>
</tr>
<tr>
<td>784</td>
<td>1 CD0</td>
<td>CALL XDOFF</td>
</tr>
<tr>
<td>785</td>
<td>1 CD0</td>
<td>XDOFF confirms the user's request for termination.</td>
</tr>
<tr>
<td>786</td>
<td>1 CD0</td>
<td>Deletes all XREX and XTOB files logged in the ANA.</td>
</tr>
<tr>
<td>787</td>
<td>1 CD0</td>
<td>Performs abnormal termination, if indicated, or</td>
</tr>
<tr>
<td>788</td>
<td>1 CD0</td>
<td>Returns normal paths to the FDS Manager and terminates.</td>
</tr>
<tr>
<td>789</td>
<td>1 CD0</td>
<td>Normally via RTE.</td>
</tr>
<tr>
<td>790</td>
<td>1 CD0</td>
<td>INPUT</td>
</tr>
<tr>
<td>791</td>
<td>1 CD2</td>
<td>COMMON XE = LU, flags, qual, REBUF</td>
</tr>
<tr>
<td>792</td>
<td>1 CD2</td>
<td>COMMON XD = ORG = ORIGIN ADDRESS OF ANA, used to calculate</td>
</tr>
<tr>
<td>793</td>
<td>1 CD2</td>
<td>INDICES INTO 'ANA' FROM ADDRESS POINTERS</td>
</tr>
<tr>
<td>794</td>
<td>1 CD2</td>
<td>OF TOC ENTRIES</td>
</tr>
<tr>
<td>795</td>
<td>1 CD2</td>
<td>ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND</td>
</tr>
<tr>
<td>796</td>
<td>1 CD2</td>
<td>TOC RETRIEVED VIA XREQ</td>
</tr>
<tr>
<td>797</td>
<td>1 CD2</td>
<td>INTERNAL VARIABLES</td>
</tr>
<tr>
<td>800</td>
<td>1 CD4</td>
<td>COMMON XS = POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC</td>
</tr>
<tr>
<td>801</td>
<td>1 CD4</td>
<td>ENTRY, MOST SIGNIFICANT (BIT 15) BIT</td>
</tr>
<tr>
<td>802</td>
<td>1 CD4</td>
<td>SET TO INDICATE END-OF-CHAIN</td>
</tr>
<tr>
<td>803</td>
<td>1 CD4</td>
<td>INDEX = VALUE COMPUTED FROM POINTER TO BE</td>
</tr>
<tr>
<td>804</td>
<td>1 CD4</td>
<td>FORTRAN INDEX INTO 'ANA' FOR NEXT TOC ENTRY</td>
</tr>
<tr>
<td>805</td>
<td>1 CD4</td>
<td>NOTES</td>
</tr>
<tr>
<td>821</td>
<td>1 CD5</td>
<td>ROUTINE USED = EXEC, PURGE, XDATA, XPI, XREQ, XREX, XRI, XRI6.</td>
</tr>
<tr>
<td>822</td>
<td>1 CD5</td>
<td>XMOV, XMSG, XRPM, XRPC, XRES, XRUP, XCT, XCOM,</td>
</tr>
<tr>
<td>823</td>
<td>1 CD5</td>
<td>XUDBG, XVABN</td>
</tr>
</tbody>
</table>
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER Responds GO AHEAD WITH TERMINATION THEN
4 CALL XERO TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR DBOE FILES IS NOT NEGATIVE THEN
6 DO UNTIL DBOE CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
8 OUTPUT 'XDOE TOC TOO LARGE, PURGE INCOMPLETE'
9 EXIT DO
10 ENDIF
11 CALL XDOE TO CREATE FILE NAME 'XXXEX'
12 CALL PURGE TO SCRATCH FILE
13 SET DBOE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
14 ENDDO
15 ENDIF
16 IF CHAIN HEAD FOR DATA BASES IS NOT NEGATIVE, THEN
17 DO UNTIL DATA BASE CHAIN HEAD IS NEGATIVE
18 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
19 OUTPUT 'XDOE TOC TOO LARGE, PURGE INCOMPLETE'
20 EXIT DO
21 ENDIF
22 IF TYPE OF DATA BASE IS UTDD, THEN
23 CALL XDOE TO CREATE FILE NAME 'XXXN'
24 CALL PURGE TO SCRATCH FILE
25 ENDIF
26 SET DATA BASE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
27 ENDDO
28 ENDIF
29 CALL XDSTA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XDOB
32 ENDIF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVARBN - NO RETURN FROM THIS CALL
35 ENDIF
36 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
37 CALL XPEIT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE VAXEC
38 ENDIF
39 RETURN
40 END XDOFF
FORTRAN CALLING SEQUENCE:

CALL XRDE (DATBUF, DBDCB)

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

INPUTS FROM CALLING SEQUENCE:

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

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

INPUTS FROM XB COMMON:

XB(154) ALFLG, XB(155) EMDBK, XB(182) TOTSIZ,
XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

DATBUF, DBDCB

OUTPUTS IN XB COMMON:

XB(151) ABFLG, XB(152) ERFLG, XB(165) TOCHDS,
XB(169) TOPOLK, XB(170) EMDBL, XB(171) TOCNX,
XB(201) NOTOC

INTERNAL XB COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) ERFLG - ERROR MESSAGE FLAG
XB(153) ALLFLG - RESTORE ALL UDB FLAG
XB(154) DEBUG - DEBUG FLAG
XB(155) FIOHAN - DB FILE NAME
XB(156) DBLF - BLOCK # WHERE DATA ITEM BEGINS
XB(157) DATHDS - WORD INDEX INTO DATBUF WHERE DATA BEGINS
XB(164) IERR - ERROR FLAG FOR FMGR CALLS
XB(165) TOCHDS - WORDS OF TOC IN TOCBUF (NOTOC +B) + B
XB(166) NOBLSK - # BLOCKS TO READ
XB(167) LEM - # WORDS OF DATA TO MOVE
XB(168) DATEND - BLOCK # WHERE DATA ITEM ENDS
XB(169) TOPOLK - BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) EMDBL - BLOCK # OF LAST BLOCK READ
XB(171) TOCNX - INDEX INTO TOC OF CURRENT DATA ITEM
XB(175) IL - # WORDS OF DATA TO READ
XB(176) TOCENT - DATA BASE TOC ENTRY
XB(201) NOTOC - # TOC ENTRIES IN TOCBUF
XB(201) TOCBUF - TOC BUFFER
1 BEGIN 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-DESE 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 COMPUTE SIZE OF DATA TO BE READ
8 CALL READ TO READ 1 BUFFER BEGINNING WITH DATBLK FOR THIS ELEMENT
9 ERREXIT IF READ ERROR TO :ERR1:
10 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
11 ELSE, DATA BEGINS IN DATBUF
12 IF DATA DOES NOT END IN DATBUF THEN
13 CALL XRMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
14 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
15 CALL READ TO READ ENOUGH TO FILL DATBUF
16 ERREXIT IF READ ERROR TO :ERR1:
17 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
18 ENDIF
19 ENDIF
20 BUILD AND REQUEST TO ALLOCATE AND STORE DATA
21 CALL EXEC TO WRITE DATA TO SAM
22 ERREXIT IF ERROR FROM EXEC TO...
23 IF DATA REQUEST BUFFER IS FULL THEN...
24 CALL XRDE TO ISSUE REQUEST...
25 EXIT XRDE IF ERROR IN XRDE
26 ENDIF
27 ENDIF
28 ENDXRDE
29 EXIT XRDE
30 :ERR1:
31 CALL XR16 TO CONVERT ERROR CODE TO ASCII
32 CALL XRMSG TO DISPLAY ERROR MESSAGE (200)
33 GO TO :ERR3:
34 :ERR2:
35 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
36 :ERR3:
37 SET ABFLG TO SAY ABOUT RESTORE
38 END XRDE
FORTRAN CALLING SEQUENCE:

CALL XDRDF (DATBUF, DBDCB)

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

INPUTS IN CALLING SEQUENCE:

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

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

INPUTS IN XB COMMON:

XB(152) ERFLE, XB(154) ALFLG, XB(165) TOCWB,
XB(160) TOPLPB, XB(170) ENDBL, XB(171) TOCBLK,
XB(182) TOTISZ

OUTPUTS IN CALLING SEQUENCE:

DBDCB

OUTPUTS IN XB COMMON:

XB(151) ABFLG

INTERNAL XB COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) ERFLE - ERROR MESSAGE FLAG
XB(154) ALFLG - RESTORE ALL UTDB FLAG
XB(155) DEBFLG - DEBUG FLAG
XB(159) FLMEXT - DD FILE NAME
XB(162) DATNXLK - BLOCK # WHERE ORDE BEGINS
XB(163) DATNXLK- - WORD INDEX INTO DATBUF WHERE ORDE BEGINS
XB(164) ERRFLG - ERROR FLAG FOR FNGR CALLS
XB(165) TOCWS- - # WORDS OF TOC IN TOCBUF (MTOC #) + 8
XB(166) NOBLKS- - # BLOCKS OF UTDB/ORDE TO READ/WRITE
XB(167) LEN- - # WORDS OF DATA TO MOVE
XB(169) TOLPLK- - BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBL- - BLOCK # OF LAST BLOCK READ
XB(171) TOCBLK- INDEX TO CURRENT TOC ENTRY
XB(172) DBDFIL- - ORDE FILE NAME
XB(173) IL- - # WORDS OF DATA TO READ/WRITE
XB(201) TOCBUF- TOC BUFFER

COMMON USED:
1 BEGIN XDRDF
2 DO WHILE THERE ARE TOC ENTRIES TO PROCESS
3 IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
4 CALL XDRDF TO CREATE THE DRE FILE NAME
5 CALL CREAT TO CREATE THE DRE FILE
6 IF THERE WAS A CREATE ERROR THEN
7 IF ERROR FLAG IS OFF THEN
8 SET ERROR FLAG ON
9 CALL XMSG TO DISPLAY MAIN MESSAGE
10 ENDIF
11 CALL XRGB TO CONVERT ERROR CODE TO ASCII
12 CALL XREV TO DISPLAY DRE NAME AND ERROR CODE
13 ELSE
14 CALL CLOSE TO CLOSE FILE
15 ERREXIT IF CLOSE ERROR TO :ERR1:
16 CALL OPEN TO OPEN DRE FILE AS TYPE 1
17 ERREXIT IF OPEN ERROR TO :ERR1:
18 DO UNTIL ALL BLOCKS OF DRE HAVE BEEN PROCESSED
19 IF DATA FOR DRE IS IN BUFFER THEN
20 CALL WRBF DATA TO DRE
21 ERREXIT IF WRBF ERROR TO :ERR1:
22 ELSE
23 CALL READF TO READ NEXT BUFFER OF DATA
24 ERREXIT IF READF ERROR TO :ERR2:
25 SET INDICES INDICATING DATA IN BUFFER
26 ENDIF
27 ENDIF
28 ENDDO
29 CALL CLOSE TO CLOSE DRE FILE
30 ERREXIT IF CLOSE FAILED TO :ERR1:
31 BUILD ANA REQUEST TO ALLOCATE DRE IN ANA
32 IF ANA REQUEST BUFFER IS FULL THEN
33 CALL XDRF TO MAKE REQUEST
34 EXIT XDRDF IF XDRF ERROR
35 ENDIF
36 ENDIF
37 ENDIF
38 ENDDO
39 1 EXIT XDRDF
40
d:ERR1:
41 CALL PURGE TO PURGE DRE FILE
42
43
d:ERR2:
44 CALL XMSG TO DISPLAY ERROR MESSAGE WITH FILE NAME
45 SET ABFLG TO SAY ABORT RESTORE
46 1 END XDRDF
C*******
C FORTRAN CALLING PROCEDURE
C
C CALL XDRST
C
C*********
C XDRST PROCESSES THE RECALL DIRECTIVE. A UTOB IS CREATED AND
C THE CONTENTS OF THE SPECIFIED PDB ARE COPIED TO IT.
C
C*********
C INPUT
C XE - CARTR, COMM, COMTR, FLAGS, LV, TOKENS
C FILES - XDRST (PDB FILE SPECIFIED)
C
C*********
C OUTPUT
C XE - REGMT, REQTR
C FILES - XDRST (UTOB FILE SPECIFIED)
C
C*********
C INTERNAL VARIABLES
C
C DCBFD - DCB FOR THE PDB FILE; ALLOCATED IN XE COMMON;
C XE CONTAINS 152 WORD BUFFER USED TO READ THE PDB
C AND TO WRITE THE UTOB
C DCBUD - DCB FOR THE UTOB FILE; ALLOCATED IS XE COMMON
C
C*********
1200 2 :ERROR7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERROR9: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERROR16: ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERROR18: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
1204 2 :ERROR19: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERROR21: ISSUE MESSAGE - "AWA OVERFLOW - XXXX NOT LOGGED"
1206 2 :ERROR22: ISSUE MESSAGE - "XXXX ALREADY EXISTS"
1207 2 :ERROR44: ISSUE MESSAGE - "FILE ACCESS ERROR 6-- XXDB"
1208 2 :RETURN:
1209 2 IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
1210 3 PURGE UTDB
1211 2 ENDF
1212 2 IF STATUS FLAG INDICATES UTDB IS LOGGED IN AWAN, THEN
1213 3 CALL XER TO DELETE UTDB FROM AWAN
1214 2 ENDF
1215 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1216 3 CALL CLOSE FOR PDB FILE
1217 2 ENDF
1218 1 "ND XDREC

XDREC
**FORTRAN CALLING PROCEDURE**

**CALL XOREN**

**XOREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE, INTERFACE TABLE, DATA ELEMENT, UDB, OR PDB. ONLY THOSE TABLES OR ELEMENTS LOGGED IN THE USERS AVA ARE RENAMED.**

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

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

**COMMON XB - (201) IDC8: EXEC BUFFER**

**EXTERNAL REFERENCES**

**ROUTINES USED -**

**EXEC**

**樽**

**KCSV**

**MARF**

**XDDBA**

**XDWBR**

**XREG**

**XRMOV**

**XRMSG**

**XRFRM**

**XRSFR**
1 BEGIN XOREN

2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"

3 SAVE POINTER TO CURRENT NAME

4 INCREMENT TO NEXT TOKEN

5 IF TOKEN IS A HYPHEN, THEN

6 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME

7 DECODE CLASS NAME

8 EXIT TO :CLESER: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,D)

9 ELSE

10 SET CLASS TO BE

11 ENDIF

12 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"

13 SAVE POINTER TO NEW NAME

14 IF CLASS IS DATA BASE OR ORDE, THEN

15 EXIT TO : TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS

16 ENDIF

17 IF CLASS IS DATA BASE, THEN

18 EXIT TO : TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION

19 ENDIF

20 IF CLASS IS INTERFACE TABLE, THEN

21 CALL EXEC TO WRITE/READ NEW NAME

22 BUILD AWA MANAGER REQUEST TO CHANGE NAME IN TOC

23 BUILD AWA MANAGER REQUEST TO STORE NEW NAME IN TABLE

24 CALL XREN TO ISSUE REQUESTS

25 EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR

26 ELSE

27 IF CLASS IS DATA BASE, THEN

28 BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY

29 EXIT TO : NAMERR: IF RETURN INDICATES ERROR

30 EXIT TO : INVALID: IF DATA BASE IS AN NDB

31 ENDIF

32 BUILD AND ISSUE MANAGER REQUEST FOR NAME CHANGE

33 EXIT TO : NAMERR: IF RETURN CODE INDICATES ERROR

34 IF CLASS IS DATA BASE OR ORDE, THEN

35 CALL XRF/D TO FORMAT FILE NAME

36 CALL FILE MANAGER TO CHANGE DISC FILE NAME

37 EXIT TO : UNDO: IF FILE MANAGER RETURNS ERROR

38 IF FILE IS A PDB, THEN

39 CALL XDD/DB TO DELETE OLD PDB FROM XDPB

40 EXIT TO : NAMANG: IF ERROR RETURNED

41 CALL XDD/DB TO ADD NEW PDB NAME TO XDPB

42 EXIT TO : TELUSER: IF ERROR RETURNED

43 ENDIF

44 ENDIF

45 ENDIF

46 EXIT XOREN

47 :SYNTAX: CALL XRMG TO DISPLAY SYNTAX ERROR AND EXIT

48 :TOOLNG: CALL XRMG ("NEW NAME IS TOO LONG") AND EXIT

49 :CALGET:

50 CALL EXEC TO GET BUFFER CONTAINING NEW TABLE NAME

51 :NAMERR: CALL XRMG TO OUTPUT APPROPRIATE MESSAGE AND EXIT

52 :CLESER: CALL XRMG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT

53 :INVALID: CALL XRMG ("AN NDB CANNOT BE RENAMED") AND EXIT
BEGIN XREQ

CALL XREQ TO PROCESS AWA REQUEST(S)

IF AN ERROR RETURNED BY AWA MANAGER, THEN

IF ERROR FLAG (ERFLG) IS ZERO, THEN

TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED

CALL XRMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'

ENDIF

CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON

IF CLASS OF ELEMENT IS DDE, THEN

CALL PURGE TO DELETE THE FILE

ELSE, ELEMENT RESIDES IN AWA

CALL EXEC TO FREE CLASS NO. AND SAM BUFFER

ENDIF

IF AWA REQUESTS EXIST IN REQBUF BEYOND FAILING REQUEST, THEN

MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER

ENDIF

ELSE

SET REQPTR TO 1 INDICATING NO REQUESTS PRESENT

ENDIF

EXIT XREQ

END XREQ
**FORTRAN CALLING PROCEDURE**

**CALL XDRES (DATBUF)**

**INPUT**

- COMMON XE - LU, FLAGS, REPB0, REB索, TOkENS,
- CARTEG, COMPE, CONQUS

**CALING SEQUENCE**

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

**OUTPUT**

- COMMON XE - REB00

**COMMON XD - ABFLG = ABORT FLAG, NON-ZERO VALUE FROM XDRE,**
- XDRE, XD03, IXLIS OR IXDEQ INDICATES ABORT OF RESTORE

- ALLFLG = SET NON-ZERO IF LIST OF ELEMENTS SPECIFIED
- DEBUG = DEBUG BIT OF 'FLAGS' IN XE COMMON
- EMB0LEX = BLOCK # OF LAST BLOCK READ
- ERLG = SET NON-ZERO IF MESSAGE #54 ISSUED SO THAT IT IS ISSUED ONLY ONCE

- FILNAM = UTDB/MDG FILE NAME
- TOCBUF = UTDB TOC ENTRIES, 8 WORDS EACH, MAX 1200 WORDS

- TOCEN = AWA TOC ENTRY FOR THE UTDB
- TCQDEX = INDEX TO NEXT UTDB TOC ENTRY

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

**NOTES**

**ROUTINES CALLED**

- CLOSE
- EXEC
- OPEN
- READF
- XDRE
- XDRE0
- XDRE0
- XDRE0
- FDRE
- XREQ
1476 1 BEGIN XDRES
1477 2  BEGIN ABLE TO ZERO
1478 3  ERRINIT IF NEXT TOKEN IS NOT A CPWAN :ERR09:
1479 4  ERRINIT IF FOLLOWING TOKEN IS NOT A NAME :ERR08:
1480 5  ENTER THIS NAME AS DATA BASE TO BE RESTORED
1481 6  INCREMENT TO NEXT TOKEN
1482 7  IF TOKEN IS NOT A CPWAN, THEN
1483 7  ERRINIT IF TOKEN IS NOT EOS :ERR04:
1484 8  ENDIF
1485 9  BUILD VIA REQUEST FOR TOC ENTRY RETRIEVE
1486 10  CALL XRED TO PROCESS VIA REQUEST
1487 11  ERRINIT IF VIA REQUEST FAILED :ERR10:
1488 12  ERRINIT IF DATA BASE FOUND IS A PDB :ERR33:
1489 13  IF DATA BASE IS A UDB, THEN
1490 14  CALL ZIRXFM TO CONSTRUCT QUALIFIED FILE NAME
1491 15  ENDIF
1492 16  CALL OPEN TO OPEN SPECIFIED FILE
1493 17  ERRINIT IF OPEN FAILED :ERR08:
1494 18  CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1495 19  ERRINIT IF READ FAILED :ERR08:
1496 20  INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1497 21  IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1498 22  CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1499 23  ERRINIT IF READ FAILED :ERR08:
1500 24  ENDIF
1501 25  UPDATE TOTBZ TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 26  CLEAR ERROR MESSAGE FLAG (ERLKG)
1503 27  IF TOKEN IS EOS (I.E. NO LIST OF ELEMENTS), THEN
1504 28  SET ALLLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 29  ELSE
1506 30  SET ALLLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 31  CALL XDLIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 32  EXIT XDRES IF ABFLG SET BY XDLIS
1509 33  ENDIF
1510 34  CALL XDRE TO RESTORE AVA RESIDENT ELEMENTS
1511 35  1 EXIT XDRES IF ABFLG SET BY XDRE
1512 36  2 CALL ZDROF TO RESTORE ORAE'S
1513 37  3 EXIT XDRES IF ABFLG SET BY ZDROF
1514 38  4 CALL CLOSE TO CLOSE DATA BASE FILE
1515 39  5 ERRINIT IF CLOSE FAILED :ERR08:
1516 40  6 DO WHILE AVA REQUESTS REMAIN IN ERBUF
1517 41  7 CALL XDRE TO PROCESS AVA REQUESTS
1518 42  8 EXIT XDRES IF ABFLG SET BY XDRE
1519 43  ENDDO
1520 1 EXIT XDRES
1521 2 :ERR04: CALL XRMAG - "SYNTAX ERROR. EXTRANEOS DATA"
1522 2 :ERR08: CALL XRMAG - "FILE MANAGER ERROR ........".
1523 2 :ERR09: CALL XRMAG - "SYNTAX ERROR. MISSING OR ILLEGAL FIELD"
1524 2 :ERR10: CALL XRMAG - "........ NOT FOUND"
1525 2 :ERR13: CALL XRMAG - "CANNOT RESTORE A PDB"
1526 2 DO UNTIL ALL AVA REQUESTS IN ERBUF 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 ENDIF
1572 1 BEGIN YDSAV
1573 2 SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
1574 3 ERRTITIF "+/" IS NOT NEXT TOKEN :ERROR:
1575 4 INCREMENT TO NEXT TOKEN
1576 5 ERREXIT IF TOKEN IS NOT 'NAME' :ERROR:
1577 6 ERREXIT IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR:
1578 7 ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
1579 8 RETAIN THIS NAME AS PDB
1580 9 INCREMENT TO NEXT TOKEN
1581 10 ERREXIT IF "+/" IS NOT NEXT TOKEN :ERROR:
1582 11 INCREMENT TO NEXT TOKEN
1583 12 ERREXIT IF TOKEN IS NOT 'NAME' :ERROR:
1584 13 ERREXIT IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR:
1585 14 ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
1586 15 RETAIN THIS NAME AS PDB
1587 16 INCREMENT TO NEXT TOKEN
1588 17 ERREXIT IF TOKEN IS NOT DOS :ERROR:
1589 18 BUILD AWS REQUEST TO RETRIEVE UDDB'S TOC ENTRY
1590 19 CALL XRVTO TO PROCESS AWS REQUESTS
1591 20 ERREXIT IF TOC RETRIEVE FAILED :ERROR:
1592 21 ERREXIT IF TOC ENTRY DOES NOT INDICATE UDDB :ERROR:
1593 22 CALL XDODA TO ADD PDB TO UDDB
1594 23 ERREXIT IF FNRE ERROR :ERROR:
1595 24 ERREXIT IF PDB LIMIT EXCEEDED :ERROR:
1596 25 SET STATUS FLAG INDICATING PDB LOGGED
1597 26 CALL XRNFM TO BUILD UDDB FILE NAME
1598 27 CALL OPEN TO OPEN UDDB FILE (SPECIFYING TYPE 1)
1599 28 ERREXIT IF OPEN FAILED :ERROR:
1600 29 SET STATUS FLAG INDICATING UDDB FILE OPEN
1601 30 BUILD AWS REQUEST TO ALLOCATE PDB -- USE SIZE OF UT:3
1602 31 CALL XOPEN TO PROCESS AWS REQUEST
1603 32 ERREXIT IF PDB IS DUPLICATE :ERROR:
1604 33 ERREXIT IF AWS OVERFLOW :ERROR:
1605 34 SET STATUS FLAG INDICATING PDB LOGGED IN AWS
1606 35 CALL XRNFM TO BUILD PDB FILE NAME
1607 36 CALL CREAT TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE
1608 37 OF UDDB FILE FROM TOC ENTRY
1609 38 ERREXIT IF CREATE FAILED :ERROR:
1610 39 SET STATUS FLAG INDICATING PDB FILE OPEN
1611 40 DO FOR ALL DATA IN UDDB FILE
1612 41 CALL ...TO READ 1 BUFFER OF UDDB DATA
1613 42 ERREXIT IF READ FAILED :ERROR:
1614 43 CALL WRITTO TO WRITE 1 BUFFER TO PDB FILE
1615 44 ERREXIT IF WRITE FAILED :ERROR:
1616 45 ENDDO
1617 46 CLOSE UDDB FILE
1618 47 CLOSE PDB FILE
1619 48 EXIT XDSAV
1620 49 :ERROR: ISSUE MESSAGE "SYNTAX ERROR -- ILLEGAL OR MISSING FIELD"
1621 50 :ERROR: ISSUE MESSAGE "INVALID UDDB FILE NAME ...
1622 51 :ERROR: ISSUE MESSAGE "INVALID PDB FILE NAME ...
1623 52 :ERROR: ISSUE MESSAGE "SYNTAX ERROR. EXTRANEOUS DATA"
1624 53 :ERROR: ISSUE MESSAGE "PDB ..... ALREADY EXISTS"
1625 2 :ERROR: ISSUE MESSAGE "PRO FILE ACCESS ERROR ... "
1626 2 :ERROR: ISSUE MESSAGE "FILE MANAGER ERROR ... "
1627 2 :ERROR: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PRO'S ALREADY REACHED"
1628 2 :ERROR: 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 XDR TO DELETE PRO FROM ANA
1635 2 ENDIF
1636 2 IF FLAG INDICATES UTOB IS OPEN, THEN
1637 2 CLOSE UTOB
1638 2 ENDIF
1639 2 IF FLAG INDICATES PRO IS IN XPRO, THEN
1640 2 CALL XDR TO DELETE PRO FROM XPRO
1641 2 ENDIF
1642 2 ENDIF
1643 1 END XDSAV
164: 1 CBO  CALL XSTA (LU)
165: 1 CBO  CALL XSTA (LU)
166: 1 CBO  CALL XSTA (LU)
167: 1 CBO OUTPUT AWA AND DMA USAGE DATA AND STATISTICS ON AWA COMPACTIOW
168: 1 CBO  PHASES
169: 1 CBO  PHASES
170: 1 CBO  INPUT
171: 1 CBO  INPUT
172: 1 CBO  INPUT
173: 1 CBO  INPUT
174: 1 CBO  INPUT
175: 1 CBO  INPUT
176: 1 CBO  INPUT
177: 1 CBO  INPUT
178: 1 CBO  INPUT
179: 1 CBO  INPUT
180: 1 CBO  INPUT
181: 1 CBO  INPUT
182: 1 CBO  OUTPUT
183: 1 CBO  OUTPUT
184: 1 CBO  OUTPUT
185: 1 CBO  OUTPUT
186: 1 CBO  OUTPUT
187: 1 CBO  OUTPUT
188: 1 CBO  OUTPUT
189: 1 CBO  OUTPUT
190: 1 CBO  OUTPUT
191: 1 CBO  OUTPUT
192: 1 CBO  OUTPUT
193: 1 CBO  OUTPUT
194: 1 CBO  OUTPUT
195: 1 CBO  OUTPUT
196: 1 CBO  OUTPUT
197: 1 CBO  OUTPUT
198: 1 CBO  OUTPUT
199: 1 CBO  OUTPUT
200: 1 CBO  OUTPUT
201: 1 CBO  OUTPUT
202: 1 CBO  OUTPUT
203: 1 CBO  OUTPUT
204: 1 CBO  OUTPUT
205: 1 CBO  OUTPUT
206: 1 CBO  OUTPUT
207: 1 CBO  OUTPUT
208: 1 CBO  OUTPUT
209: 1 CBO  OUTPUT
210: 1 CBO  OUTPUT
211: 1 CBO  OUTPUT
212: 1 CBO  OUTPUT
213: 1 CBO  OUTPUT
214: 1 CBO  OUTPUT
215: 1 CBO  OUTPUT
216: 1 CBO  OUTPUT
217: 1 CBO  OUTPUT
218: 1 CBO  OUTPUT
219: 1 CBO  OUTPUT
220: 1 CBO  OUTPUT
221: 1 CBO  OUTPUT
222: 1 CBO  OUTPUT
223: 1 CBO  OUTPUT
224: 1 CBO  OUTPUT
225: 1 CBO  OUTPUT
226: 1 CBO  OUTPUT
227: 1 CBO  OUTPUT
228: 1 CBO  OUTPUT
229: 1 CBO  OUTPUT
230: 1 CBO  OUTPUT
231: 1 CBO  OUTPUT
232: 1 CBO  OUTPUT
233: 1 CBO  OUTPUT
234: 1 CBO  OUTPUT
235: 1 CBO  OUTPUT
236: 1 CBO  OUTPUT
237: 1 CBO  OUTPUT
238: 1 CBO  OUTPUT
239: 1 CBO  OUTPUT
240: 1 CBO  OUTPUT
241: 1 CBO  OUTPUT
242: 1 CBO  OUTPUT
243: 1 CBO  OUTPUT
244: 1 CBO  OUTPUT
245: 1 CBO  OUTPUT
246: 1 CBO  OUTPUT
247: 1 CBO  OUTPUT
248: 1 CBO  OUTPUT
249: 1 CBO  OUTPUT
250: 1 CBO  OUTPUT
251: 1 CBO  OUTPUT
252: 1 CBO  OUTPUT
253: 1 CBO  OUTPUT
254: 1 CBO  OUTPUT
255: 1 CBO  OUTPUT
256: 1 CBO  OUTPUT
257: 1 CBO  OUTPUT
258: 1 CBO  OUTPUT
259: 1 CBO  OUTPUT
260: 1 CBO  OUTPUT
261: 1 CBO  OUTPUT
262: 1 CBO  OUTPUT
263: 1 CBO  OUTPUT
264: 1 CBO  OUTPUT
265: 1 CBO  OUTPUT
266: 1 CBO  OUTPUT
267: 1 CBO  OUTPUT
268: 1 CBO  OUTPUT
269: 1 CBO  OUTPUT
270: 1 CBO  OUTPUT
271: 1 CBO  OUTPUT
272: 1 CBO  OUTPUT
273: 1 CBO  OUTPUT
274: 1 CBO  OUTPUT
275: 1 CBO  OUTPUT
276: 1 CBO  OUTPUT
277: 1 CBO  OUTPUT
278: 1 CBO  OUTPUT
279: 1 CBO  OUTPUT
280: 1 CBO  OUTPUT
281: 1 CBO  OUTPUT
282: 1 CBO  OUTPUT
283: 1 CBO  OUTPUT
284: 1 CBO  OUTPUT
285: 1 CBO  OUTPUT
286: 1 CBO  OUTPUT
287: 1 CBO  OUTPUT
288: 1 CBO  OUTPUT
289: 1 CBO  OUTPUT
290: 1 CBO  OUTPUT
291: 1 CBO  OUTPUT
292: 1 CBO  OUTPUT
293: 1 CBO  OUTPUT
294: 1 CBO  OUTPUT
**Prolog**

**Keywords:** CALL

**Description:**
- **Description:** This routine is a common use routine, designed to process data based on certain conditions.
- **Condition:** The routine processes data from a file, checks for errors, and handles various operations related to the data.

**Variables and Parameters:**
- **Variables:**
  - **XB(151)**: abort flag
  - **XB(152)**: error flag
  - **XB(153)**: message number
  - **XB(157)**: total size of UDB file
  - **XB(158)**: total words in file
  - **XB(159)**: file name
  - **XB(162)**: data record
  - **XB(164)**: UDB error flag
  - **XB(166)**: UDB name
  - **XB(201)**: UDB file buffer

**Routines:**
- **Routines:** CALL

**Comments:**
- **Comments:** This routine is used to call other routines and process data accordingly.

---

**Prolog**

**Keywords:** COMMON

**Description:**
- **Description:** This routine is a common use routine, designed to process data based on certain conditions.
- **Condition:** The routine processes data from a file, checks for errors, and handles various operations related to the data.

**Variables and Parameters:**
- **Variables:**
  - **XE(1)**: LU
  - **XE(19)**: RPRTR
  - **XE(29)**: NAME
  - **XE(33)**: COMMA
  - **XE(142)**: ICR
  - **XE(144)**: COMPRTR

**Routines:**
- **Routines:** CALL

**Comments:**
- **Comments:** This routine is used to call other routines and process data accordingly.

---

**Prolog**

**Keywords:** RTE

**Description:**
- **Description:** This routine is a common use routine, designed to process data based on certain conditions.
- **Condition:** The routine processes data from a file, checks for errors, and handles various operations related to the data.

**Variables and Parameters:**
- **Variables:**
  - **XDLIS**, **XDBRT**, **XRCP**, **XRED**, **XREX**, **XRMOV**
  - **XRMSG**, **XRSET**, **XRSFL**, **XRSFR**, **XRSMN**, **XUBB**

**Routines:**
- **Routines:** CALL

**Comments:**
- **Comments:** This routine is used to call other routines and process data accordingly.
1737 1 BEGIN XSTO
1738 2:2 ABLE TO ZERO (ABORT FLAG)
1739 3 ERR: IF UTB NAME IS NOT VALID TO :ERR2:
1740 4 BUILD REQUEST FOR AWA TOC
1741 5 CALL XREQ TO MAKE MANAGER REQUEST
1742 6 CALL EXEC TO GET AWA TOC
1743 7 ERR: IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1744 8 DO FOR ALL DATA BASE CLASS ENTRIES
1745 9 ERR: IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1746 10 END
1747 11 END DO
1748 12 ELSE
1749 13 CALL XLS TO PROCESS LIST TO BE STORED
1750 14 ERR: IF ABLE IS NOT ZERO TO :ERR:
1751 15 END IF
1752 16 SET NTOC = 0 (NUMBER OF UTB TOC ENTRIES)
1753 17 DO FOR IT, ST, DE, OR DR CHAINS
1754 18 IF STORE/RESTORE BIT IS ON THEN
1755 19 INCREMENT NTOC BY 1
1756 20 ENDIF
1757 21 END DO
1758 22 ELSE
1759 23 ERR: IF THERE ARE NO UTB TOC ENTRIES (NTOC=0) TO :ERR2:
1760 24 COMPUTE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1761 25 CALL XP " TO CREATE FILE NAME
1762 26 CALL TO CREATE UTB FOR NTSIZ
1763 27 ERR: IF ERROR IN CREATE TO :ERR3:
1764 28 CALL XWR TO WRITE UTB FILE
1765 29 ERR: IF ABLE IS 4 (ORE LARGER THAN SPECIFIED) TO :ERR1:
1766 30 ERR: IF ABLE IS 3 (ORE FILE ERROR) TO:ERRA:
1767 31 ERR: IF ABLE IS 2 (ORE FILE ERROR) TO:ERR3:
1768 32 CALL WRIT TO WRITE TOC RECORDS AT RECORD 1
1769 33 ERR: IF ERROR IN WRIT TO :ERR3:
1770 34 CALL CLOSE TO CLOSE UTB FILE
1771 35 ERR: IF ERROR IN CLOSE TO :ERR3:
1772 36 BUILD REQUEST TO AAW UTB IN AWA
1773 37 CALL XREQ TO MAKE REQUEST
1774 38 ERR: IF AWA OVERFLOW TO :ERR1:
1775 39 EXIT XSTO
1776 40 ERR:1:
1777 41 CALL XMSG TO DISPLAY MSGNO
1778 42 G0 TO :ERR4:
1779 43 ERR:2:
1794 2 CALL XREMSG TO DISPLAY MSGNO
1795 2 GO TO :ERR5:
1796 2 :ERR3:
1797 2 CALL XREMSG 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 XREMSG TO DISPLAY STORE ABORTED/ UTDB NOT CREATED MESSAGE
1803 1 END XDSTO

XDSTO XDSTO
FORTRAN CALLING PROCEDURE

CALL XDOTC

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

INPUT

COMBF, COMPR, FLAGS, LU

AWA HEADER AND TOC (SEE MODULE XMWA)

FILES - DATA BASE FILES AS APPROPRIATE

OUTPUT

AWA OR UTDB TOC DISPLAY

LOCAL VARIABLES

TOTAL ALLOCATABLE SIZE OF AWA (SEE MODULE XMWA)

FIRST PORTION OF AWA (HEADER AND TOC)

ARRAY OF ALLOCABLE CLASS DESIGNATORS FOR TOC DIRECTIVE

CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE

CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'

END OF TOC CHAIN DESIGNATOR (-32768)

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

ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)

TOC ENTRY DIM FIELD (WORD 8)

BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE

MULT - ARRAY OF MULTIPLIERS USED TO DETERMINE NUMBER OF WORDS IN COLUMNS OF TWO DIMENSIONAL DATA ELEMENTS (FUNCTION OF DATA TYPE)

ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY

ORIGIN ADDRESS OF AWA (SEE MODULE XMWA)

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

TOC ENTRY SIZE FIELD (WORD 7)

MAXIMUM SIZE OF TOC WHICH CAN BE TOTALLY ACCOMMODATED BY INTERNAL BUFFER

TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)

ARRAY OF TYPE ID CODES FOR DE AND ORDE DISPLAYS

LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY

NOTES

USES CLOSE, EXEC, IAND, IXOR, KCVT, OPEN, READF, XDOTA, XREG, XREG, XMMOV, XMRS, XREG, XREG

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 XDOTA USE THE SAME SIZE BUFFER FOR TOC MANIPULATION; THUS, A DATA BASE FILE MAY NOT HAVE A TOC TOO LARGE FOR THE XDOTC INTERNAL BUFFER.
1866 1 BECm XDOTC
1867 2 FOR OUTPUT OF ALL USER CLASSES FROM AWA TO TERMINAL
1868 3 IF NEXT TOKEN IS NOT EOS
1869 4 THEN
1870 5 IF TOKEN IS A HYPHEN
1871 6 THEN
1872 7 INCREMENT TO NEXT TOKEN
1873 8 IF TOKEN IS NOT A CHARACTER T
1874 9 THEN
1875 0 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
1876 1 SET OUTPUT UNIT FOR LINE PRINTER
1877 2 ENDIF
1878 3 INCREMENT TO NEXT TOKEN
1879 4 ENDIF
1880 5 IF TOKEN IS NOT EOS
1881 6 THEN
1882 7 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1883 8 INCREMENT TO NEXT TOKEN
1884 9 IF TOKEN IS NOT A COMMA
1885 0 THEN
1886 1 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (D, F, I, S OR B)
1887 2 SET OPTION FOR INDICATED CLASS
1888 3 INCREMENT TO NEXT TOKEN
1889 4 ENDIF
1890 5 IF TOKEN IS NOT EOS
1891 6 THEN
1892 7 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1893 8 INCREMENT TO NEXT TOKEN
1894 9 EXIT TO :ERROR: IF TOKEN IS NOT A FOUR CHARACTER NAME
1895 0 INCREMENT TO NEXT TOKEN
1896 1 EXIT TO :ERROR: IF TOKEN IS NOT EOS
1897 2 CALL XREQ TO RETRIEVE DATA BASE FILE TOC ENTRY
1898 3 EXIT TO :ERROR: IF NOT LOGGED IN TOC
1899 4 GET TYPE AND SET NAME IN HEADER
1900 5 CONSTRUCT FILE NAME
1901 6 READ FIRST DATA BASE FILE TOC RECORD
1902 7 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AWA TOC
1903 8 READ SUBSEQUENT TOC RECORDS
1904 9 EXIT TO :ERROR: IF FILE ACCESS FAILS
1905 0 DO FOR EACH NON-EMPTY TOC CHAIN
1906 1 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
1907 2 IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NULL CHAIN EXISTED
1908 3 THEN
1909 4 MARK PREVIOUS ENTRY AS AN END OF CHAIN
1910 5 ENDIF
1911 6 EMDO
1912 7 DO FOR EACH ENTRY IN TOC
1913 8 IF NOT MARKED AS AN END OF CHAIN
1914 9 THEN
1915 0 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
1916 1 ENDIF
1917 2 ENDDO
1918 3 EXCLUDE CHAIN 8 (DATA BASE FILES) FROM DISPLAY
1919 4 ENDIF
1920 5 ENDIF
1921 6 IF REFERENCING AWA TOC
1922 7 THEN
1923 8 CALL XREQ TO RETRIEVE AWA TOC
1924 9
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 DIEMS WITHIN LIMIT OF BUFFER
    THEN
      FORMAT NAME & SIZE FIELDS
      IF DEBUG AND/OR TRACE FLAGS SET
        THEN
          FORMAT I-DIM FIELD IN OCTAL
        ENDIF
      FORMAT TYPE FIELD IN INTEGER
      IF CLASS 2 OR 3
        THEN
          FORMAT TYPE FIELD USING DATA TYPE TABLE
          IF CLASS 2
            THEN
              FORMAT I-DIM & J-DIM FIELDS
            ENDIF
          ELSE
            IF CLASS 8
              THEN
                FORMAT TYPE FIELD USING FILE TYPE TABLE
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      ELSE
        PRINT 'DATA LOST' MESSAGE
        EXIT PROCESSING FOR THIS CHAIN
      ENDIF
    ENDIF
  ENDIF
ENDDO

IF PROCESSING AWA TOC
  THEN
    CALL XDTA TO DISPLAY AWA USAGE STATISTICS
  ENDIF
EXIT XDTOC

:ERROR5: EXIT WITH INVALID OUTPUT DEVICE ID
:ERROR4: EXIT WITH SYNTAX ERROR
:ERROR5: EXIT WITH INVALID CLASS DESIGNATOR
:ERROR6: EXIT WITH INVALID UDDB FILE NAME
:ERROR7: EXIT WITH UDDB FILE ACCESS ERROR
1 END XDTOC
CALL XDWRT (DATBUF)

XDWRT WRITES INTERFACE TABLES, SEQUENCE TABLES, DATA ELEMENTS
AND DRIE FILES OUT TO A

INPUTS FROM CALLING SEQUENCE:

DATBUF - (INTEGER, 1480 WORDS) BUFFER USED TO READ
IN AWA RESIDENT DATA AND DRIE DATA

INTERNAL XB COMMON USED:

ABFLE - (INTEGER, 1 WORD) ABORT FLAG
MESSAGE NUMBER TO BE DISPLAYED
NUMBER OF REQUESTS IN REQBUF
TOTAL # WORDS IN A FILE
RECORD # WHERE DATA GOES NEXT
INDEX INTO DATBUF WHERE
NEXT DATA ELEMENT IS TO START
FILE ERROR FLAG
DRDE = FILE ERROR FLAG
DRDE = TYPE 3 FILE FLAG
TOC INDEX FOR CURRENT REQUEST
CURRENT TOC INDEX
DPHE FILE NAME
LENGTH OF 1 COPY (DRDE TO UTDI)
NUMBER OF UTDI TOC ENTRIES
1200 WORDS) UTDI TOC BUFFER

COMMON USED:

(EQ(1), LU ),
+ (EX(2), FLGS ), (EX(19), REQPTR),
+ (EX(20), REQBUF)

RTE ROUTINES USED:

CLOSE, EXEC, KCVT, OPEN, READF, WRIT

FDS ROUTINES USED:

XREQ, XREXT, XMOV, XMSG,
XRFM, XRSFL, XRSFR
BEGIN XDRAT
  SET WRDNO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
  SET MORES = 0 (NUMBER OF AWA REQUESTS IN REBUFF)
  DO FOR ALL UTDTOG TOC ENTRIES UNTIL CLASS IS DRDE
    BUILD REQUEST FOR DATA FROM AWA
    INCREMENT MORES BY 1
    IF REQUEST BUFFER IS FULL (MORES=8) THEN
      PERFORM READTO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
    ENDIF
  ENDIF
  IF THERE ARE REMAINING REQUESTS (MORES>0) THEN
    SET NEXT REQUEST TO SAY END OF REQUEST LIST
    PERFORM READTO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
  ENDIF
  IF THERE IS A PARTIAL DATA RECORD LEFT (WRDNO>1) THEN
    PERFORM WRITE TO OUTPUT DATA TO UTDTOG FILE
  ENDIF
  DO FOR EACH DRDE UTDTOG TOC ENTRY
    STORE DATREC IN UTDTOG TOC ENTRY
    CALL DRDFNM TO CREATE FILE NAME
    IF DRDE FILE IS TYPE 3 THEN
      CALL OPEN TO OPEN FILE AS CORRECT TYPE
      IF OPEN ERROR TO :ERR1:
        DO UNTIL EOF IS READ
          CALL READ TO READ 1 RECORD
          ERREXIT IF READ ERROR TO :ERR1:
          STORE RECORD LENGTH AT FRONT AND REAR OF DATA
          INCREMENT WRDNO BY LENGTH + 2
          IF THERE IS ENOUGH DATA TO WRITE (WRDNO>128) THEN
            PERFORM WRITE TO OUTPUT DATA TO UTDTOG FILE
          ENDIF
        ENDIF
      ENDIF
    ENDIF
    IF THERE IS REMAINING DATA (WRDNO>1) THEN
      PERFORM WRITE TO OUTPUT DATA TO UTDTOG FILE
    ENDIF
    SET DATREC TO NEXT AVAILABLE RECORD FOR DATA
  ELSE
    CALL OPEN TO OPEN FILE AS TYPE 1
    ERREXIT IF OPEN ERROR TO :ERR1:
    COMPUTE TOTAL SIZE OF FILE IN WORDS
    DO UNTIL ALL DATA IS COPIED TO UTDTOG (SIZE=0)
      IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
        SET LENGTH = SIZE
      ENDIF
      CALL READ TO READ LENGTH DATA
      ERREXIT IF READ ERROR TO :ERR1:
      CALL WRITF TO WRITE LENGTH DATA
      ERREXIT IF WRITE ERROR TO :ERR5:
      INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
      DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
    ENDIF
  ENDIF
ENDDO
CALL CLOSE TO CLOSE DRDE FILE
ERREXIT IF CLOSE ERROR TO :ERR1:
ENDDO
EXIT XDRAT
BEGIN READ;
CALL XREF TO MAKE REQUEST
EREREERK IF THERE IS AN ANA OVERFLOW TO ERR4:
DO FOR NUMBER OF REQUESTS (MOREB) IN BUFFER
CALL EXEC TO GET DATA FROM SAM
SET DATREC AND WRKNO INTO THIS UTDB TOC ENTRY
INCREMENT WRKNO BY SIZE OF THIS ELEMENT
IF THERE IS ENOUGH DATA TO WRITE (WRKNO>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
ENDDO
REINITILIZE REQUEST BUFFER (MOREB=0)
END READ;

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

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

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

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

:ERRA:
SET ABIFLG TO SAY AWA OVERFLOW
DO FOR REMAINING REQUESTS
CALL EXEC TO FREE CLASS NUMBER
ENDDO
END XDMRT
FORTRAN CALLING PROCEDURE
CALL XELDS (THREE WORD ARRAY CONTAINING 'XECAL')

FDX EXECUTIVE INITIALIZATION SEGMENT MAIN ROUTINE

INPUT
COMMON XE - SUBSTA

OUTPUT
COMMON XB - INITIALIZED ACCORDING TO THE VALUE OF SUBSTA
(SEE APPROPRIATE INITIALIZATION SUBROUTINE)

NOTE!
USES XEINO, XEINI, XEIN, XEINS, XEINX, XERN

BEGIN XECAL
CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SREDT:, :INTED:, SUBSTA (-1 TO 3))

:GLOBAL: CALL XEINO TO INITIALIZE GLOBAL COMMON
:DIRECT: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES
:EXECUT: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL
:SREDT: CALL XEINS TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING
:INTED: CALL XEINX TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING

END XECAL
1 BEGIN XEINE
2 INITIALIZE COMMON TO ZEROS
3 SET FILE (EXIT+ DCE NUMBER
4 SET STATES TO DIRECTIVE LEVEL
5 SET TKNLNG
6 INITIALIZE TOKENS
7 READ LIBRARY DIRECTORY FIRST RECORD
8 EXIT TO :ERROR1: IF READ FAILED
9 STORE NUMBER OF PROCESSORS IN NOPROC
10 READ PROCESSOR NAME RECORD
11 EXIT TO :ERROR1: IF READ FAILED
12 CALL XREN TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AMA
13 EXIT TO :ERROR1: 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 XEINE WITH INITIALIZATION FAILURE
25 ENDIF
26 CALL OPEN TO OPEN PDB/MDB DIRECTORY (XPOB)
27 EXIT TO :ERROR1: IF OPEN FAILED
28 CALL READF TO READ 1ST RECORD OF XPOB (LIST OF MDB'S)
29 EXIT TO :ERROR1: IF READ FAILED
30 PERFORM DLOG TO LOG MDB'S IN AMA
31 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER
32 CALL READF TO READ THAT RECORD OF XPOB
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 XREN TO PROCESS THE AMA REQUESTS
38 ENDIF
39 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
40 EXIT XEINE
41 BEGIN DLOG
42 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
43 BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
44 IF 8 AMA REQUESTS HAVE BEEN BUILT, THEN
45 CALL XREN TO PROCESS AMA REQUESTS
46 ENDIF
47 EXIT TO :ERROR1: IF A REQUEST FAILED
48 ENDDO
49 END DLOG
50 :ERROR1: LIBRARY INITIALIZATION ERROR TERMINATION
51 1 END XEINE
POSITION TO 1ST TOKEN AFTER 'INTO'

EXIT IF TOKEN IS NOT ',' :ERR1:

INCREMENT TO NEXT TOKEN

EXIT IF TOKEN IS NOT A NAME :ERR2:

SET PRCNAM TO THIS NAME

INCREMENT TO NEXT TOKEN

SET INTNAM TO 0

SET NEWTAB TO 'INTAB'

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

EXIT IF TOKEN IS NOT A COMMA :ERR2:

INCREMENT TO NEXT TOKEN

IF TOKEN IS NOT EOS, THEN

IF TOKEN IS A NAME, THEN

SET INTNAM TO THE NAME

INCREMENT TO NEXT TOKEN

ENDIF

IF TOKEN IS NOT EOS, THEN

EXIT IF TOKEN IS NOT A COMMA :ERR2:

INCREMENT TO NEXT TOKEN

IF TOKEN IS A NAME, THEN

SET NEWTAB TO THE NAME

INCREMENT TO NEXT TOKEN

ENDIF

ENDIF

EXIT IF TOKEN IS NOT EOS :ERR2:

ENDIF

ELSE

INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER

NEWTAB IS SET TO 'INTAB'

MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED

ENDIF

MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE

EXIT IF REQUEST IS UNSUCCESSFUL :ERR11:

START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR

EXIT IF PROCESSOR PRCNAM IS FOUND

ELSE

INCREMENT TO NEXT ENTRY

ENDLOOP

EXIT :ERR8:

ENDSEARCH

SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND

IF INTNAM = 0, THEN

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

OPEN THE DEFAULT INTERFACE TABLE FILE

EXIT IF OPEN FAILED :ERR12:

READ THE DEFAULT INTERFACE TABLE

EXIT IF THE READ FAILED :ERR16:

READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM

OF WKBUF

EXIT IF READ FAILED :ERR14:

ELSE

MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE

EXIT IF ERROR OR COULD NOT FIND :ERR4:

ENDIF

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

CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"

OPEN THE PROMPT TABLE

EXIT IF OPEN FAILED :ERR15:
READ 2ND RECORD OF THIS PROMPT TABLE
EXEIT 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: "INTNAME" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNAME DOES MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
304  1 CD0      FORTRAN CALLING PROCEDURE
305   CD0
306   CD0
307   CD0      CALL XEINS
308   CD0
309   CD0
310   CD0
311   CD0      INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING
312   CD0
313   CD0
314   CD0
315   CD0      INPUT
316   CD0
317   CD0      COMMON XE - Cartag, Combuf, Comptr, Lu, Noproc, Tokens
318   CD0
319   CD0
320   CD0
321   CD0      OUTPUT
322   CD0
323   CD0      COMMON XB - Debug, Direct, Newtab, Numdir, Nument,
324   CD0      Oldtab, Prntmg, Wkblng, Wkbuf, Xlibd
325   CD0
326   CD0
327   CD0
328   CD0
329   CD0
330   CD0
331   CD0      NOTES
332   CD0
333   CD0
334   CD0      USES FILES - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE
335   CD0
336   CD0
337   CD0
338   CD0
339   CD0      USES ROUTINES
340   CD0
341   CD0
342   CD0
343   CD0
344   CD0
345   CD0
346   CD0
347   CD0      C*********
1 BEGIN XEINS
2 BUILD AWA REQUEST TO RETRIEVE "XLIBD"
3 ERREXIT IF TOKEN IS NOT COMMA :ERR02:
4 INCREMENT TO NEXT TOKEN
5 IF TOKEN IS NAME, THEN
6 BUILD AWA REQUEST TO RETRIEVE OLDTAB
7 RETAIN THIS NAME AS OLDTAB
8 INCREMENT TO NEXT TOKEN
9 ELSE
10 SET OLDTAB TO ZERO
11 EMDIF
12 ERREXIT IF TOKEN IS NOT COMMA :ERR02:
13 INCREMENT TO NEXT TOKEN
14 ERREXIT IF TOKEN IS NOT NAME :ERR02:
15 IF NEWTAB NOT EQUAL TO OLDTAB, THEN
16 BUILD AWA REQUEST TO VERIFY EXISTENCE OF NEWTAB
17 EMDIF
18 CALL XREQ TO PROCESS AWA REQUEST(S)
19 IF NEWTAB NOT EQUAL TO OLDTAB, THEN
20 ERREXIT IF NO ERROR FROM XREQ :ERR12:
21 EMDIF
22 IF OLDTAB NOT ZERO, THEN
23 ERREXIT IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED :ERR04:
24 SET PROMPT MODE AS UPDATE...
25 SET NO. ENTRIES AS OLDTAB SIZE / 7
26 READ OLDTAB INTO WORKING BUFFER
27 ERREXIT IF FIRST REQUEST (RETRIEVE "XLIBD") FAILED :ERR01:
28 READ "XLIBD" INTO COMMON
29 ELSE
30 SET PROMPT MODE AS CREATE
31 SET NO. ENTRIES TO ZERO
32 EMDIF
33 OPEN, READ AND CLOSE FILE XSPRM
34 SET COUNT AND SEREDIT DIRECTIVES INTO XB COMMON
35 SET SUBSTATE FLAG TO SEQ. EDIT. (=2)
36 1 EXIT XEINS
37 2 :ERR01: CALL XRMSG - 'INITIALIZATION ERROR ...'
38 2 :ERR02: CALL XRMSG - 'SYNTAX ERROR'
39 2 :ERR04: DEFAULT MESSAGE TO '...NOT FOUND'
40 2 IF ERROR WAS NO AWA SPACE THEN
41 2 SET MSG TO '...NO AWA SPACE'
42 2 EMDIF
43 2 CALL XRMSG TO DISPLAY MESSAGE
44 2 :ERR12: CALL XRMSG - '... ALREADY EXISTS'
45 1 END XEINS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XE - LITPTR, NUMARG, WKBLEN, WKBUF

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

OUTPUT

COMMON XE - LITPTR, LITDYN, NARC, WBUF

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

USES ROUTINES

XIEXT XRMOV XRNKB XRSIG XRSRT
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 D-BIT 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-BIT 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 EMDIF
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 O'S) INTO NEW LITERAL ENTRY AREA
18 EMDO
19 EMDIF
20 INCREMENT TO NEXT ARGUMENT
21 ENDOLOOP
22 ERREXIT IF NOT CALLED BY LIBRARY MAINTENANCE:ERRS:
23 SKIP TO NEXT LITERAL ENTRY
24 ENDSWARCH
25 1 EXIT TO :RETURN:
26 :ERRS: CALL XRMSG - 'SYSTEM INITIALIZATION ERR S'
27 2 :RETURN:
28 1 END XEINT
FORTRAN CALLING PROCEDURE

CALL XEINX

initialize XE AND XB COMMON FOR EXECUTION CONTROLLER

common XE - COMBUF, COMPTR, FLAGS, LU, MASSA, NOPROC, TOKENS

common XF - AYN - SEQUENCE TABLE, LIBRARY DIRECTORY NAME TABLE

common XF - LIBD, NOPRC2, SELNG, WKBLNG, WKBUF

NOTES

USES EXEC, PRIM, XREQ, XREAT, XR16, XRMOV, XRMSG, XUDG, XVABN
1 BEGIN XEIN
2 IF INITIALIZATION FROM DIRECTIVE
3 THEN
4 IF DIRECTIVE IS NAME
5 THEN
6 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT EOS
7 ELSE
8 IF DIRECTIVE IS AUTO
9 THEN
10 IF TOKEN IS A HYPHEN
11 THEN
12 INCREMENT TO NEXT TOKEN
13 EXIT TO :ERROR3: IF TOKEN IS NOT THE NAME 'T'
14 CHANGE EXECUTION CONTROL MODE TO AUTO-T
15 INCREMENT TO NEXT TOKEN
16 ENDIF
17 EDBF
18 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A COMMA
19 INCREMENT TO NEXT TOKEN
20 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A NAME
21 STORE NAME IN SEGNAME
22 CALL XREG TO RETRIEVE SEQUENCE TABLE
23 EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
24 INCREMENT TO NEXT TOKEN
25 SET SENSTA TO FIRST SEQUENCE NUMBER
26 SET SESENND TO LAST SEQUENCE NUMBER
27 IF TOKEN NOT EOS
28 THEN
29 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
30 INCREMENT TO NEXT TOKEN
31 IF TOKEN IS AN INTEGER
32 THEN
33 STORE STARTING RANGE NUMBER
34 SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
35 EXIT TO :ERROR3: IF NOT FOUND
36 INCREMENT TO NEXT TOKEN
37 ENDIF
38 IF TOKEN NOT EOR
39 THEN
40 EXIT TO :ERROR2: IF TOKEN NOT A COMMA
41 INCREMENT TO NEXT TOKEN
42 EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
43 ERREXIT TO :ERROR3: IF 'ENDING SEG # < BEGINNING SEG #
44 SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
45 EXIT TO :ERROR3: IF NOT FOUND
46 INCREMENT TO NEXT TOKEN
47 EXIT TO :ERROR2: IF TOKEN NOT :OS
48 EDBF
49 ENDIF
50 SET SENPRT TO SENSTA
51 EDBF
52 EDBF
53 INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
54 EXIT TO :ERROR1: IF INITIALIZATION FAILS
55 1 EXIT KEINP
56 2 :ERROR1: INITIALIZATION FAILURE TERMINATION
57 2 :ERROR2: SET SUBSTA /D DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
551  2 :ERR03: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552  2 :ERR04: SET SUBSTA TO DIRECTIVE LEVEL
553  2 IF ERROR WAS 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 EXIT XEIXX
1 CD000000
562 1 CD0000000000000
563 1 000000000000000000000
564 1 CD000000
565 1 000000000000000000000
566 1 CD000000
567 1 000000000000000000000
568 1 CD000000
569 1 000000000000000000000
570 1 CD000000
571 1 000000000000000000000
572 1 CD000000
573 1 000000000000000000000
574 1 CD000000
575 1 000000000000000000000
576 1 CD000000
577 1 000000000000000000000
578 1 CD000000
579 1 000000000000000000000
580 1 CD000000
581 1 000000000000000000000
582 1 CD000000
583 1 000000000000000000000
584 1 CD000000
585 1 000000000000000000000
586 1 CD000000
587 1 000000000000000000000
588 1 CD000000
589 1 000000000000000000000
590 1 CD000000
591 1 000000000000000000000
592 1 CD000000
593 1 000000000000000000000
594 1 CD000000
595 1 000000000000000000000
596 1 0
597 1 0
598 1 0
599 1 BEGIN XELDS
600 2 :XELDS:
601 2 CALL XELDS
602 2 CALL SEGMENT NAME INTO EXEC PARAMETER LIST
603 2 CALL SEGMENT NAME INTO EXEC PARAMETER LIST
604 2 :XELDIS:
605 2 RETURN TO MAIN PROGRAM
606 1 END XELDS

FORTRAN CALLING PROCEDURE:
CALL XELDS (SEGNAME) TO CALL SEGMENT FROM MAIN
CALL XELTH TO RETURN TO MAIN PROGRAM

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

INPUTS IN CALLING SEQUENCE:
SEGNAME - (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 XELTH
2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
3) STATEMENT - TYPE 3
4) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XELTH"
5) FOLLOWED BY A CALL TO MAIN WHICH IS NOT EXECUTED
6) WHEN LOADER IS RUN, THE FOURTH PARAMETER MUST BE 1
7) INDICATING TO LOAD MAIN PLUS SEGMENTS

BEGIN XELDS

EXECUTE THE FILE YELDS.PDF
IF XECN IS CALLED AFTER COMPLETION OF ALL AUTOMATIC AND SELF-AUTOMATIC EXECUTIONS TO ASSURE THAT ALL DO AND SCAN PROCESSOR FILES ARE PURGED FROM THE SYSTEM.

0200 CALL XECN

0205 ENDS XECN

End of XECN

END OF XECN
655  1  CD9  FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
656  1  CD1  ONE PROGRAM PER SIGNED-ON USER
657  1  CD1  
658  1  CD1  
659  1  CD1  c********
660  1  CD2  INPUT
661  1  CD2  SCHEDULING PARAMETERS - LU, CLASHO, QUAL, FLAGS (SEE XE COMMON)
662  1  CD2  
663  1  CD2  c********
664  1  CD3  OUTPUT
665  1  CD5  COMMON XE - COMBUF, COMPTR, MASSTA, SUBSTA, PLUS XEINE
666  1  CD5  INITIALIZATION
667  1  CD5  COMMON XB - INITIALIZATIONS FROM XEIN, XEINI, XEINS, XEINX
668  1  CD3  
669  1  CD3  c********
670  1  CD5  NOTES
671  1  CD5  USES AMPAR, XDCLD, XDCLF, XELDS, XINTE, XRCPR, XRMIG, XSEGE,
672  1  CD5  XTCOM, XTCMT
673  1  CD3  
674  1  CD5  THE CALLS TO XELDS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
675  1  CD5  XECAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
676  1  CD5  
677  1  CD5  THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CONTROL OCCURS
678  1  CD5  BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XTCMT FROM
679  1  CD5  CALLING XSEGE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
680  1  CD5  MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
681  1  CD5  WHEN MASSTA IS SET TO THE DIRECTIVE LEVEL.
682  1  CD5  
683  1  CD5  c********
BEGIN EXEC
RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASMO, QUAL & FLAGS
CALL XEINI TO INITIALIZE GLOBAL COMMON
DO FOREVER -- TERMINATES INSIDE HANDLER FOR XOFF
CALL ATCOM FOR INPUT OF DIRECTIVE
IF ERROR OR NOT A VALID DIRECTIVE NAME
THEN
ISSUE MESSAGE ED6
ELSE
IF NAME IS INTE
THEN
SET STATES TO INTE LEVEL
CALL XEINI TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XINTE TO EDIT TABLE
ELSE
IF NAME IS SENE
THEN
SET STATES TO SENE LEVEL
CALL XEIMS TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XSENE TO EDIT TABLE
ELSE
IF NAME IS FOR SOME EXECUTION CONTROL OPTION
THEN
SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
DO UNTIL MASSTA IS AT DIRECTIVE LEVEL
CALL XEINX TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XXCMNT TO PERFORM EXECUTIONS
IF SUBSTA IS SET TO SENE LEVEL
THEN
CALL XEINS TO REINITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XSENE TO SUPPORT EXECUTION CONTROL
ENDIF
IF SUBSTA IS SET TO INTE LEVEL
THEN
CALL XEINI TO REINITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XINTE TO SUPPORT EXECUTION CONTROL
ENDIF
ENDIF
IF EXECUTION MODE WAS SEMI OR AUTO
THEN
CALL XESCNT TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES
ENDIF
ELSE
CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?
ENDIF
ENDIF
ENDIF
ENDIF
:RESET:
IF SUBSTA IS NOT DIRECTIVE LEVEL
THEN
CALL XEIND TO REINITIALIZE DYNAMIC COMMON
ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CD0 FORTRAN CALLING PROEDURE
2 CD0
3 CD0 CALL XINTE
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0 OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR
9 CD0
10 CD0
11 CD2
12 CD2 INPUT
13 CD2
14 CD2 COMMON XE -
15 CD2
16 CD2
17 CD2 COMMON XB -
18 CD2
19 CD2
20 CD2 DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE
21 CD2
22 CD2
23 CD5 NOTES
24 CD5
25 CD5 USES Routines
26 CD5 XINIX
27 CD5 XUDMG
28 CD5 XERTW
29 CD5
30 CD5
31 * XINTE IS THE INTERFACE ROUTINE FOR THE INTERFACE TABLE EDITOR
32 * WHEN CALLED IN THE FDS EXECUTIVE'S OVERLAY STRUCTURE.
33 * 
34 BEGIN XINTE
35 CALL XINIX TO EXECUTE INTERFACE TABLE EDITOR
36 CALL XERTW TO RETURN TO XEXEC IN MAIN SEGMENT
37 END XINTE
FORTRAN CALLING PROCEDURE

CALL XINIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

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

COMMON XB -
LEN = LENGTH OF LITERAL AREA OF INTERFACE
TABLE IN WBUF
MARG = INDEX TO START OF SHORT PROMPT AREA OF
MBUF
NEWTAB = NAME OF INTERFACE TABLE TO BE CREATED BY
EDIT. INPUT = 0 IF XINTER IS CALLED
BY FD$ LIBRARY MAINTENANCE PROGRAM.
NUMARG = NUMBER OF ARGUMENTS IN THIS INTERFACE
TABLE
MBUF = 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
MBUF = WORKING BUFFER CONTAINING INTERFACE TABLE
WITH LITERALS IN COMPACTED FORM
AMA = XXXXX = NEW INTERFACE TABLE AND LITERAL AREA
(NOT DONE IF CALLED B/ LIBRARY MAINTENANCE)

USES ROUTINES
EXEC
XREG
XILE
XIPRM
XINPT
1 COS
2 XMOV
3 COS
4 XMBSEG
5 XCOM
6 *********
7 1 *
8 1 THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'INIT' DIRECTIVE
9 1 OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
10 1 PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
11 1 TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
12 1 OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
13 1 * NOTE: ALL INITIALIZATION, INCLUDING WKBUF (OLD INTERFACE TABLE),
14 1 HAS BEEN PERFORMED BY XENINI.
15 1 *
16 1 BEGIN XENIX
17 2 IF NP (NO. OF PARAMETERS) > 0, THEN
18 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
19 2 * PRNTMD = 1 => CREATE A MODE
20 2 * PRNTMD = 3 => CREATE A MODE
21 2 * PRNTMD = 4 => CREATE CONTINUE MODE
22 3 * PRNTMD = 5 => MODIFY MODE
23 4 DO UNTIL 'EXIT' OR 'X' IS ENTERED
24 5 CALL XPMRM TO CONSTRUCT A PROMPT BASED ON PRNTMD, SIZE, TYPE, AND STATUS
25 6 OF NEXT ARGUMENT
26 7 CALL XTCOM TO PROMPT USER AND RETURN PARSED INPUT
27 8 IF 'X' WAS NOT ENTERED, THEN
28 9 IF '\' WAS ENTERED, THEN
29 10 SET PRNTMD TO 5
30 11 ELSE
31 12 IF NOTHING WAS ENTERED (I.E. TOKEN IS COS), THEN
32 13 INCREMENT TO NEXT ARGUMENT
33 14 ELSE
34 15 CALL XNMT TO PROCESS THE USER'S INPUT
35 16 ENDF
36 17 ENDF
37 18 ENDO
38 19 IF 'X' WAS ENTERED, THEN
39 20 SET RETURN CODE INDICATING X
40 21 (I.E. MASSTA = 0)
41 22 ELSE
43 23 COMPRESS THE LITERAL LIST AREA
44 24 ENDF
45 25 ENDF
46 26 STORE INTERFACE TABLE AS NEWNAME
47 27 IF STORE INTO AW FAILED, THEN
48 28 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
49 29 ELSE
50 30 SET GOOD RETURN CODE
51 31 ENDF
52 1 END XENIX
FORTRAN CALLING PROCEDURE

CALL XIPRM

PROMPT CONSTRUCTOR FOR INTERFACE TABLE EDITOR

COMMON XB -
ARGNO, CFLAG, DFLAG, IARG, IOLAG,
SIZE, ITYPE, LMEFF, LITSIX, MARG,
MODBTR, MUMAR, PRMND, WKBUF

COMMON XB -
ARGNO, ISUB, MODSAV, PRMLEN,
PRMND, PROMPT

NOTES

USES ROUTINES
CMUMO
XIEXT
XILSD
XRMV
XRMB
XRPCX
XRUPK
XRUPK
XRUPK
10 CONSTRUCT PROMPT TO BE ISSUED
105 BEGIN XIPRM
106 DO UNTIL A PROMPT IS CONSTRUCTED
107 IF PRMTNO = 5, THEN
108 CONSTRUCT A \"1\" PROMPT
109 ELSE
110 IF PRMTNO = 4 (CONTINUE MORE), OR
111 PRMTNO = 6 (CONTINUE HERE MORE), THEN
112 IF ARGNO IS A SCALAR, THEN
113 CONSTRUCT PROMPT AS \"ARG\" OR \"ARG\"
114 ELSE
115 IF PRMTNO NOT = 6, THEN
116 IF THERE ARE NO EMPTY SLOTS BEYOND LAST ENTERED (LASTE)
117 COMPUTE CURRENT ELEMENT NO. (SUBSCRIPT) FROM CURRENT INDEX
118 SET PRMTNO TO 4
119 ENDIF
120 IF ARGNO IS DOUBLY SUBSCRIPTED PARAMETER, THEN
121 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
122 CONSTRUCT PROMPT AS \"ARG\"(I,J):'
123 ELSE
124 CONSTRUCT PROMPT AS \"ARG\"(ISUB):'
125 ENDIF
126 ELSE
127 IF ARGNO = BP, THEN
128 SET PRMTNO TO 5
129 ELSE
130 INCREMENT ARGNO TO NEXT PARAMETER
131 SET IFLAG TO 1, 2, OR 3 INDICATING 1: 0, OR 10
132 SET LAST ENTERED INDIKATOR (LASTE) TO 0
133 IF PRMTNO = 3, THEN
134 IF SOME DATA VALUE(S) OR PARAM NAME EXISTS FOR ARGNO, THEN
135 CALL XILSD TO LIST DATA FOR THIS ARGUMENT
136 ENDIF
137 CONSTRUCT PROMPT AS \"ARG\": OR 0 OR =0
138 ELSE
139 IF ARGNO IS MARKED INCOMPLETE, THEN
140 IF A PARTIAL LITERAL LIST EXISTS, OR
141 THIS ARGUMENT IS A SCALAR, THEN
142 COMPUTE ISUB AS FIRST EMPTY ELEMENT
143 IF DOUBLY SUBSCRIPTED PARAMETER, THEN
144 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
145 CONSTRUCT PROMPT \"ARG\"(I,J):'
146 ELSE
147 CONSTRUCT PROMPT \"ARG\"(ISUB):'
148 ENDIF
149 ELSE
150 CONSTRUCT PROMPT \"ARG\": OR 0 OR =0
151 ENDIF
152 ENDIF
153 ENDIF
154 ENDIF
155 ENDIF
3.7 1 * XILST PROCESSES THE LIST DIRECTIVE
3.8 2 BEGIN XILST
3.9 3 IF TOKEN IS ' - ', THEN
3.10 4 POSITION TO NEXT TOKEN
3.11 5 ERREXIT IF TOKEN IS NOT NAME : ERROR2;
3.12 6 ERREXIT IF NAME IS NOT 'C', 'Y', OR 'A' : ERROR2;
3.13 7 IF XMODEF = 1 OR XMODEF = 3, THEN
3.14 8 CALL XCHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
3.15 9 ELSE
3.16 10 IF XMODEF = 2 OR XMODEF = 3, THEN
3.17 11 CALL XILSL TO WRITE DATA VALUES OF THIS ARGUMENT
3.18 12 ENDIF
3.19 13 DO UNTIL EOS IS REACHED
3.20 14 ERREXIT IF TOKEN IS NOT COMMA : ERROR2;
3.21 15 ERREXIT IF NEXT TOKEN IS NOT NAME : ERROR2;
3.22 16 START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
3.23 17 IF NAME = ARGNO'S NAME IN PROP'T TABLE
3.24 18 IF XMODEF = 1 OR XMODEF = 3, THEN
3.25 19 CALL XCHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
3.26 20 ENDIF
3.27 21 IF XMODEF = 2 OR XMODEF = 3, THEN
3.28 22 CALL XILSL TO WRITE DATA VALUES OF THIS ARGUMENT
3.29 23 ENDIF
3.30 24 ENDLOOP
3.31 25 PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
3.32 26 ENDSWACH
3.33 27 ENDSWACH
3.34 28 EXIT IF NEXT TOKEN
3.35 29 :ERROR2: CALL XRESG TO WRITE 'INACTIVE SYNTAX'
3.36 23 RETURN;
3.37 21 EXIT TO :RETURN;
3.38 1 END XILST
1 CDO FORTRAN CALLING PROCEDEU&E
2 CDO CALL XISUB
3 C**********
4 CDO1 EFFECTIVE SUBSCRIPT CALCULATION ROUTINE
5 CDO1 C**********
6 CD02 INPUT
7 CD02 C**********
8 CD02 COMMON XE - COMPBUF, COMPTR, TOKENS
9 CD02 COMMON XB - IDIM, ISIZE, LENEFF
10 CD02 C**********
11 CD03 OUTPUT
12 CD03 C**********
13 CD03 COMMON XB - IRETC, ISUB
14 CD03 C**********
15 CD05 USES Routines
16 CD05 XRMG
17 CD05 C**********
18 CD05 XISUB
19 CD05 C**********
20 * XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
21 * THE INPUT SUBSCRIPT
22 BEG XISUB
23 INCREMENT TO NEXT TOKEN
24 ERRExT IF TOKEN IS NOT INTEGER VALUE :ERR14:
25 IF IDIM FOR THIS ARGUMENT > 0, THEN
26 ERRExT IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
27 INCREMENT TO NEXT TOKEN
28 ERRExT IF TOKEN IS NOT COMMA :ERR15:
29 INCREMENT TO NEXT TOKEN
30 ERRExT IF TOKEN IS NOT INTEGER VALUE :ERR14:
31 CALCULATE ISUB AS (J-1)*IDIM+1
32 ELSE
33 SET ISUB TO INTEGER VALUE
34 ENDF
35 ERRExT IF ISUB > X :ERR16:
36 INCREMENT TO NEXT TOKEN
37 ERRExT IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
38 EXIT TO RETURN:
39 ERR14: CALL XRMG - 'INVALID SUBSCRIPT SYNTAX'
40 ERR15: CALL XRMG - 'DOUNTY SUBSCRIPTED - MUST SPECIFY BOTH'
41 ERR16: CALL XRMG - 'INVALID SUBSCRIPT VALUE'
42 RETURN:
43 END XISUB
FORTRAN CALLING PROCEDURE

CALL XDAT

LITERAL DATA PROCESSOR

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS
COMMON XB - CFLAG, COMFLG, DFLAG, IARG, IARG4,
ISIZE, ISIZE, LTYPE, LNEFF, LITDSP,
LITOWN, LITSZ, MODSY, MXBTH, MXBIT,
PRTNFD, SFLAG, UXBLNG, WKBTF

OUTPUT

COMMON XB - IRET, ISUB, LITDSP, LITOWN, MXBTH,
PRTNFD, WKBUF

INTERNAL VARIABLES

COMMON XS -

IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS
LITERAL DATUM
NUMCL = NUMBER OF CONSECUTIVE ELEMENTS TO BE
MARKED COMPLETED AS A RESULT OF THIS
STACK = PUSH-DOWN LIST (MAX. OF 4 ENTRIES)
DESCRIBING NESTED REPEAT GROUPS.

WORD 1 = INDEX TO 1ST TOKEN (IN COMBUF)
AFTER REPEAT SYMBOL
WORD 2 = REPEAT COUNT
WORD 3 = FLAG INDICATING WHETHER
REPEAT GROUP IS PARENTHETICALLY
GROUPED

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

NOTES

USES ROUTINES

XISUB
XRBIT
XRMOV
XRMSG
1 CD5 XWXB
2 CD5 XRIET
3 CD5
4 XW
5 *PROCESS THE INPUT LITERAL LIST
6 BEGIN XDAT
7 DO UNTIL EOS IS REACHED, OR
8 UNTIL AN ERROR OCCURS
9 IF THIS IS A DATA ELEMENT, THEN
10 (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
11 IF ARG. TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
12 ERRER IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
13 SET LENGTH TO BE MOVED (LENOV) TO EFFECTIVE LENGTH FOR DATA
14 OF ARG'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
15 ELSE
16 IF ARG. TYPE IS CHAR., THEN
17 SET LENGTH TO BE MOVED (LENOV) TO EFFECTIVE LENGTH
18 FOR DATA INPUT
19 ELSE THIS MUST BE CHARACTER DATA BEING INPUT
20 IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ERR10:
21 INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
22 COMPUTE NO. WORDS IN INPUT CHARACTER STRING
23 IF NO. WORDS (LENOV) > EFFECTIVE LENGTH OF
24 THIS ARGUMENT'S DATA (LENEFF) :ERR10:
25 ENDIF
26 ENDIF
27 INCREMENT TOKEN POINTER TO THE DATA INPUT
28 VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
29 ARGUMENT FOR DATA INPUT (ISUB <= SSIZE - LEMOV + 1)
30 ERRER IF INSUFFICIENT SPACE :ERR11:
31 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
32 ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
33 ENDIF
34 MOVE DATA FROM INPUT COMMUNICATIONS BUFFER TO LITERAL AREA
35 SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
36 ARGUMENT, TO LEMOV
37 IF LEMOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
38 MOVE LEMOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
39 ENDIF
40 INCREMENT TO NEXT TOKEN
41 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LEMOV
42 SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
43 ELSE , TOKEN INDICATES NON-DATA ELEMENT
44 IF THIS IS A "(" , THEN
45 CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
46 BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITYPE) AND
47 SECONDARY DIMENSION (IDIM)
48 EXIT XDAT IF ERROR (IUREC < 0)
49 ELSE
50 IF THIS IS AN "[", THEN
51 CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
52 ELSE
53 IF ONE ELEMENT OF THIS ARG AT ISUB AS INCOMPLETE
54 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
55 LENGTH OF ONE ELEMENT (LENEFF)
56 ENDIF
57 TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
58 INCREMENT TO NEXT TOKEN
59
ELSE
  IF TOKEN IS A REPEAT COUNT, THEN
    IF NEXT TOKEN IS "(", THEN
      SET PARENFLAG TO 1
    INCORRECT TO NEXT TOKEN
  ELSE
    SET PARENFLAG TO 0
  ENDIF
  SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
  ELSE
    ERREXIT (INVALID FIELD) :ERROR
  ENDIF
  INCREMENT TO NEXT TOKEN
ENDIF
DO UNTIL TOKEN IS NOT "("
  IF PUSH-DOWN STACK IS NOT EMPTY, AND
    (PREVIOUS TOKEN WAS DATA, AND
    PARENFLAG OF TOP OF STACK ENTRY IS 0), OR
    (CURRENT TOKEN IS '), AND
    PARENFLAG OF TOP OF STACK ENTRY IS 1), THEN
    GET REPEAT COUNT OF TOP OF STACK ENTRY
    DECREMENT REPEAT COUNT BY 1
    IF REPEAT COUNT > 0, THEN
      SET TOKEN INDEX TO INDEX ON PUSH-DOWN STACK
      REPLACE NEW REPEAT COUNT ON PUSH-DOWN STACK
    ELSE
      POP (I.E. REMOVE) ENTRY FROM TOP OF STACK
    IF TOKEN IS ")", THEN
      INCREMENT TO NEXT TOKEN
      EXIT LOOP
      ENDF
ELSE
  IF PREVIOUS TOKEN WAS NOT A SUBSCRIPT, THEN
    IF THIS TOKEN IS NOT AN EOS, THEN
      ERREXIT IF TOKEN IS NOT A COMMA :ERROR
    INCORRECT TO NEXT TOKEN
    ENDF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
IF PRIORITY MODE NOT CONTINUE (<4), AND
  (THERE ARE EMPTY ELEMENTS BEYOND LASTE, OR
  PREVIOUS TOKEN WAS A COMMA), THEN
  RETAIN PRIORITY MODE (SET MODE TO PRMTWMD)
  SET PRIORITY MODE TO CONTINUE (<4)
ENDIF
IF ALL LITERAL SLOTS FILLED, THEN
  MARK ARGNO COMPLETE
  IF ALL ARGUMENTS ARE COMPLETE, THEN
    SET COMPLETE FLAG FOR INTERFACE TABLE
    ENDF
ELSE
  TURN OFF COMPLETE FLAG FOR INTERFACE TABLE
ENDIF
ENDIF
ENDIF
ENDIF
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>622</td>
<td>1 CD0 FORTRAN CALLING PROCEDURE</td>
</tr>
<tr>
<td>623</td>
<td>1 CD0</td>
</tr>
<tr>
<td>624</td>
<td>1 CD0</td>
</tr>
<tr>
<td>625</td>
<td>1 CD0 CALL XINPT</td>
</tr>
<tr>
<td>626</td>
<td>1 CD0</td>
</tr>
<tr>
<td>627</td>
<td>1 C*******</td>
</tr>
<tr>
<td>628</td>
<td>1 CD1</td>
</tr>
<tr>
<td>629</td>
<td>1 CD1 INTERFACE TABLE EDITOR'S INPUT PROCESSOR</td>
</tr>
<tr>
<td>630</td>
<td>1 CD1</td>
</tr>
<tr>
<td>631</td>
<td>1 C*******</td>
</tr>
<tr>
<td>632</td>
<td>1 CD2</td>
</tr>
<tr>
<td>633</td>
<td>1 CD2 INPUT</td>
</tr>
<tr>
<td>634</td>
<td>1 CD2</td>
</tr>
<tr>
<td>635</td>
<td>1 CD2 COMMON XE - COMBUF, COMPTR, TOKENS</td>
</tr>
<tr>
<td>636</td>
<td>1 CD2</td>
</tr>
<tr>
<td>637</td>
<td>1 CD2 COMMON XB - DIRECT, IFLAG, ISUB, NARG, NDIR, NUMARG, PRNMD, WKBUF</td>
</tr>
<tr>
<td>638</td>
<td>1 CD2</td>
</tr>
<tr>
<td>639</td>
<td>1 CD2</td>
</tr>
<tr>
<td>640</td>
<td>1 C*******</td>
</tr>
<tr>
<td>641</td>
<td>1 CD3</td>
</tr>
<tr>
<td>642</td>
<td>1 CD3 OUTPUT</td>
</tr>
<tr>
<td>643</td>
<td>1 CD3</td>
</tr>
<tr>
<td>644</td>
<td>1 CD3 COMMON XB - ARGNO, IRETC</td>
</tr>
<tr>
<td>645</td>
<td>1 CD5</td>
</tr>
<tr>
<td>646</td>
<td>1 C*******</td>
</tr>
<tr>
<td>647</td>
<td>1 CD5</td>
</tr>
<tr>
<td>648</td>
<td>1 CD5 NOTES</td>
</tr>
<tr>
<td>649</td>
<td>1 CD5</td>
</tr>
<tr>
<td>650</td>
<td>1 CD5 USES ROUTINES</td>
</tr>
<tr>
<td>651</td>
<td>1 CD5</td>
</tr>
<tr>
<td>652</td>
<td>1 CD5 XIFMT</td>
</tr>
<tr>
<td>653</td>
<td>1 CD5 XIELF</td>
</tr>
<tr>
<td>654</td>
<td>1 CD5 XILSF</td>
</tr>
<tr>
<td>655</td>
<td>1 CD5 XIPAM</td>
</tr>
<tr>
<td>656</td>
<td>1 CD5 XIMIF</td>
</tr>
<tr>
<td>657</td>
<td>1 CD5 XRCPR</td>
</tr>
<tr>
<td>658</td>
<td>1 CD5 XRMIF</td>
</tr>
<tr>
<td>659</td>
<td>1 CD5 XRMIF</td>
</tr>
<tr>
<td>660</td>
<td>1 CD8</td>
</tr>
<tr>
<td>661</td>
<td>1 C*******</td>
</tr>
</tbody>
</table>
1 * XINPT PROCESSES THE USER'S INPUT TEXT
2 BEGIN XINPT
3 IF PRTPNO = 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 1
14 ENDF
15 ELSE IF TOKEN IS 'B', THEN
16 SET IFLAG TO 0
17 ENDF
18 ELSE IF IFLAG NOT SET, THEN
19 CASE NAME (:EXIT: :PROMPT: :LIST:)
20 ERREXIT IF ANOTHER TOKEN FOLLOWS :ERROR2:
21 :EXIT: SET IRETC SO THAT PROMPTING LOOP TERMINATES
22 :PROMPT: CALL XINPT TO PROCESS PROMPT DIRECTIVE
23 :LIST: CALL XLST TO PROCESS LIST DIRECTIVE
24 ENDCASE
25 ENDF
26 START SEARCH UNTIL IF ENTRIES
27 EXEC IF NAME FOUND IN PROMPT TABLE
28 SET AENO TO ENTRY NO.
29 SET ISUB TO 1
30 GOREL
31 INC KERMENT TO NEXT PROMPT TABLE ENTRY
32 ENDLMP
33 ERREXIT :ERROR10:
34 ENDSEARCH
35 ERREXIT IF IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:
36 ENDF
37 IF NEXT TOKEN IS A NAME, THEN
38 CALL XIPAR TO PROCESS A PARAMETER FIELD
39 ELSE
40 ERREXIT IF IFLAG IS NOT I ("=") :ERROR8:
41 CALL XIDAT TO PROCESS DATA LIST
42 ENDF
43 1 EXIT XINPT
44 1 EXIT TO :RETURN:
45 2 :ERROR2: CALL XMSG "INVALID SYNTAX"
46 2 :ERROR8: CALL XMSG "MUST USE PARAMETER NAME TO RIGHT OF & OR =" 
47 2 :RETURN:
48 1 END XINPT
FORTRAN CALLING PROCEDURE

CALL XIPAR

********

PROCESS AN INPUT PARAMETER NAME AND ANY ASSOCIATED SUBSCRIPT

FIELD(S)

********

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - DFLAG, IARG, IAREA, ICLASS,
LITDSP, SFLAG, WKBLOG, WKBUF

********

OUTPUT

COMMON XB - IRETC, LITDWN, WKBUF

********

NOTES

USES ROUTINES

XRMCT

XRMCS

XRMSE

XRMST

********
1 * XIPAR PROCESSES 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 ERREXIT 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 (LITDOWN)
11 PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT
12 ELSE
13 SET LITOSP IN ARGNO'S SPECS TO ISUB
14 ENDF
15 ELSE
16 SET LITOSP IN ARGNO'S SPECS TO 0
17 ENDF
18 ERREXIT IF ORDE HAS MORE THAN 4 CHARACTERS :ERR18:
19 ERREXIT IF EXTRAN'OUS FIELD INPUT :ERR02:
20 TURN OF D-FLAG (SOME LITERAL DATA) IN ARGNO'S SPECS.
21 SET PARAMETER NAME INTO ARGUMENT'S CHARACTERISTICS
22 SET COMPLETE (AND S) FLAG IN ARGUMENT'S CHARACTERISTICS
23 IF ALL ARGUMENTS ARE COMPLETE, THEN
24 SET INTERFACE TABLE COMPLETE FLAG
25 ENDF.
26 ELSE, MUST BE A & INPUT
27 "REXIT IF NOT AN AMPERSAND (&) INPUT :ERR02:
28 CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS
29 SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
30 ENDF
31 EXIT TO :RETURN:
32 :ERR02: CALL XRMG - "INVALID SYNTAX"
33 :ERR14: CALL XRMG - "INVALID SUBSCRIPT SYNTAX"
34 :ERR18: CALL XRMG - "INVALID ORDE NAME"
35 :RETURN:
36 1 END XIPAR
832 9 * XILSP WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
833 1 * IS RETURNED AS A PROMPT.
834 2 BEGIN XILSP
835 3 SET ARGUMENT NAME INTO BUFFER
836 4 USE T0FLG TO DETERMINE WHICH OF 'B', 'A', OR 'B'
837 5 WILL GO INTO THE PRINT BUFFER
838 6 ** BY-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
839 7 IF A PARAMETER NAME IS SPECIFIED, THEN
840 8 PUT PARAMETER NAME INTO BUFFER
841 9 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
842 10 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
843 11 PUT SUBSCRIPT INTO BUFFER
844 12 ELSE
845 13 IF LITSP OF ARGUMENT IS > 0, THEN
846 14 COMPUTE AND CONVERT THIS SUBSCRIPT
847 15 PUT SUBSCRIPT INTO BUFFER
848 16 ENDF
849 17 WRITE OUT THE PRINT BUFFER BUILT
850 18 ENDF
851 19 ELSE
852 20 LOCATE LITERAL LIST AND MASK
853 21 IF SYMBOLIC STRING, THEN
854 22 CALL XILS TO PRINT SYMBOLIC STRING
855 23 ELSE
856 24 DO UNTIL ALL ELEMENTS PROCESSED
857 25 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
858 26 UNTIL ALL ELEMENTS PROCESSED
859 27 COMPUTE AND CONVERT THE SUBSCRIPT
860 28 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
861 29 PUT "0" INTO BUFFER
862 30 ELSE
863 31 CONVERT THE DATA USING XRD6, XRE14, OR XR16
864 32 PUT DATA AND "," INTO BUFFER
865 33 ENDF
866 34 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
867 35 REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
868 36 ENDIF
869 37 WRITE OUT THE PRINT BUFFER BUILT
870 38 ENDF
871 39 ENDIF
872 40 END XILSP
1  C00    FORTRAN CALLING PROCEDURE
879 1 C00
880 1 C00
881 1 C00
882 1 C00
883 1 C00
884 1 C00
885 1 C00
886 1 C00
887 1 C00
888 1 C00
889 1 C00
890 1 C00
891 1 C00
892 1 C00
893 1 C00
894 1 C00
895 1 C00
896 1 C00
897 1 C00
898 1 C00
899 1 C00
900 1 C00
901 1 C00
902 1 C00
903 1 C00
904 1 C00

872 1 C00
873 1 C00
874 1 C00
875 1 C00
876 1 C00
877 1 C00
878 1 C00
879 1 C00
880 1 C00
881 1 C00
882 1 C00
883 1 C00
884 1 C00
885 1 C00
886 1 C00
887 1 C00
888 1 C00
889 1 C00
890 1 C00
891 1 C00
892 1 C00
893 1 C00
894 1 C00
895 1 C00
896 1 C00
897 1 C00
898 1 C00
899 1 C00
900 1 C00
901 1 C00
902 1 C00
903 1 C00
904 1 C00

XILSS IS CALLED BY XILSS TO LIST SYMBOLIC STRING DATA

INPUT

COMMON XI - LU
COMMON XB - DEBUG, LISTLU, WKBUF
COMMON XS - BUFFER = PRINT LINE BUFFER ALREADY INITIALIZED WITH NAME = NAME
BUFF = INDEX INTO BUFFER OF NEXT POSITI:
DATPTR = INDEX INTO WKBUFF OF SYMBOLIC STRING DATA

OUTPUT

COMMON XS - BUFFER, BUFPTR, DATPTR

C**
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, TOKSIZ)
EXIT TO ERROR 2 IF FIELD = 0
IF SIZE < 0, THEN
SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMBOLIC STRING
ENDIF
IF TOKSIZ < 0, THEN
SET TOKSIZ TO SIZE + 2
ENDIF
IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
CLEAR PRINT BUFFER TO BLANKS
ENDIF
IF FIELD > 0, THEN
MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
ELSE
CASE ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT, -FIELD

:ONE: CALL XR16 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING
AND PUT RESULTS INTO PRINT BUFFER
: TWO: CALL XR14 WITH VALUE IN NEXT 2 WORDS OF SYMBOLIC STRING
AND PUT RESULTS INTO PRINT BUFFER
: THREE: CALL XR18 WITH VALUE IN NEXT 3 WORDS OF SYMBOLIC STRING
AND PUT RESULTS INTO PRINT BUFFER
: FOUR: MOVE THE NEXT 3 WORDS OF SYMBOLIC STRING INTO PRINT BUFFER
: FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMBOLIC STRING
: SIX: CALL XR16 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING AND PUT RESULTS
INTO PRINT BUFFER FOLLOWED BY AN 'R'
:EXIT: PUT A * CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
INDEX BY 1
CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
EXIT XILSS
ENDCASE
INCREMENT PRINT BUFFER INDEX BY SIZE
FORTRAN CALLING PROCEDURE

CALL XICHR

PRINT THE CHARACTERISTICS OF AN ARGUMENT

INPUT

COMMON XE - LU

COMMON XB - ARGNO, IOSTAT, ISIZE, ITYPE,
LENFR, NARG

USES ROUTINES

EXEC

XRIOA

XRMOV

XRPOX

XRUMF

WRITE ARGUMENT CHARACTERISTICS

BEGIN XICHR

BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPT:, I/O TYPE AND DATA TYPE

WRITE OUT THE PRINT BUFFER

END XICHR
1053 1 CD0       FORTRAN CALLING PROEDURE
1054 1 CD0
1055 1 CD0
1056 1 CD0       CALL XIEXT
1057 1 CD0
1058 1 CD0
1059 1 C********
1060 1 CD1       EXTRACT VARIOUS FIELDS OF AN ARGUMENT CHARACTERISTICS
1061 1 CD1       AND PUT VALUES INTO COMMON
1062 1 CD1
1063 1 CD1
1064 1 C********
1065 1 CD2       INPUT
1066 1 CD2
1067 1 CD2
1068 1 CD2       COMMON XB - ARGMO, ISIZES, WKBUF
1069 1 CD2
1070 1 C********
1071 1 CD3
1072 1 CD3
1073 1 CD3       COMMON XB - CFLAG, DFLAG, IDARG, JARGA,
1074 1 CD3       ICLASS, IDIM, IOFLAG, ISIZE,
1075 1 CD3
1076 1 CD3       ISUB, ITYPE, LNEFF, LIODSP,
1077 1 CD3       LITSIZE, MDIBTM, MOBITM, SFLAG
1078 1 CD3
1079 1 C********
1080 1 CD5
1081 1 CD5       NOTES
1082 1 CD5
1083 1 CD5       USES ROUTINES
1084 1 CD5
1085 1 CD5       IAMD
1086 1 CD5       XIEXT
1087 1 CD5
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. (ARGMO), 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
1071 1 CBO FORTRAN CALLING PROCEEDURE
1072 1 CBO CALL XILIT
1073 1 CBO********
1074 1 CBO PACK LITERAL ENTRIES INTO FORMAT FOR STORAGE OF INTERFACE
1075 1 CBO TABLE INTO AMA
1076 1 CBO********
1077 1 CBO********
1078 1 CBO********
1079 1 CBO********
1080 1 CBO********
1081 1 CBO********
1082 1 CBO********
1083 1 CBO********
1084 1 CBO********
1085 1 CBO********
1086 1 CBO********
1087 1 CBO********
1088 1 CBO********
1089 1 CBO********
1090 1 CBO********
1091 1 CBO********
1092 1 CBO********
1093 1 CBO********
1094 1 CBO********
1095 1 CBO********
1096 1 CBO********
1097 1 CBO********
1098 1 CBO********
1099 1 CBO********
1100 1 CBO********
1101 1 CBO********
1102 1 CBO********
1103 1 CBO********
1104 1 CBO********
1105 1 CBO********
1106 1 CBO********
1107 1 CBO********
1108 1 CBO********
1109 1 CBO********
1110 1 CBO********
1111 1 CBO********
1112 1 CBO********
1113 1 CBO********
1114 1 CBO********
1115 1 CBO********
1116 1 CBO********
1117 1 CBO********
1118 1 CBO********
1119 1 CBO********
1120 1 CBO********
1121 1 CBO********
1122 1 CBO********
1123 1 CBO********
1124 1 CBO********
1125 1 CBO********
1126 1 CBO********
1127 1 CBO********
1128 1 CBO********
1129 1 CBO********
1130 1 CBO********
1131 1 CBO********
1132 1 CBO********
1133 1 CBO********
1134 1 CBO********
1135 1 CBO********
1136 1 CBO********
1137 1 CBO********
1138 1 CBO********
1139 1 CBO********
1140 1 CBO********
1141 1 CBO********
1142 1 CBO********
1143 1 CBO********
RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:

LD XLMAN,LU,0,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 4 - ON

INTERNAL VARIABLES:

PROMPT - (INTEGER, 31 WORDS) PROMPT FOR OPTION TO BE EXECUTED
TEMP - (INTEGER, 1 WORD) TEMPORARY USED FOR RESPONSE
TOKENS - (INTEGER, 29 WORDS) THE 29 TOKENS TO BE INITIALIZED INTO COMMON

RTE FUNCTIONS AND SUBROUTINES USED:

EXEC,AMPAR

FDS FUNCTIONS AND ROUTINES USED:

XELDS, XLLDF, XLDCL, XLMNT, XLMOD, XLMNG, XLPFM, XLPFG, XLCOM, XLMIV, XLMNG

XE COMMON USED:

EQUIVALENCE (XE(1), LU ),
+ (XE(2), ICCLASS), (XE(3), ISECU ),
+ (XE(4), FLAGS ), (XE(5), MAPST ),
+ (XE(6), SUBSTR ), (XE(?), NUMER ),
+ (XE(?), XVALFLG), (XE(85), TOKENS ),
+ (XE(142), ICR ), (XE(145), COMBUF )
1 BEGIN XLXMAN
2 CALL RMPAR TO GET INPUT PARAMETERS
3 INITIALIZE COMMON TO ZERO
4 SET CLASS NUMBER TO ZERO
5 CALL EXEC TO GET A CLASS NUMBER
6 EXIT XLXMAN IF SECURITY CODE IS NOT VALID
7 CALL XMOV TO INITIALIZE TOKENS IN COMMON
8 DO FOREVER
9 :PROMPT:
10 :INITIALIZE MASTER AND SUBSTATE FLAGS
11 CALL XCCM TO PROMPT FOR OPTION
12 IF XCTM RETURN CODE IS NOT ZERO OR
13 FIRST TOKEN IS NOT AN INTEGER OR
14 INTEGER > 7 THEN
15 CALL XMSG TO WRITE INVALID RESPONSE
16 GO TO :PROMPT:
17 ENDIF
18 CASE INTEGER (:XLPM:, :XLPRM:, :XLPRM:, :XLPRO:, :XLDEL:, :XLMOD:, :
20 :XLPM:
21 SET NUMBER TO INTEGER
22 CALL XELS TO LOAD XLPM TO CREATE SYSTEM PROMPT FILE
23 :XLPRO:
24 SET VALFLAG TO SAY ORIGINAL XLPRO REQUEST
25 CALL XELS TO LOAD XLPRO TO ADD A PROCESSOR
26 DO UNTIL VALFLAG SAYS EXIT (X)
27 CALL XELS TO LOAD XLMNT TO ENTER DEFAULT VALUES
28 CALL XELS TO LOAD XLMNT TO COMPLETE PROCESSING
29 ENDDO
30 :XLDEL:
31 CALL XELS TO LOAD XLDEL TO DELETE A PROCESSOR
32 :XLMOD:
33 SET VALFLAG TO SAY ORIGINAL XLMOD REQUEST
34 CALL XELS TO LOAD XLMOD TO MODIFY A PROCESSOR
35 DO UNTIL VALFLAG SAYS EXIT (X)
36 CALL XELS TO LOAD XLMNT TO ENTER DEFAULT VALUES
37 CALL XELS TO LOAD XLMNT TO COMPLETE PROCESSING
38 ENDDO
39 :XLMSG:
40 CALL XELS TO LOAD XLMSG TO ADD A MESSAGE
41 :XLDBF:
42 CALL XELS TO LOAD XLDBF TO HANDLE DATA BASE FILES
43 ENDDO
44 :EXIT:
45 CALL EXEC TO RELEASE CLASS NUMBER
46 END XLXMAN
FORTAN CALLING PROCEDURE:
CALL XELDS ('XLPRM')

CREATES ONE OF THE SYSTEM PROMPT FILES DEPENDING ON "NUMBER"

INPUT FROM COMMON:

NUMBER - (INTEGER, 1 WORD) USERS RESPONSE INDICATING WHICH SYSTEM PROMPT FILE TO CREATE:
1 - >XIPRM
2 - >XIPRM
3 - >XIPRM

INTERNAL VARIABLES:

CON3 - (INTEGER, 1 WORD) FIRST TOKEN IN COMBUF
CON4 - (INTEGER, 1 WORD) FIRST DATA IN COMBUF
MESS - (INTEGER, 1 WORD) CONTAINS APPROPRIATE MESSAGE NUMBER
WITH WHICH TO CALL XMSG
NAME - (INTEGER, 9 WORDS) 3 ELEMENT ARRAY, EACH ELEMENT IS A SYSTEM PROMPT FILE NAME
MODIR - (INTEGER, 1 WORD) NUMBER OF DIRECTIVES
N7OK - (INTEGER, 1 WORD) NUMBER OF TOKENS
PROMS - (INTEGER) PROM1, PROM2, PROM3, PROM4 ARE ALL USER PROMPT ARRAYS

FTE FUNCTIONS AND SUBROUTINES USED:
RCVT, CLOSE, CREATE, PURGE, WATT

FDS FUNCTIONS AND Routines USED:
XRM0, XRM5, XTCOM

COMMON USED:

ERUVALENCE

+F(3), ISECU, (X(7), NUMBR),
BEGIN XLPM

CALL XICOM TO PROMPT USER FOR O DIRECTIVES
EREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
CALL CREAT TO CREATE PROMPT FILE
IF FILE ALREADY EXISTS THEN
CALL XICOM TO PROMPT USER FOR SCRATCH/CANCEL
EXIT XLPM IF RESPONSE IS CANCEL
CALL PURGE TO PURGE PROMPT FILE
EREXIT IF PURGE ERROR TO :FILERR:
GO TO :PRM1:
ELSE (CREATE NEW FILE)
EREXIT IF CREATE ERROR TO :FILERR:
CALL XICOM TO PROMPT USER FOR LIST OF DIRECTIVES
EREXIT IF LIST IS INCONSISTENT WITH O OF DIRECTIVES TO :ERR1:
DO FOR EACH DIRECTIVE
EREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
ENDDO
CALL WRTIF TO WRITE LIST OF DIRECTIVES TO FILE
EREXIT IF WRTIF ERROR TO :FILERR:
DO FOR EACH DIRECTIVE
CALL XICOM TO PROMPT USER FOR DEFINITION
DO UNTIL EOS IS REACHED IN RESPONSE
EREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
EREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
MOVE RESPONSE INTO BUFFER
SET CONTROL CHARACTERS IN BUFFER
EREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
ENDDO
SET REMAINDER OF BUFFER TO NULL
CALL WRTIF TO WRITE DEFINITION
EREXIT IF WRTIF ERROR TO :FILERR:
ENDDO
CALL CLOSE TO CLOSE FILE
EREXIT IF CLOSE ERROR TO :FILERR:
CALL XMSG TO DISPLAY FILE CREATED MESSAGE
ENDIF
EXIT XLPM

:ERR1: (ERROR IN RESPONSE)
CALL XMSG TO DISPLAY ERROR IN RESPONSE
RETURN TO REPROMPT USER FOR ANOTHER RESPONSE

:FILERR: (FILE ACCESS ERROR)
CALL XMSG TO DISPLAY ERROR
CALL CLOSE TO CLOSE FILE
CALL PURGE TO PURGE FILE
END XLPM

REPORTIBILITY OF THE ORGINIAL PAGE IS POOR
208 1 CD************
209 1 CD0
210 1 CD0 FORTRAN CALLING SEQUENCE:
211 1 CD0 CALL XELS ('XLPAD')
212 1 CD0 **********
213 1 CD1 XLPAD WILL ADD A PROCESSOR TO THE LIBRARY DIRECTORY. IF
214 1 CD1 THE PROCESSOR HAS A DEFAULT INTERFACE TABLE, IT WILL ALSO
215 1 CD1 CREATE A PROMPT FILE AND INTERFACE TABLE FILE
216 1 CD1 **********
217 1 CD2 INPUT FROM COMMON:
218 1 CD2 VALFLG - PROCESS CONTROL
219 1 CD2 0 - ORIGINAL REQUEST TO ADD
220 1 CD2 1 - COMPLETE PROCESSING
221 1 CD2 **********
222 1 CD4 INTERNAL VARIABLES:
223 1 CD4 ABSTR - (INTEGER, 128 WORDS) CONTAINS THE ABSTRACT OF THE
224 1 CD4 PROCESSOR IN LATER BUILDS. PRESENTLY IS A NULL RECORD.
225 1 CD4 BLOCKS - (INTEGER, 1 WORD) NUMBER OF BLOCKS TO ALLOCATE
226 1 CD4 TO INTERFACE TABLE FILE
227 1 CD4 DIRECT - (INTEGER, 6 WORDS) NAME ARRAY CONTAINING VALID
228 1 CD4 DIRECTIVES FOR THE INTERFACE TABLE EDITOR
229 1 CD4 NAMEY - (INTEGER, 3 WORDS) COMBINATION OF PROCESSOR
230 1 CD4 NAME AND VERSION NUMBER
231 1 CD4 PRMAH - (INTEGER, 3 WORDS) PROCESSOR NAME
232 1 CD4 PROMS - (CHARACTERS) PROMPTS FOR USER TO BE PROMPTED WITH
233 1 CD4 TYPE - (INTEGER, 1 WORD) TYPE OF PARAMETER USED IN
234 1 CD4 CALCULATING SIZE
235 1 CD4 VERS - (INTEGER, 1 WORD) VERSION NUMBER OF PROCESSOR
236 1 CD4 **********
237 1 CDS RTE AND FNR Routines USER:
238 1 CDS EXEC, INAD, EVNT, CREAT, OPEN,
239 1 CDS READ, WAFT, CLOSE, PURGE
240 1 CDS **********
241 1 CDS FDS Routines USER:
242 1 CDS XLFIL, XLIM, XPLFL, XLPB
243 1 CDS XREP, XNMD, XRMGD, XPCOK, XRSET, XLIU, XTOP, XTCOM
244 1 CDS **********
245 1 CDS COMMON USER:
246 1 CDS EQUIVALENCE
247 1 CDS (XEB(3), ECSCU),
248 1 CDS XEB(3), MASTIA), (XEB(4), SUOSTA),
249 1 CDS (XEB(5), EOSTOK), (XEB(6), IINTOK),
250 1 CDS XEB(7), WALTG), (XEB(8), PRMAC),
251 1 CDS XEB(9), XRB(26), ZIPR ), (XEB(10), COMUF),
252 1 CDS XEB(11), VERS ), (XEB(12), ARCNO ),
253 1 CDS XEB(13), IT ), (XEB(14), LITPIT),
1 BEGIN XLPRO
2 IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
3 INITIALIZE MASTER AND SUB STAGES
4 PERFORM LINK TO UPDATE LIBRARY DIRECTORY
5 IF PROCESSOR HAS AN INTERFACE TABLE THEN
6 SET RETN = 2
78 CALL XTCOM TO PROMPT FOR 8 PARAMETERS
8 ERREXIT IF XTCOM RETURN CODE IS NOT ZERO OR
9 ERREXIT IF 8 PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
10 CREATE HEADEF ENTRY WITH 8 PARAMETERS AND PROCESSOR NAME
11 CALL XTLMK TO WRITE INSTRUCTIONS FOR ENTERING SPECS
12 DO FOR 8 PARAMETERS
13 CALL XLPS3 TO CREATE ONE PARAMETER ENTRY
14 ENDIF
15 SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
16 CALL XLPLF TO CREATE PROMPT FILE
17 SET RETN = 3
18 IF RETURN CODE IS NOT ZERO OR
19 IF RESPONSE WAS YE THEN
20 SET FLAG TO CALL INTERFACE TABLE EDITOR
21 EXIT XLPRO
22 ENDIF
23 ELSE
24 SET CODES ARRAY TO ADD ONLY ABSTRACT
25 CALL XLPLF TO CREATE PROMPT FILE
26 PERFORM XLPRO - NO RETURN EXPECTED
27 ENDIF
28 ENDIF
29 CALL XTFL TO CREATE THE DEFAULT INTERFACE TABLE FILE
30 SET VALFLAG TO SET ORIGINAL REQUEST TO ADD A PROCESSOR
31 PERFORM XLPRO - NO RETURN EXPECTED
32 :PRMERR:
33 CALL XMGM TO DISPLAY ERROR MESSAGE
34 GO TO (:PRMPT1:, :PRMPT2:, :PRMPT3:), RETN
351 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 XRMGS TO WRITE ERROR: TOO MANY PROCESSORS
11 EXIT XLPRO
12 ENDF
13 SET RETM = 1
14
15 :PRMT1:
16 CALL XTCOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
17 EXIT XLPRO IF RETURN CODE SAYS I ENTERED
18 ERREXIT IF RETURN CODE > ZERO OR
19 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
20 CALL XRMV 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 XRSRT TO SET BIT ON/OFF
29 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
30 INCREMENT # PROCESSORS BY 1
31 CALL XRMV 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:**

```fortran
CALL XLCDB
```

**INPUTS IN COMMON:**

- XE(3) QNUM
- XE(7) NUMBR
- XE(9) SECU
- XE(14) LCR
- XB(3) OLDFIL
- XB(6) NEWFIL
- XB(9) TOTSIZ

**INTERNAL VARIABLES:**

- XB(17) FREC - FIRST RECORD NUMBER IN DATBUF
- XB(18) LREC - LAST RECORD NUMBER IN DATBUF
- XB(40) IDCB - DCB FOR OLDFIL
- XB(56) IDCB2 - DCB FOR NEWFIL
- XB(200) TDBUF-BUFFER FOR COMPLETE DATA BASE TOC

**RTE ROUTINES USED:**

- CLOS, CREATE, KCVT, OPE,
- PURGE, READF, WRITF

**FSN ROUTINES USED:**

- XDOBD, XREXT, XRSNG

**COMMON USED:**

- EQUIVALENCE (XE(3), QNUM), (XE(7), NUMBR), (XE(9), SECU)
- +XE(14), LCR
- +XB(3), OLDFIL
- +XB(6), NEWFIL
- +XB(9), TOTSIZ
- +XB(10), FILCHR
- +XB(12), OLDFIL
- +XB(14), SIZE
- +XB(16), TOCPTR
- +XB(17), FREC
- +XB(39), JERR
- +XB(40), IDCB
- +XB(56), IDCB2
- +XB(200), TDBUF
10   BEGIN XLCD
11   CALL OPEN TO OPEN OLD FIL
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   ENDF
20   SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
21   CALL CREAT TO CREATE NEWFIL
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 CHAR OR
26       ERREXIT IF CLASS IS DNE AND NAME > 2 CHAR TO :ERR1:
27       APPEND NOB CONVENTION TO FRONT OF NAME
28   ELSE
29       REMOVE NOB CONVENTION FROM NAME
30   ENDF
31   ENDDO
32   CALL WRITE TO WRITE NEW TOC RECORDS TO NEWFIL
33   ERREXIT IF WRIT ERROR TO :ERR1:
34   SET TOCPTR TO FIRST TOC ENTRY
35   SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
36   CONVERT SIZE FROM BLOCKS TO WORDS
37   DO UNTIL ALL RECORDS ARE CUPIED (SIZE = 0)
38       SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
39       IF SIZE < LENGTH THEN
40           SET LENGTH TO SIZE
41       ENDF
42       CALL READ TO READ LENGTH WORDS OF DATA
43       ERREXIT IF READ ERROR TO :ERR1:
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 NO IN TOC ENTRY > LAST RECORD NUMBER
48       IF CLASS IS AN INTERFACE TABLE THEN
49           COMPLETE INDEX INTO DATBUF FROM FIRST RECORD &, RECORD # IN TOC ENTRY
50           AND INDEX IN TOC ENTRY
51       SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
52       ENDF
53   ENDSCH
54   CALL WRIT TO WRITE LENGTH WORDS OF DATA TO NEWFIL
55   ERREXIT IF WRIT ERROR TO :ERR1:
56   ENDDO
458 2 CALL CLOSE TO CLOSE NEWFIL
459 2 CALL CLOSE TO CLOSE OLDfil
460 2 CALL XDODB TO DELETE OLDfil FROM PDB LOG FILE
461 2 CALL PURGE TO PURGE OLDfil FROM SYSTEM
462 1 EXIT XLCDB

463 2 :ERR1:
464 2 CALL CLOSE TO CLOSE NEWFIL
465 2 CALL PURGE TO PURGE NEWFIL

466 2 :ERR2:
467 2 CALL CLOSE TO CLOSE OLDfil

468 2 :ERR3:
469 2 IF REQUEST WAS PDB TO MOD THEN
470 3 SET QUAL TO SAT DELETE MOD FILE
471 3 ELSE (REQUEST WAS MOD TO PDB)
472 3 SET QUAL TO SAT DELETE PDB FILE
473 2 ENDF
474 2 CALL XDODB TO DELETE MOD/PDB FROM LOG FILE
475 2 IF ERROR WAS FILE MANAGER THEN
476 3 CALL XRMSG TO DISPLAY ERROR AND RETURN CODE
477 3 ELSE
478 3 ENDF
479 2 ENDF
480 1 END XLCDB
FORTRAN CALLING SEQUENCE:

CALL XELOD ('XLDBF')

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

INPUTS IN COMMON:

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

INTERNAL VARIABLES IN COMMON

XE(3) QUAL - USER QUALIFIER REQUIRED BY XDB ROUTINES AND XRQFW
XE(8) SECU - TEMPORARY SLOT FOR ISECU DURING THIS OVERLAY
XE(35) OLDNIL - OLD FILE NAME
XE(6) NEWFIL - NEW FILE NAME
XE(9) TOTSIZE - SIZE OF OLDFIL/NEWFIL IN BLOCKS
XE(10) FILCHR - 6 BASE CHARACTERS OF FILE NAME
XE(12) QUALIF - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPENW

FDS ROUTINES USED:

XDDBA, XDBF, XERTH, XLDDB, XLPCR

COMMON USES:

ENQUIVENCE (XE(3), ISECU),
+ (XE(3), QUAL ), (XE(7), NUMORD ),
+ (XE(8), SECU ), (XE(85), TOKENS),
+ (XE(142), ICR ), (XE(145), COMDFU),
+ (XE(35), OLDNIL), (XE(6), NEWFIL),
+ (XE(9), TOTSIZE), (XE(10), FILCHR),
+ (XE(12), QUALIF), (XE(99), IERR),
+ (XE(100), ISECB)
1 BEGIN XLDBF:
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 XLPCF TO CREATE MDB/PDB LOG FILE:
6      ELSE
7        ERREXIT IF OPEN ERROR TO :FILEERR:
8      ENDIF
9    ENDIF
10   ELSE
11     DO UNTIL USER REQUESTS EXIT (X):
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 4 CHARACTER NAME AND ID IN COMMON:
16          IF REQUEST WAS PDB TO MDB THEN
17            SET QUALIFIER TO SEARCH FOR PDB NAME:
18            ELSE IF REQUEST WAS PDB TO PDB THEN
19              SET QUALIFIER TO ADD PDB TO LOG FILE:
20              ELSE IF REQUEST WAS MDB TO PDB THEN
21                SET QUALIFIER TO ADD PDB TO LOG FILE:
22                ELSE IF REQUEST WAS MDB TO PDB THEN
23                  CALL XDDOB TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER:
24                  ERREXIT IF NAME WAS NOT FOUND TO :ERR1:
25                  ERREXIT IF FILE MANAGER ERROR TO :FILEERR:
26                  IF REQUEST WAS PDB TO MDB THEN
27                    CALL XRQFM TO SET OLDFIL TO PDB NAME:
28                    SET NEWFIL TO PDB NAME:
29                    ELSE IF REQUEST WAS PDB TO PDB THEN
30                      CALL XRQFM TO SET NEWFIL TO PDB NAME:
31                      ELSE IF REQUEST WAS MDB TO PDB THEN
32                        CALL XRQFM TO COPY OLDFIL TO NEWFIL:
33                      ENDIF
34                    ENDIF
35                    CALL XLDBB TO COPY OLDFIL TO NEWFIL:
36                    IF REQUEST IS TO COPY OLDFIL TO NEWFIL THEN
37                      ENDIF
38                  ENDIF
39                ENDIF
40              ENDIF
41            ENDIF
42          ENDIF
43        ENDIF
44      ENDIF
45    ENDIF
46  ENDIF
47  ENDIF
48  ENDIF
49  ENDIF
50  ENDIF
51  ENDIF
52  ENDIF
53  ENDIF
54  ENDIF
55  ENDIF
56  ENDIF
57  ENDIF
58  ENDIF
59  ENDIF
60  ENDIF
61  EXIT XLDBBF:

1  :ERRT:
2  CALL XRMSG TO DISPLAY ERROR:
3  RETURN TO PROMPT FOR ANOTHER 4 CHARACTERS AND USER ID:

2  :FILEERR:
2  CALL XRMSG TO DISPLAY FILE ACCESS ERROR:
1 END XLDBBF
588 1 CD***********
589 1 CD0
590 1 CD0 FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
591 1 CD0 CALL XELD'S ('XLDEL')
592 1 CD0
593 1 CD0***********
594 1 CD0
595 1 CD1
596 1 CD1 XLDEL DELETES A PROCESSOR FROM THE LIBRARY DIRECTORY AND
597 1 CD1 THE PROMPT FILE. IF THE PROCESSOR HAS AN INTERFACE TABLE,
598 1 CD1 IT DELETES THE DEFAULT INTERFACE TABLE FILE ALSO.
599 1 CD1
600 1 CD***********
601 1 CD4
602 1 CD4 INTERNAL VARIABLES:
603 1 CD4 COMMN - (INTEGER, 7 WORDS) IS A TEMPORARY WORK AREA
604 1 CD4 WHERE ENTRY IN LIBRARY DIRECTORY IS STRIPPED
605 1 CD4 DOWN TO PROCESSOR NAME
606 1 CD4 PRNAM - (INTEGER, 2 WORDS) PROCESSOR NAME TO BE
607 1 CD4 DELETED.
608 1 CD4
609 1 CD***********
610 1 CD5
611 1 CD5 RTE/ FMGR Routines used:
612 1 CD5 IAND,KCVT,CREAT,OPEN,READF,WRITE,CLOSE,PURGE
613 1 CD5
614 1 CD5 FDS ROUTINES USED:
615 1 CD5
616 1 CD5 XRPCR, XREXT, XRMOV, XRMSG, XRPCK, XRSET, XRUPK, XTCON
617 1 CD5
618 1 CD5 XE AND XB COMMON USED
619 1 CD5
620 1 CD5 EQUIVALENCES (X(3), ISECU ),
621 1 CD5 + (X(142), ICR ), (X(145), COMBUF ),
622 1 CD5 + (X(48), LIBD1 ), (X(51), LIBD2 )
623 1 CD5
624 1 CD***********
1 BEGIN XLDEL

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

```fortran
CALL XLIFL
```

**XLIFL CREATES THE DEFAULT INTERFACE TABLE FILE**

**INTERNAL VARIABLES**

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

**FDS ROUTINES USED:**

- XREXT, XMEN

**RTE ROUTINES USED:**

- CLOSE, CREAT, WRITF

**COMMON USED:**

- XE(3), ISEXU
- XE(142), ICR
- XE(90), LIPTIR
- XE(91), LITLEM
- XE(96), MOPARM
- XE(101), MEMP
- XE(108), PAMMS
- XS(1), IERR
- XS(2), IDCB

**EXTERNAL FILES:**

- READ float
- WRITE float
FCRTAN CALLING SEQUENCE:
CALL XLIN

XLIN 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 XLIN
CALL EXEC TO DISPLAY ALL 7 LINES
END XLIN
FORTRAN CALLING PROCEDURE:

CALL XELBS (XLINT)

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

FDX FUNCTIONS AND SUBROUTINES USED:

XEINT, XERTW, XINIX, XINOV

COMMON USED:

EQUIVALENCE (XEC(5), MASSTA),
(XEB(2), DIRECT), (XEB(25), LSTFLG),
(XEB(26), NAGI), (XEB(41), PANTMP),
(XEB(73), IISIZE), (XEB(89), LITDOWN),
(XEB(90), LITFTN), (XEB(91), LITLEN),
(XEB(92), MAPE),
(XEB(96), NUMARG), (XEB(97), MENTAB),
(XEB(100), NGBKNG), (XEB(101), NGBKUF),
(XEB(1400), ENDB)
850 1 BEGIN XLMOD
851 2 IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
852 3 SET RTN = 1
853 4 IF (PRMID = "" OR PRMID = "XL") THEN
854 5 CALL XLMOD TO DISPLAY SPEC INSTRUCTIONS
855 6 INITIALIZE MASTERS AND SUB STATE FLAGS
856 7 CALL XCP TO PROMPT FOR PROCESSOR NAME
857 8 EXIT XLMOD IF RETURN CODE SAYS X ENTERED
858 9 ERREXIT IF XCMN RETURN CODE NON-ZERO OR
859 10 ERREXIT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO :PRMERR:
860 11 CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
861 12 ERREXIT IF THERE IS A FILE ERROR TO :FILEERR:
862 13 ERREXIT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO :PRMERR:
863 14 SAVE INTERFACE TABLE BIT AND VERSION NUMBER
864 15 PERFORM VERSION TO UPDATE VERSION NUMBER
865 16 IF THE PROCESSOR HAS AN INTERFACE TABLE THEN
866 17 SET NEW VERSION NUMBER IN INTERFACE TABLE
867 18 CREATE DEFAULT INTERFACE TABLE NAME
868 19 CALL OPEN AND READF TO READ IN NEAR AND SPECS
869 20 IF THERE ARE LITERALS THEN
870 21 CALL READF TO READ IN LITERALS
871 22 ENDIF
872 23 CALL CLOSE TO CLOSE FILE
873 24 ERREXIT IF THERE WAS A FILE ERROR TO :FILEERR:
874 25 CREATE PGM NAME
875 26 CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
876 27 ERREXIT IF THERE WAS A FILE ERROR TO :FILEERR:
877 28 CALL NAMF TO RENAME PROMPT FILE >XLMP
878 29 ERREXIT IF NAMF ERROR TO :FILEERR:
879 30 SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPEC
880 31 PERFORM DLPFM TO DELETE PARAMETERS
881 32 PERFORM MODP FM TO MODIFY PARAMETERS
882 33 PERFORM ADDP FM TO ADD PARAMETERS
883 34 CALL XLPFL TO CREATE NEW PROMPT FILE
884 35 PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
885 36 ELSE
886 37 CALL NAMF TO RENAME PROMPT FILE >XLMP
887 38 ERREXIT IF NAMF ERROR TO :FILEERR:
888 39 SET CODES ARRAY TO MODIFY ABSTRACT ONLY
890 40 PERFORM XLMOD - NO RETURN EXPECTED
891 41 ENDIF
892 42 CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
893 43 ERREXIT IF FILE ERROR TO :FILEERR:
895 45 CALL XLMOD TO CREATE NEW DEFAULT INTERFACE TABLE FILE
896 46 SET VALFILE TO SAY ORIGINAL REQUEST TO MODIFY
897 47 PERFORM XLMOD - NO RETURN EXPECTED
898 48 IF (PRMID = "" OR PRMID = "XL") THEN
899 49 CALL XLMOD TO DISPLAY ERROR MESSAGE
900 50 GO TO (:PRM1;:PRM2;:PRM3;:PRM4;:PRM5;:PRM6;:PRM7;:RTN
901 51 ELSE IF (PRMID = "" OR PRMID = "XL") THEN
902 52 CALL XLMOD TO DISPLAY FILE ERROR
903 53 END XLMOD
904 54
1 BEGIN VERSION
2 SET RTN = 2

:PRON2:
1 CALL XCTOM TO PROMPT USER FOR VERSION NUMBER
2 IF RETURN CODE IS NOT CR THEN
3 ERROR IF RETURN CODE IS NOT-ZERO CR
4 ERROR IF VERSION IS INVALID TO :PRMIND:
5 IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
6 CALL XRSET TO PUT NEW VERSION IN ENTRY
7 CALL OPEN, WRITF, CLOSE TO UPDATE LIBRARY DIRECTORY
8 ERROR IF FILE ERROR TO :FILEERR:
9 ENDIF
10 ENDIF
11 END VERSION
12 1 =
13 1 =
14 1 BEGIN DELPRM
15 2 SET RTN = 3

:PRON3:
16 2 DO UNTIL RETURN CODE IS CR ENTERED
17 3 IF NUMBER OF PARAMETERS > 1 THEN
18 4 CALL XCTOM TO PROMPT FOR DELETE PARAMETER NAME
19 4 IF RETURN CODE IS NOT CR ENTERED THEN
20 5 PERFORM RSPMD TO INTERPRET RESPONSE
21 5 PERFORM CHDAT TO CHECK FOR EXISTING DATA
22 5 SET ARGNO TH WORD IN CODES TO SAY 'DELETED'
23 5 CALL XRMOV TO MOVE DATA TO DELETE PARAMETER
24 5 DECREMENT # PARAMETERS BY 1
25 ENDIF
26 ELSE
27 4 CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
28 5 EXIT DELPRM
29 6 ENDF.
30 ENDIF
31 2 ENDDELPRM
32 1 =
33 1 =
34 1 BEGIN MODPRM
35 2 SET RTN = 4

:PRON4:
36 2 DO UNTIL RETURN CODE IS CR ENTERED
37 3 CALL XCTOM TO PROMPT FOR MODIFY PARAMETER NAME
38 3 IF RETURN CODE IS NOT CR ENTERED THEN
39 4 PERFORM RSPMD TO INTERPRET RESPONSE
40 4 PERFORM CHDAT TO CHECK FOR EXISTING DATA
41 4 SET ARGNO TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
42 4 CALL CLSPE TO PROMPT USER FOR SPECIFICATIONS
43 4 SET IT COMPLETE BIT OFF
44 4 ENDF.
45 2 ENDO.
46 1 END MODPRM
1. BEGIN ADPAM  
2. SET RTN = 5
3. PROMPT: RETURN COMPUTED VALUE ENTERED THEN RETURN
4. IF RETURN COMPUTED VALUES
5. THEN RETURN COMPLETED VALUE TO MEMORY
6. END IF
7. ELSE CALL XPAR TO DISPLAY NO MORE PARAMETERS CAN BE ADDED
8. END ADPAM

ELSE CALL XPAR TO PROMPT FOR ADD/MODIFY/DELETE DEFAULT VALUES
9. CALL XPAR FOR ADD/MODIFY/DELETE DEFAULT VALUES
10. IF RESPONSE VALUE IS NOT-ZERO THEN CALL INTERFACE TABLE EDITOR
11. EXIT ADPAM
12. END ADPAM

5=145
CD*********
1019  CD0        FORTRAN CALLING PROCEDURE
1020  CD0        CALL HELDS ('XLMSG ')
1022  CD0        **
1023  CD0        *********
1025  CD1        XLMSG PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XRMSG
1026  CD1        *
1027  CD1        *********
1028  CD2        **
1029  CD2        XE COMMON - LU, ISEQU, FLAGS, TOKENS, ICR
1030  CD2        **
1031  CD2        TERMINAL - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES
1032  CD2        UPDATE MODE, MESSAGE NUMBER AND TEXT
1033  CD2        **
1034  CD2        MESSAGE FILE - DIRECTORY AND OLD TEXT
1035  CD2        **
1036  CD2        *********
1037  CD3        OUTPUT
1038  CD3        **
1039  CD3        XE COMMON - COMBUF
1040  CD3        **
1041  CD3        MESSAGE FILE - DIRECTORY AND TEXT UPDATES
1042  CD3        *********
1043  CD4        LOCAL
1045  CD4        AREA - NUMERICAL AREA INDICATOR FOR MESSAGE
1046  CD4        **
1047  CD4        DIRECT - MESSAGE DIRECTORY (SEE SDD 6.2.4.12)
1048  CD4        **
1049  CD4        I - INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY
1050  CD4        **
1051  CD4        IDCB - FILE MANAGER DATA CONTROL BLOCK
1052  CD4        **
1053  CD4        IERR - FILE MANAGER $ XTOM RETURN CODE
1054  CD4        **
1055  CD4        NUMB - MESSAGE NUMBER WITHIN MESSAGE AREA
1056  CD4        **
1057  CD4        NTLBLK - BLOCK NUMBER WITHIN FILE
1058  CD4        **
1059  CD4        CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION
1060  CD4        **
1061  CD4        UPDATE MODE - NUMBER OF BLOCK CONTAINING MESSAGE
1062  CD4        **
1063  CD4        RECPOS - MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 35, 65 OR 97)
1064  CD4        **
1065  CD4        *********
1067  CD5        NOTES
1068  CD5        **
1069  CD5        USES APDIN, CLOSE, CREATE, EXEC, IAND, KCVT, OPEN, READF, WRITF,
1070  CD5        **
1071  CD5        XERIN, XRIG, XRMSD, XRMSG, XTOM, XUBGB
1072  CD5        **
1073  CD5        WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE
1074  CD5        **
1075  CD5        'EXISTING TEXT IN PLACE.'
1076  CD5        **
1077  CD5        MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X
1078  CD5        **
1079  CD5        **
1080  CD5        **
1081  CD5        **
1082  CD5        **
1083  CD5        **
1084  CD5        **
1085  CD5        **
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 LOCATION
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-NUL Response
1100   34              THEN
1101   35              STORE TEXT IN BLOCK
1102   36              REWRITE BLOCK
1103   37              IF MESSAGE NUMBER > LAST MESSAGE NUMBER
1104   38              THEN
1105   39              REPLACE LAST MESSAGE NUMBER WITH NEW NUMBER
1106   40              ENDIF
1107   41              ENDIF
1108   42      ELSE
1109   43          OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110   44      ENDIF
1111   45      ENDDO
1112   46      REWRITE DIRECTORY BLOCK
1113   47    END XLMSG
1149 1 BEGIN XLCR
1150 2 CALL CREATE TO CREATE MDB/PDB LOG FILE
1151 2 ERREXIT IF CREATE ERROR TO :FILEERR:
1152 2 INITIALIZE LOG RECORD BUFFER TO ZEROS
1153 2 SET # MDB FILES CURRENTLY USED TO ZERO
1154 2 SET MAXIMUM NUMBER MDBS TO 20
1155 2 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
1156 2 ERREXIT IF WRITE ERROR TO :FILEERR:
1157 2 DO FOR EACH REMAINING LOG RECORD
1158 3 DO FOR THIS PAIR OF USER ID'S
1159 4 CALL XTCOM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
1160 4 SET MAXIMUM # PDB FILES TO RESPONSE
1161 4 SET # PDB FILES CURRENTLY USED TO ZERO
1162 3 ENDDO
1163 3 CALL WRITE TO WRITE 1 PDB RECORD TO LOG FILE
1164 3 ERREXIT IF WRITE ERROR TO :FILEERR:
1165 2 ENDDO
1166 2 CALL CLOSE TO CLOSE FILE
1167 2 ERREXIT IF CLOSE ERROR TO :FILEERR:
1168 1 EXIT XLCR

1169 2 :FILEERR:
1170 2 CALL KRMSG TO DISPLAY FILE ERROR
1171 2 CALL CLOSE TO CLOSE FILE
1172 1 END XLCR
FORTRAN CALLING SEQUENCE:

CALL XLPFL (NOCOD, CODES)

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

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XE(1) LU, XE(3) ISECU, XE(16) PRCHNM,
XE(142) IRC, XB(96)NOPARN, XB(108)PARMS

RTE FUNCTIONS USED:

FDS FUNCTIONS USED:

XERTN, XRPCF, XIMOV, XRMSG,
XRCK, XRUP, XICOM

COMMON USED:

EQUIVALENCE (XE(1), LU),
+ (XE(3), ISECU), (XE(77), VALFLD),
+ (XE(16), PRCHNM), (XE(85), TKMEN),
+ (XE(142), IRC), (XE(144), COMPR),
+ (XE(145), COMBUS), (XB(96), NOPARN),
+ (XB(108), PARMS)
1227 1 BEGIN XLFL
1228 2 1 COME SIZE OF FILE AS 0 PARAMETERS + 3
1229 3 CALL CREAT TO CREATE PROMPT FILE
1230 4 ERRNEXIT IF CREAT ERROR TO :FILERR:
1231 5 STUFF SYNTAX RECORD 0 AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS
1232 6 CALL WITF TO WRITE SHORT PROMPT RECORD(S)
1233 7 ERRNEXIT IF WITF ERROR TO : F I L E R R :
1234 8 IF ABSTRACT CODE IS UTILITY THEN
1235 9 CALL OPEN TO OPEN OLD PROMPT FILE > X L T M P
1236 10 ERRNEXIT IF OPEN ERROR TO :FILERR:
1237 11 CALL REA D T O READ EXISTING ABSTRACT
1238 12 ERRNEXIT IF READ ERROR TO :FILERR:
1239 13 CALL ITCOM TO PROMPT USER TO MODIFY ABSTRACT
1240 14 IF RESPONSE IS CR (NO MODIFICATION) THEN
1241 15 CALL WITF TO WRITE EXISTING ABSTRACT TO NEW FILE
1242 16 ERRNEXIT IF WITF ERROR TO :FILERR:
1243 17 ELSE (NEW ABSTRACT WAS ENTERED)
1244 18 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1245 19 E N D I F
1246 20 ELSE (ABSTRACT CODE IS AD)
1247 21 CALL ITCOM TO PROMPT USER TO ENTER NEW ABSTRACT
1248 22 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1249 23 E N D I F
1250 24 E N D I F
1251 25 IF PARAMETERS IS NOT ZERO THEN
1252 26 PERFORM EXPRM TO GET DEFINITIONS FOR EACH PARAMETER
1253 27 E N D I F
1254 28 CALL CLOSE TO CLOSE PROMPT FILE
1255 29 ERRNEXIT IF CLOSE ERROR TO :FILERR:
1256 30 IF THERE WAS AN OLD FILE THEN
1257 31 CALL CLOSE TO CLOSE OLD FILE > X L T M P
1258 32 CALL PURGE TO PURGE OLD FILE > X L T M P
1259 33 E N D I F
1260 1 EXIT XLFL
1261 2 :FILERR:
1262 3 CALL X R M S G TO DISPLAY ERROR CODE
1263 4 CALL CLOSE TO CLOSE NEW FILE
1264 5 CALL CLOSE TO CLOSE OLD FILE > X L T M P
1265 6 CALL PURGE TO PURGE OLD FILE > X L T M P
1266 7 SET VALFLG = 0
1267 2 CALL X T R N TO RETURN TO MAIN ***NO RETURN TO HERE***
1268 1 END XLFL
1270 1 BEGIN FORMAT
1271 2 INITILIZE 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 ZMOV 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 1 CALL XRNSG TO DISPLAY INVALID RESPONSE
1287 1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
BEGIN EXTRMN
DO FOR EACH ENTRY IN CODES ARRAY
BEGIN CASE (:NOCHNG:, :MOD:, :DEL:, :ADD:), CODES+1

:NOCHNG:  (SPECs Had NO CHANGES)
CALL READ TO READ EXISTING DEFINITION FROM >XTMP
ERREXIT IF READF ERROR TO :FILERR:
CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
ERREXIT IF WRITF ERROR TO :FILERR:

:MOD:  (SPECs WERE MODIFIED)
CALL READ TO READ EXISTING DEFINITION FROM >XTMP
ERREXIT IF READF ERROR TO :FILERR:
CALL EXEC TO DISPLAY EXISTING DEFINITION
IF RESPONSE IS CR (NO RESPONSE) THEN
CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
ERREXIT IF WRITF ERROR TO :FILERR:
ELSE (NEW DEFINITION WAS ENTERED)
PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
ENDIF

:DEL:  (SPECs WERE DELETED)
CALL POSN TO POSITION >XTMP OVER THIS ENTRY
ERREXIT IF POSN ERROR TO :FILERR:

:ADD:  (SPECs WERE ADDED)
CALL XCTMN TO PROMPT USER TO ENTER NEW DEFINITION
PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
END CASE
DO UNTIL RESPONSE IS CR (EXIT)
CALL XCTMN TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
EXIT EXTRMN IF RESPONSE IS CR
SEARCH SHORT PROMPTS FOR RESPONSE
ERREXIT IF NOT FOUND TO :ERR:
CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
ERREXIT IF READF ERROR TO :FILERR:
CALL EXEC TO DISPLAY EXISTING DEFINITION
CALL XCTMN TO PROMPT USER TO MODIFY DEFINITION
IF RESPONSE IS POSITIVE
CALL POSN TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
ERREXIT IF POSN ERROR TO :FILERR:
PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
ENDIF
ENDIF
END EXTRMN
1387 1 BEGIN XLPMO
1388 2 DO UNL TIL USER RESPONDS EXIT (2)
1389 3 CALL XTOM 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 # ALLORED FOR THIS USER AND # CURRENTLY USED
1395 9 CALL XTOM 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 WRIT ERROR TO :FILERR:
1402 16 ELSE
1403 17 DISPLAY LIST OF CURRENT FILES
1404 18 SET SPURGED = 0
1405 19 DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1406 20 CALL XTOM 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 SPURGED BY 1
1412 26 DECREMENT DIFFERENCE BY 1
1413 27 COMPRESS OLD FILE NAME OUT OF LIST
1414 28 ENDDO
1415 29 ENDDO
1416 30 IF RESPONSE WAS NOT EXIT THEN
1417 31 CALL WRIT TO REWRITE UPDATED LOG RECORD
1418 32 ERREXIT IF WRIT ERROR TO :FILERR:
1419 33 DO FOR #SPURGED FILES
1420 34 CALL XTOM TO CREATE FILE NAME
1421 35 CALL PURGE TO PURGE FILE
1422 36 ENDDO
1423 37 ENDDIF
1424 38 ENDDIF
1425 39 ENDDIF
1426 40 ENDDO
1427 41 CALL CLOSE TO CLOSE NDB/PDB LOG FILE
1428 42 1 EXIT XLPMO
1429 2 :ERR1:
1430 2 CALL XMSG TO DISPLAY ERROR IN RESPONSE
1431 2 RETURN TO REISSUE LAST PROMPT
1432 2 :FILERR:
1433 2 CALL XMSG TO DISPLAY FILE ERROR
1434 2 CALL CLOSE TO CLOSE NDB/PDB FILE
1435 1 END XLPMO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

CLASS - (INTEGER, 1 WORD, XE(1)) IS THE PARAMETER CLASS
IPRT - (INTEGER, 1 WORD) IS THE INDEX INTO PARMS WHERE
THE SHORT PROMPT IS TO BE STORED
ISPEC - (INTEGER, 1 WORD) IS THE INDEX INTO PARMS WHERE
THE SPEC 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, XRCP, XRMOV, XRMSG, XRSET, XTOM

XE COMMON USED:

EQUIVALENCE (XE(8), RETC, XE(9), RETN),
+ (XE(10), CLASS, XE(11), TYPE ),
+ (XE(145), COMBUF, XE(37), ARPNO ),
+ (X9(96), NOPARM, XE(108), PARMS )
BEGIN XLSPS
  CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
  
  :PRMT1:
  SET RETURN TO 1
  CALL KCOM TO PROMPT FOR PROMPT,CLASS,TYPE AND I/O FLAGS
  ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
  INITIALIZE THIS SHORT PROMPT AND ENTRY
  ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
  ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
  CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
  ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
  CALL XRES TO SET CLASS IN ENTRY
  ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
  ERREXIT IF CLASS IS ORDER AND TYPE IS SYMBOLIC STRING TO :PRMERR:
  CALL XRES TO SET TYPE IN ENTRY
  ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
  ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
  CALL XRES TO SET I/O FLAGS IN ENTRY
  IF CLASS IS DATUM ELEMENT THEN
    IF TYPE IS NOT SYMBOLIC STRING THEN
      SET RETURN TO 2
      :PRMT2:
      CALL KCOM TO PROMPT FOR I AND J DIMENSIONS
      ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
      ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
      I: J DIMENSION IS NOT ENTERED THEN
      SET J DIMENSION TO 1
      ELSE
      ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
      IF J DIMENSION < 2 THEN
      SET J DIMENSION TO 1
      ELSE
      STORE I DIMENSION IN ENTRY
      ENDIF
      IF SIZE = I DIMENSION * J DIMENSION * TYPE LENGTH
      ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
      ELSE
      SET RETURN TO 3
      END:}
    ELSE
    :PRMERR:
    CALL XRMSE TO DISPLAY ERROR MESSAGE
    GO TO (:PRM1:, :PRMT2:, :PRMT3), RETURN
    END XLSPS
**FDU EXECUTION**

1 * D5: EXECUTION, SCHEDULED BY FDS CONFIGURATION MANAGER. ONE PROGRAM
2 * D5: PER SIGNED ON USER.
3 **---------- INPUT**
4 * D2: FDS STATUS TABLE (SEE SDD 6.2.2)
5 * D2: FDS MANAGEMENT REQUESTS (SEE SDD 6.2.6.2)
6 **---------- OUTPUT**
7 * D3: FDS MANAGER RESPONSES (SEE SDD 6.2.6.3)
8 **---------- NOTES**
9 * D5: USES XHNG, XHVA, XMPN, XMSN, XMST, XDMP
10 RMPN, SLIBN, SLIB, RNR, CUMN, XEE
11 * D5: XHVA IS A MANAGER GLOBAL DATA AREA CONTAINING THE AWA AND VARIOUS
12 * D5: CONTROL VALUES
13 **----------
14 1 *
15 1 *
16 BEGIN XHGR
17 * D5: INPUTS: P1 = LU
18 * D5: P2 = XVSTB ENTRY ADDRESS
19 * D5: P3 = USER ID
20 * D5: P4 = OPTIONS
21 * D5: P5 = NUMBER OF DWA TRACKS
22 CALL RM?AR
23 LOCK THE MANAGER INTO THE PARTITION
24 SET XVSTA (FDS STATUS TABLE ENTRY ADDRESS)
25 RESOLVE XVSTB ADDRESS
26 CALL XM?IN INITIALIZATION Functions
27 IF ERROR RETURN THEN
28 EXIT TO XRENDB
29 ENDIFF
30 * D5: GET EXEC'S CLASS NUMBER
31 GET PROCESSOR'S CLASS NUMBER
32 CALL S?LB GET PRIVILEGED
33 SET EXEC'S CLASS NUMBER IN STBEC
34 SET PROCESSOR'S CLASS NUMBER IN STBP
35 SET CURRENT TASK IN STBEC FROM EXEC ID STBEC
36 CALL SL?BN ENABLE
37 CALL EXEC'S NAME FROM 'X?Y' & ASCII LU(STBRA)
38 SCHEDULE EXEC WITH WAIT
39 CALL XM?SN
40 DO UNTIL EXEC REQUESTS TERMINATION OR ABENDS(P1=9 OR -32768)
41 IF REQUEST IS IN P1
42 IF REQUEST IS FOR AWA MANAGEMENT (P1=1) THEN
43 PERFORM X?HNG
44 ELSE
45 IF REQUEST IS FOR SEQUENCE TABLE EXECUTION (P1=2) THEN
46 PERFORM 'X?OS
47 ELSE
48 IF REQUEST NOT TERMINATION (P1=9 OR -32768) THEN
49 ISSUE MESSAGE '*** XHGR INVALID REQUEST (II) FROM XEN"
61 6 * II=REQUEST & WW IS LU
62 6 * SET UP FOR FDS TERMINATION
63 5 EXIT TO :XMEND
64 5 ENDIF
65 4 ENDIF
66 3 ENDIF
67 2 ENDDO
68 2 :XMEND
69 2 IF DUMP OPTION THEN
70 3 CALL XUDMP TO DUMP OUR PARTITION
71 2 ENDIF
72 2 RELEASE DWA TRACK SPACE
73 2 FREE EXEC'S & PROCESSOR'S CLASS NUMBERS
74 2 CALL RRNR SET LOCK ON STATUS TABLE
75 2 UNLOCK THE MANAGER FROM THE PARTITION
76 2 CALL SLIBR DISABLE
77 2 CLEAR EXEC'S ID SEGMENT
78 2 CALCULATE EQT ADDRESS FOR ATTENTION ID
79 2 CLEAR THE STATUS TABLE ENTRY
80 2 CALL SLIBX ENABLE
81 2 CALL RRNR CLEAR THE LOCK ON THE STATUS TABLE
82 2 ISSUE MESSAGE '***XMDZ SIGN OFF FOR LU "WW"'
83 2 CALL SLIBX GET PRIVLEDGED AND DISABLED
84 2 CALL SLIST TO MAKE US DORMANT
85 2 CLEAR OUR OWN ID
86 2 RESET EQT TO REMOVE ATTENTION ID
87 1 EXIT :SEZEQ TO THE RTE DISPATCHER
88 1 END XMGRT
BEGIN XMGST
CALL XMFSM TO FIND BSEGTB TOC ENTRY
CALL XMSSM TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
CALL XMSTT TO CONVERT STARTING SEQUENCE NUMBER INTO CURRENT DISPLACEMENT
COMPUTE CURRENT ENTRY ADDRESS
DO UNTIL THE LAST TABLE ENTRY IS EXECUTED OR
UNTIL THE TERMINATION ENTRY IS EXECUTED
IF PROCESSOR REQUIRES AN INTERFACE TABLE (WORD 3 BIT 8 IS SET) THEN
EXIT TO :ERROR: IF INTERFACE TABLE NOT SPECIFIED (WORD 4 = 0) (PARMS = 1)
EXIT TO :ERROR: IF INTERFACE TABLE NOT FOUND (CHAIN 4) (PARMS = 2)
EXIT TO :ERROR: IF TABLE NOT IN AWA, THEN
CALL INDRT TO RETRIEVE FROM AWA
EXIT TO :ERROR: IF SPACE IS NOT FOUND
ENDIF
EXIT TO :ERROR: IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PARMS = 3)
EXIT TO :ERROR: IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PARMS = 4)
EXIT TO :ERROR: IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PARMS = 5)
ENDIF
SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED
EXIT TO :ERROR: IF NOT FOUND
CALL SLIBR TO BE PRIVILEGED
SET CURRENT TASK TO THIS PROCESSOR
CALL SLITB TO BE UN-PRIVILEGED
IF PROCESSOR REQUIRES AN INTERFACE TABLE THEN
CALCULATE LENGTH OF INTERFACE TABLE HEADER AND SPEC
CALL EXEC TO CLASS I/O WRITE HEADER AND SPEC
ENDIF
CALL EXEC TO SCHEDULE PROCESSOR WITH WAIT
CALL XRSC TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XMFSM TO FIND BSEGTB TOC ENTRY
FIND ADDRESS OF BSEGTB
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARM = 1)
CALL XHANG TO HONOR AWA REQUEST
ENDD
CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
EXIT TO :ERROR: IF PROCESSOR REQUESTED TERMINATION (PARM = 8)
EXIT TO :ERROR: IF PROCESSOR ABENDED (PARM = -32768)
IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PARM = 3) THEN
EXIT TO :ERROR: IF RESET NUMBER IS ZERO
CALL XMSTT TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY
EXIT TO :ERROR: IF RESET SEQUENCE NUMBER IS NOT FOUR
IF TERMINAL ENTRY WAS JUST EXECUTED THEN
SET UP PARMS TO SHOW RESET SEQUENCE NUMBER
PERFORM :END: = **NO RETURN EXPECTED**
ENDIF
ELSE
IF REQUEST IS NOT NORMAL COMPLETION (PARM = 0) THEN
DISPLAY ERROR MESSAGE - INVALID REQUEST
EXIT TO :ERROR: IF CURRENT AT IS DORMANT
CALL XMXIL TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAM TO RESCHEDULE PROCESSOR
EXIT TO :ERROR: TO TERMINATE SEQUENCE
ENDIF
ENDD
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDD
171  SET PARM1 = 0 (NORMAL COMPLETION)
172  PERFORM :END: - NO RETURN EXPECTED
173  :ERROR1:  SET PARM1 = 1
174  :ERROR2:  SET PARM1 = 8 AND PARM5 TO APPROPRIATE REASON CODE
175  :END:
176  CALL SLIBR TO BECOME PRIVLEDGED
177  SET CURRENT TASK IN MGR, STATUS TABLE, AND ANA TO EXEC
178  CALL SLIBX TO BECOME UN-PRIVLEDGED
179  CALL XMPAV TO POST EXEC AND WAIT FOR NEXT REQUEST
180  END XMKQT
CALLING PROCEDURE

JSB XMAFR
DEF ADD
DEF SIZE

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

INPUT

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

EXTERNAL SYMBOLS FROM XMAWA
XMBCP, XMFCP, XMFCN, XMFFE

OUTPUT (EXTERNAL SYMBOLS FROM XMAWA)
XMBCP, XMFCP, XMFCN, XMFFE

NOTES
USES .ENTR
1 BEGIN XMAFR
2 SET NEW FE SIZE FIELD TO MAX(SIZE, 3)
3 INCREMENT TOTAL FREE SPACE BY NEW FE SIZE
4 INDEX TO FORWARD CHAIN POINTER (FCP) HEAD
5 START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)
6 EXIT IF 'ADDR' < FCP VALUE
7 SET NEW FE FCP TO CURRENT FCP VALUE
8 SET CURRENT FCP VALUE TO 'ADDR'
9 SET NEW FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE
10 SET NEXT FE'S BCP VALUE TO 'ADDR'
11 PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE
12 OR ELSE
13 INDEX TO NEXT FE FCP
14 END LOOP
15 SET NEW FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
16 SET CURRENT FE'S FCP VALUE TO 'ADDR'
17 SET NEW FE BCP TO BCP HEAD VALUE
18 SET BCP HEAD TO 'ADDR'
19 END SEARCH
20 IF NEW FE BCP NOT = END-OF-CHAIN
21 THEN
22 PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
23 END IF
24 IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW
25 (OR CONSOLIDATED) FE, THEN
26 MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
27 ENDIF
28 END XMAFR
29 1 BEGIN MERGE
30 IF FE 1 IS ADJACENT TO FE 2
31 THEN
32 INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
33 SET FE 1 FCP TO VALUE OF FE 2 FCP
34 IF FE 1 FCP NOT = END-OF-CHAIN
35 THEN
36 SET FE 3 BCP TO ADDRESS OF FE 1
37 ELSE
38 SET BCP HEAD TO ADDRESS OF FE 1
39 ENDIF
40 ENDIF
41 END MERGE
255 1 000 CALLING PROCEDURE
256 1 000 JSB XNAGT
257 1 000 DEF +43
258 1 000 DEF OPTM
259 1 000 DEF SIZE
260 1 000 DEF Sizen
261 1 000 ************
262 1 001 FIND A BLOCK OF FREE SPACE IN THE AWA AT LEAST SIZE WORDS LARGE
263 1 001 ************
264 1 001 INPUT
265 1 002 OPTM - INDICATOR OF WHICH FREE CHAIN TO SEARCH
266 1 002 D = FORWARD POINTER CHAIN (FOR TDC SPACE)
267 1 002 1 = BACKWARD POINTER CHAIN (FOR DATA SPACE)
268 1 002 SIZE = NUMBER OF WORDS NEEDED (A MINIMUM OF 3 WORDS WILL BE
269 1 002 ALLOCATED EVEN IF SIZE IS 1 OR 2)
270 1 002 EXTERNAL SYMBOLS FROM XNAM
271 1 002 XMFC, XMFC, XMFC
272 1 002 ************
273 1 003 OUTPUT
274 1 003 A-REG = ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
275 1 003 INDICATING NONE AVAILABLE
276 1 004 EXTERNAL SYMBOLS FROM XNAM
277 1 004 XMFC, XMFC, XMFC
278 1 004 ************
279 1 005 NOTES
280 1 005 IF SOME FE EXACTLY SIZE WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
281 1 005 ERROR RETURN (A-REG = -32768) IS TAKEN
282 1 005 USES .ENIR
BEGIN XNAGT
2 IF TOTAL FREE SPACE < MAX(SIZE,3)
3 THEN
4 CALL XNPK1 PURGE DNA ELEMENTS FROM ANA
5 SET PHASE1_FLAG
6 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
7 RETURN VALUE= -32768
8 EXIT TO :XNAGR
9 ENDIF
10 ENDIF
11 PERFORM Xmerce(OPTy,SIZE)
12 IF RETURN CODE IS NOT FOUND, THEN
13 CALL XNPK
14 IF RETURN CODE IS NOT FOUND, THEN
15 RETURN VALUE= -32768
16 EXIT TO :XNAGR
17 ENDIF
18 RETURN VALUE= AREA
19 :XNAGR
20 ENDIF
21 CLEAR PHASE1_FLAG
22 END XNAGT
318 1 BEGIN XMSRC
319 2 SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FHEAD(OTP1)
320 3 START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
321 4 WHILE TOC SPACE FENCE HAS NOT BEEN CROSSED
322 5 EXIT IF FE SIZE = MAX('SIZE', 3)
323 6 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
324 7 DECHAIN FE
325 8 RETURN ADDRESS OF AREA
326 9 EXIT IF FE SIZE >= MAX('SIZE', 3) + 3
327 10 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
328 11 IF ALLOCATING FROM HEAD OF SPACE (OTP1 = 0)
329 12 THEN
330 13 CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
331 14 RECHAIN NEW FE
332 15 RETURN ADDRESS OF AREA
333 16 ELSE
334 17 CHANGE SIZE FIELD TO FE SIZE - MAX('SIZE', 3)
335 18 COMPUTE AND RETURN ADDRESS OF AREA
336 19 ENDIF
337 20 OR ELSE
338 21 INDEX TO NEXT FE
339 22 END LOOP
340 23 SET RETURN CODE TO 0('NOT FOUND')
341 24 END SEARCH
342 25 IF TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
343 26 INCREMENT TOC SPACE FENCE OR 'SIZE'
344 27 ENDIF
345 28 END XMSRC
CALLING PROCEDURE

***
1 *00 JSB XMAPG
1 *00 DEF CLNID
1 *00 ***
1 *01 PROVIDE AMA MANAGEMENT BASED ON REQUEST LIST (SEE FDS SDD
1 *01 TABLE 6.2 - III)
1 *01 ***
1 *02 INPUT
1 *02 CLNID - CLASS I/O NUMBER CONTAINING REQUEST LIST
1 *02 ***
1 *02 OUTPUT
1 *02 REQUEST LIST (SEE TABLE 6.2 - III)
1 *02 ***
1 *03 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
1 *03 ***
1 *03 REQUEST LIST FIELD 8
1 *03 ***
1 *05 NOTES
1 *05 ***
1 *05 USES EXEC,XMAFR,XMAGT,XMAPK,XMIN
1 *05 XMDIN,XMDAL,XMDOA,XMDST,XMDXT
1 *05 ***
1 *05 NEITHER EXEC NOR THE PROCESSORS WILL MAKE AMA MANAGEMENT REQUESTS
1 *05 FOR DWA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DWA
1 *05 ***
1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
1 *05 ***
1 *05 THE DWA AND COPY THEM BACK TO THE AMA AS NECESSARY.
1  BEGIN XMANG
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 XMAPK TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
11         ENDIF
12      WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O
13      STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT
14      SET REQUESTED SIZE FROM TOC SIZE
15      SET RETURN PARM1
16      ELSE
17          IF REQUESTS TO CLEAR (CODE 17)
18          THEN
19             GET A(&DWA) FROM XMDWA
20             SAVE HEADER AND DIRECTORY-INDEX
21             CLEAR XMDWO THRU XMDWF
22             BUILD AN FE AT XMDWA FOR AWA SIZE
23             CALL XMAST TO ALLOCATE A TOC ENTRY FOR &DWA
24             CHAIN IN TOC ENTRY TO &MDWO
25             CALL XMAST TO ALLOCATE SPACE FOR &DWA
26             SET LOCATION, SIZE, & KEY IN THE TOC
27             SET DIRECTORY ADDRESS AT XMDWA
28             CLEAR THE DIRECTORY
29             SET UO, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY
30             ELSE
31                 CALL XMFRN TO SEARCH TOC FOR INDICATED ENTRY
32                 CASE (:VERIFY:, :VERALO:, :VERALO:, :RNAME:, :DELVER:, :DELVER:, :STORE:,
33                                       (:RETRVE:, :RETRVE:) REQUEST CODE
34                      :VERIFY:
35                      IF ENTRY NOT FOUND
36                      THEN
37                          SET RETURN PARM1 AND PARM2 (2 & INDEX)
38                      ENDIF
39                      :VERALO:
40                      IF ENTRY ALREADY EXISTS
41                      THEN
42                          IF ALLOCATE REQUEST (33)
43                              THEN
44                                  SET RETURN PARM1 AND PARM2 (3 & INDEX)
45                              ELSE
46                                  IF TYPE, SIZE AND I-DIM FIELDS DO NOT MATCH
47                                  THEN
48                                      SET RETURN PARM1 AND PARM2 (4 & INDEX)
49                                  ENDIF
50                                  ELSE
51                                  END
52                      ELSE
53                          CALL XMAST TO ALLOCATE TOC SPACE
54                          IF CLASS EQ 3 OR 8, THEN
55                              CHAIN IN NEW TOC ENTRY
56                              SET DATA SPACE ADDRESS TO ZERO
57                          ELSE
58                              CALL XMAST TO ALLOCATE DATA SPACE
59                      END
IF SPACE NOT AVAILABLE
THEN
SET RETURN PARM1 AND PARM2 (1 & INDEX)
ELSE
IF DATA ELEMENT (CLASS 2)
THEN
IF CHARACTER STRING (TYPE 4-8)
THEN
INITIALIZE AREA TO BLANKS
ELSE
INITIALIZE AREA TO ZEROS
ENDIF
ENDIF
CHAIN IN NEW TOC ENTRY
ENDIF
IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE
THEN CALL XMDDA DWA ALLOCATION
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDER
ENDIF

:STORE:

IF ENTRY NOT FOUND

THEN

SET RETURN PARM1 AND PARM2 (2 & INDEX)

ELSE

IF (TOC TYPE > 0 AND INCONSISTENT WITH REQUEST TYPE) OR

DISPLACEMENT OR SPECIFIED SIZE < 0, OR

DISPLACEMENT + REQUESTED SIZE > ALLOCATED SIZE

THEN

SET RETURN PARM1 AND PARM2 (4 & INDEX)

ELSE

GET DATA FROM INDICATED CLASS I/O; STORE INTO AMA

FREE CLASS NUMBER

IF CLASS EQ 4 OR 6, THEN

CALL XMST DMA STORE DATA

ENDIF

ENDIF

:RETRIEVE:

IF ENTRY NOT FOUND

THEN

SET RETURN PARM1 AND PARM2 (2 & INDEX)

ELSE

IF VALUES REQUESTED (8)

THEN

IF (TOC TYPE .NE. 0 AND .NE. REQUESTED TYPE) OR

DISPLACEMENT OR SPECIFIED SIZE < 0, OR

DISPLACEMENT + SPECIFIED SIZE > ALLOCATED SIZE

THEN

SET RETURN PARM1 AND PARM2 (4 & INDEX)

ELSE

IF REQUESTED SIZE = ZERO

THEN

CALCULATE AMOUNT OF DATA TO RETRIEVE AS ACTUAL SIZE MINUS DISPLACEMENT

STORE COMPUTED SIZE IN REQUEST WORD SIX

ENDIF

IF CLASS EQ 4 OR 6 AND TOC ADDRESS EQ 0, THEN

THE ELEMENT EXISTS ONLY ON THE DMA

CALL XMST MOVE INTO AMA

IF NO SPACE THEN

SET RETURN PARM1 AND PARM2 TO(1, INDEX)

EXIT TO $XMREX

ENDIF

ENDIF

ENDIF

WRITE VALUES TO CLASS I/O

STORE TYPE IN LOW BYTE OF REQUEST WORD 1

STORE CLASS NUMBER IN REQUEST WORD 8

SET NEWRITE FLAG

ENDIF

ELSE

WRITE TOC ENTRY TO CLASS I/O

STORE CLASS NUMBER IN REQUEST WORD EIGHT

SET NEWRITE FLAG

ENDIF

$XMREX

ENDIF
CALLING PROCEDURE

1.80      JSB XMTFM
1.81      DEF #2
1.82      DEF KEYS
1.83      ********
1.84      #1      EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'
1.85      #1      INPUT
1.86      #1      KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC
1.87      #1      #2      EXTERNAL SYMBOLS FROM XMD.
1.88      #1      #2      XMD
1.89      #1      #2      OUTPUT
1.89      #1      A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR
1.89      #1      ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY
1.89      #1      WHERE 'KEYS' COULD BE CHAINED IN
1.89      #1      #3      BEGIN XMTFM
1.89      #1      #3      ISOLATE CLASS FROM KEY AND INDEX "0 A-PROPRIATE CHAIN HEAD
1.89      #1      #3      START SEARCH WHILE CHAIN POINTER #0 = END-OF-CHAIN (-32768)
1.89      #1      #3      COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY
1.89      #1      #3      EXIT IF MATCH
1.89      #1      #3      RETURN ADDRESS OF ENTRY
1.89      #1      #3      EXIT IF 'KEYS' < TOC ENTRY (EXPECTED ENTRY NOT IN CHAIN)
1.89      #1      #3      RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET
1.89      #1      #3      OR ELSE
1.89      #1      #3      INDEX TO NEXT TOC ENTRY ON CHAIN
1.89      #1      #3      END LOOP
1.89      #1      #3      RETURN ADDRESS OF LAST (PREVIOUS) ENTRY WITH INDIRECT BIT SET
1.89      #1      END XMTFM
CALLING PROCEDURE

JSB XMSST

ADJUST SEQUENCE LOCATION COUNTER

*********

INPUT

A-REG - INTEGER VALUE OF SEQUENCE NUMBER TO BE LOCATED

EXTERNAL SYMBOLS

SERAD - ADDRESS OF EXECUTING SEQUENCE TABLE IN XWA

ENDTB - DISPLACEMENT TO LAST ENTRY OF EXECUTING SEQUENCE TABLE

*********

OUTPUT

A-REG - DISPLACEMENT TO ENTRY IF SEQUENCE NUMBER IS FOUND, ELSE -32768

*********

BEGIN XMSST

SAVE RESET SEQUENCE NUMBER IN LOCAL VARIABLE SEGNO

SET DISPLACEMENT IN A-REG TO ZERO

GET ADDRESS OF FIRST ENTRY IN SEQUENCE TABLE INTO X-REG

START SEARCH UNTIL CURRENT ENTRY DISPLACEMENT > LAST ENTRY DISPLACEMENT

EXIT IF INDICATED NUMBER FOUND

OR ELSE

INCREMENT DISPLACEMENT IN A-REG TO NEXT ENTRY

END LOOP

SET A-REG TO -32768 TO INDICATE NUMBER NOT FOUND

END SEARCH

END XMSST
649 1 BEGIN  XMPAW
650 2 *00 ENTRY: JSB XMPAW OR CALL XMPAW
651 2 *02  DEF RETURN ADDRESS
652 2 *02  DEF RETURN ADDRESS
653 2 *02  DEF RETURN ADDRESS
654 2 *02  DEF RETURN ADDRESS
655 2 *02  DEF RETURN ADDRESS
656 2 *02  DEF RETURN ADDRESS
657 2 *02  DEF RETURN ADDRESS
658 2 *02  DEF RETURN ADDRESS
659 2 *02  DEF RETURN ADDRESS
660 2 *02  DEF RETURN ADDRESS
661 3  DO UNTIL GOOD PAKMS RECEIVED OR TOP AT TERMINATES
662 4  IF ABORT CURRENT FLAG CM (- STBLU) THEN
663 4  SET ABORT FLAG IN XMPAW
664 4  TURN OFF ABORT CURRENT FLAG
665 4  ENDIF
666 3  GET CURRENT TOP ASSOCIATED TASK(AT)
667 3  SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
668 3  JSB XVPAR
669 3  DEF *+3 RETURN
670 3  DFC 0 MANAGER CALL
671 3  DEF XMPAW PARM FIELD
672 3  THIS IS AN IMPLIED WAIT
673 3  XMSCH GET CURRENT XVSTO ENTRY (XUSTA)
674 4  IF THERE HAS BEEN A CALL TO PRTN (P1 FIELD IS NOT TOP AT) OR
675 4  IL SEGMENT IS DORMANT OR
676 4  ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
677 4  IF PARM IS NOT 0,3,8,9, OR -32768 THEN
678 5  SET PARM TO -32768
679 5  PRINT ERROR "INVALID REQUEST"
680 5  ELSE
681 5  SET PARM FIELD FROM MANAGER'S ID SEGMENT
682 5  ENDIF
683 5  ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PAK)
684 5  IF PARM NOT 1 OR 2 THEN
685 5  CALL XMIL TO AORPT TOP AT
686 5  PRINT ERROR "INVALID REQUEST"
687 5  ELSE
688 5  SET PARM FROM CURRENT ID SEGMENT
689 5  ENDIF
690 2 ENDDO
691 1 END  XMPAW
CALLING PROCEDURE

FUNCTION

INITIALIZE TOC

& DMA FUNCTIONS

OUTPUT

BREG D=COMPLETE

MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

MTFN, XMAG

BEGIN XMIDIN

GET NUMBER OF DMA TRACKS FROM PS

CALCULATE SIZE OF EDWA(3-6H) N IS # OF TRACKS

CALL XMTFN (EDWA)

CALL XMAG (0,8) TOC ENTRY FOR EDWA

CALL XMAG (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 '***XMIDN 'N' TRACKS NOT AVAILABLE'

ERROR RETURN

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DMA MANAGEMENT

ENDIF

END XMIDIN
1 BEGIN XNBAL
2 * DWA ALLOCATION
3 * CALLING PROCEDURE
4 JSB XNBAL
5 *
6 FUNCTION
7 * ALLOCATE DWA SPACE
8 *
9 INPUTS
10 TOC ENTRY OF THE ANA ELEMENT
11 *
12 IN TREG
13 *
14 OUTPUTS
15 UPDATE TOC ENTRY FOR DWA
16 *
17 RETURNS BREG= ZERO- ALLOCATION COMPLETE
18 = MINUS- ERROR CONDITION
19 *
20 NOT'S
21 *
22 CALLS XNR2, XNBDT
23 *
24 IF (DWA) .NE. 0 .THEN
25 SET START-ADDRESS TO FIRST TRACK WORD IN DWA DIRECTORY
26 GET SIZE(IN WORDS FROM TOC ENTRY)
27 SIZE(IN SECTORS) = (SIZE+63/64)
28 DO FOR NUMBER OF TRACKS IN DWA OR DWA SIZE .GE. SIZE
29 SET BITNUM TO ZERO
30 DO UNTIL DWA SIZE .GE. SITE OR BITNUM .GE. 96
31 CALL XNR2 (0,BITNUM,START-ADDRESS)
32 STARTBIT =BITNUM
33 CALL XNBRB (1,BITNUM,START-ADDRESS)
34 DMA-SIZE=BITNUM-STARTBIT
35 END-DO
36 IF BITNUM .GE. 96 .THEN
37 START-ADDRESS=START ADDRESS+1 TRACK ADDRESS
38 ENDIF
39 *
40 EMDO
41 *
42 IF D2='C' AREA FOUND, THEN
43 * START-A='F'S & BITNUM DEFINES THE TRACK & SECTOR
44 SET DISC ADDRESS & SIZE IN TOC ENTRY
45 CALL XNBR (TOC-ENTRY)
46 SET NORMAL RETURN
47 ELSE
48 ELSE
49 **XNBR NO DWA SPACE REMAINING**
50 ISSUE MESSAGE **XNOS NO DWA SPACE REMAINING**
51 SET ERROR RETURN
52 ENDF
53 2 ENDF
54 1 END XNBAL.
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
BEGIN XHOST
2+ DWA STORE DATA
2+ CALLING PROCEDURE
2+ JSB XHOST
2+ FUNCTION
2+ UPDATE DWA ELEMENT WHICH CORRESPONDS
2+ TO THE ANA ELEMENT
2+ INPUTS
2+ TOC ENTRY ADDRESS OF ANA ELEMENT
2+ IN TREG
2+ OUTPUTS
2+ UPDATE DWA ELEMENT ON DISC TRACKS
2+ NOTES
2+ USES WRITE
2+ IF A(EDUA) ME Q, THEN
2+ SET DISC ADDRESS FOR THE WRITE
2+ SET DATA ADDRESS FOR THE WRITE
2+ WRITE DATA
2+ ENDF
1 END XHOST
BEGIN XSHORT
  
  DATA RETRIEVE CALLING PROCEDURE
  JSR XSHORT
  
  FUNCTION
  RETRIEVE DATA INTO AVA
  
  INPUTS
  A(TOC ENTRY) IN YREG
  
  OUTPUTS
  ADDRESS OF DATA IN THE TOC
  REG=0, RETRIEVE SUCCESSFUL
  REG=1, RETRIEVE ERROR
  REG=2, RETRIEVE NO DATA
  
  NOTES
  USES XMAST, XNOWA, EXEC(READ)
  
  IF NO DATA DIRECTORY, THEN
  SET ERROR CODE 3
  
  ELSE
  CALL XMAST, GET DATA SPACE
  IF NO SPACE, THEN
  SET ERROR CODE 1
  
  ELSE
  SET DATA ADDRESS IN TOC
  GET DISC ADDRESS
  READ DATA INTO AVA
  SET RETURN CODE TO 0
  
  ENDIF
  
  END XSHORT
862 1 BEGIN XMBST
864 2 DO 0 DO XMBST
865 2 DO 0 JSB XMBST
866 1 FUNCTION
867 1 SET ALLOCATION & DEALLOCATION IN DWA DIRECTORY
868 1 INPUTS
869 2 DO 2 AWA TOC ENTRY
870 2 DO 2 OUTPUTS
871 2 DO 3 UPDATES DWA DIRECTORY
872 2 DO 5
873 2 DO 5 GET TRACK-ADDRESS FROM TOC
874 2 DO 5 SUBTRACT START OF TRACKS FROM DWA DIRECTORY FOR RELATIVE TRACK
875 2 DO 5 DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)
876 2 DO 5 QUOTIENT IS NUMBER OF RELATIVE WORDS
877 2 DO 5 REMAINDER IS BIT-POSITION(BP)
878 2 DO 5 WORD ADDRESS=DWA ADDRESS-RELATIVE TRACK+RELATIVE WORDS
879 2 DO 5 SUBTRACT BIT-POSITION FROM 16 GIVING NBFM(NUMBER BITS IN FIRST WORD)
880 2 DO 5 GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)
881 2 DO 5 BITS-NBFM=REMAINING BITS(RB)
882 2 DO 5 IF RB =E. O, THEN
883 2 DO 5 SET NUMBER OF WORDS(MNW) TO ZERO
884 2 DO 5 SET NUMBER OF BITS LAST WORD(MBLW) TO ZERO
885 2 DO 5 SET NBFM TO BITS
886 2 DO 5 ELSE
887 2 DO 5 DIVIDE RB BY 16
888 2 DO 5 SET NW TO QUOTIENT
889 2 DO 5 SET NBLW TO REMAINDER
890 2 DO 5 ENDIF
892 2 DO 5 LLOAD DATA POINTED TO BY WORD ADDRESS
893 2 DO 5 ROTATE LEFT (BP+NBFM-1 BITS) SAVE BP BITS & POSITION NBFM BITS
894 2 DO 5 EXCLUSIVE OR SIGN BIT (ON TO OFF; OFF TO ON)
895 2 DO 5 IF NBFM .LT. 1, THEN
896 2 DO 5 SHIFT RIGHT (ARITHMETIC) BY NBFM-1 PROPAGATE BITS
897 2 DO 5 ENDIF
898 2 DO 5 IF DATA IS NEGATIVE, THEN
899 2 DO 5 SET FILL WORD TO -1
900 2 DO 5 ELSE
901 2 DO 5 SET FILL WORD TO 0
902 2 DO 5 ENDIF
904 2 DO 5 ROTATE RIGHT BY BP, RESET SAVED BITS
905 2 DO 5 STORE WHERE WORD-ADDRESS POINTS
906 2 DO 5 DO WHILE NN .GE. 0
907 2 DO 5 WORD-ADDRESS=WORD-ADDRESS+$
908 2 DO 5 SET FILL WORD INTO WHERE WORD-ADDRESS POINTS
909 2 DO 5 ENDIF
910 2 DO 5 IF NBFM .LT. 0, THEN
911 2 DO 5 WORD-ADDRESS=WORD-ADDRESS+
912 2 DO 5 LOAD DATA POINTED TO BY WORD ADDRESS
913 2 DO 5 IF NBFM .LT. 1, THEN
914 2 DO 5 ROTATE LEFT NBLW-1 BITS
915 2 DO 5 ENDIF
916 2 DO 5 EXCLUSIVE OR SIGN BIT
917 2 DO 5 IF NBFM .LT. 1, THEN
918 2 DO 5 SHIFT RIGHT ARITHMETIC BY NBLW-1
919 2 DO 5 ENDIF
920 2 DO 5 STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00
3 *00  CALLING PROCEDURE
4 *00  JSB XMAPK
5 *00  DEF OPTION
6 *00  DEF PHASE 1 FLAG
7 *00  GENERAL COLLAPSE INTERFACE
8 *00
9 *00  OUTPUT
10 *00  RETURNS AREA ADDRESS FOR AREA FOUND
11 *00  RETURNS -32768 AREA NOT FOUND
12 *00
13 *00  NOTES
14 *00  USES XMPK1,XMPK2,XMPK3,XMSRC
15 *00
16 2 IF PHASE1 FLAG NOT SET, THEN
17 3 CALL XMPK3  PURGE DWA ELEMENTS FROM AWA
18 3 PERFORM XMSRC(OPTN,SIZE)
19 2 ELSE
20 3 SET RETURN CODE TO NOT FOUND
21 2 ENDIF
22 2 IF RETURN CODE IS NOT FOUND, THEN
23 3 IF OPTN=1(BACKWARD CHAIN), THEN
24 4 CALL XMPK2  PACK TDC CHAIN
25 4 CALL XMSRC(OPTN,SIZE)
26 3 ENDIF
27 3 IF RETURN CODE IS NOT FOUND, THEN
28 4 CALL XMPK3  PACK AWA DATA AREAS
29 4 PERFORM XMSRC(OPTN,SIZE)
30 3 ENDIF
31 2 ENDIF
32 2 SET RETURN VALUE TO RETURN CODE
33 1 END XMAPK
BEGIN XMPK1
* CALLING PROCEDURE
JSB XMPK1

PHASE 1 OF COLLAPSE(PURGE DNA ELEMENTS)
OUTPUT
UPDATES ADDRESS FIELD IN TOC
FOR EACH DNA DATA ELEMENT

NOTES
USES XMAFR, XMND6, XMND6, XMPK1
CURRENTLY ONLY CLASS 4 & 6 ELEMENTS
INTERFACE TABLES & SEQUENCE TABLES
ARE DNA ELEMENTS.

INCREMENT XMPK1 UPDATE PHASE 1 COUNT
IF DNA DIRECTORY ADDRESS NOT EQ 0, THEN
SAVE X & Y REGS
DO FOR ALL CLASS 4 ELEMENTS
CALL XMAFR(LAST, SIZE)
ENDDB
DO FOR ALL CLASS 6 ELEMENTS
CALL XMAFR(DATA, SIZE)
ENDDB
RESTORE X & Y REGS

ENDIF
END XMPK1
1033 1 BEGIN XMVP3  
1034 2 *00  
1035 2 JSB XMVP3  
1036 2 *00  
1037 2 *01  
1038 2 *01  
1039 2 *01  
1040 2 *01  
1041 2 *03  
1042 2 *03  
1043 2 *03  
1044 2 *05  
1045 2 *05  
1046 2 *05  
1047 2 *05  
1048 2 *05  
1049 2 *05  
1050 2 *05  
1051 2 *05  
1052 2 *05  
1053 2 *05  
1054 2 *05  
1055 2 *05  
1056 2 *05  
1057 2 *05  
1058 2 *05  
1059 2 *05  
1060 2 *05  
1061 2 *05  
1062 1 END XMVP3  

BEGIN XMVP3 CALLING PROCEDURE
JSB XMVP3

FUNCTION
COMPRESS AUTO DATA AREA BY MIGRATING DATA AREAS TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES

OUTPUT
UPDATES XMBCP, XMFCP, XMFC, XMFR, XMFP3

NOTES
USES XMFR

INCREMENT XMVP3 (NUMBER OF PHASES 3+)
IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
CALL XMPP2, INSURE TOC IS COMPRESSED
DO WHILE XMBCP .GT. XMFC . UNTIL 1 FREE AREA ADJACENT TO THE FENCE
IF FCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
SET HIGH-WATER TO XMFC
ELSE
MULTIPLE AREAS
SET HIGH-WATER TO ECP OF LAST-FREE
ENDIF
SET LOW-WATER TO A(LAST-FREE)
PERFORM XMPC(LLOW-WATER,HIGH-WATER)
EXIT IF TOC-ADDRESS .EQ. 0
PERFORM XMNV(TOC-ADDRESS)
ENDDO
ENDIF
END XMVP3
1064 1 BEGIN XMTSC  TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR 0 (NONE FOUND).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XLOWA, THE
1070 2 * LAST TOC-ENTRY IS AT XMFNC-0.
1071 2 TOC-ENTRY=XLOWA
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 DO UNTIL TOC-ENTRY .GE. XMFNC
1074 3 IF DATA ADDRESS IN TOC-ENTRY:
1075 4 NE 0, AND IS
1076 4 LT LOW-WATER, AND IS
1077 4 GT HIGH-WATER, AND IS
1078 4 GT TEST-AD,
1079 3 THEN
1080 4 TEST-AD=DATA ADDRESS
1081 4 TEST-TOC=TOC-ENTRY
1082 3 ENDF
1083 3 ADD 8 TO TOC-ENTRY
1084 2 ENDOO
1085 2 TOC-ADDRESS=TEST-TOC  RETURN 0 OR A TOC ADDRESS
1086 1 ENDO XMTSC
1088 1 BEGIN XNAMV  ANA 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 XMAPR(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 XNAMV
FORTRAN CALLING PROCEDURE

CALL XPATH (LU, INBUF, INTLG, MBUF, IMUN, NAME, TYPE, SIZE,
IDIN, DSPFY)

XPATH 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.
INTLG - LENGTH OF INBUF
MBUF - MANAGER REQUEST BUFFER (4X WORDS) USED TO COMMmUcATE
WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACCESS 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 DEVICE WHERE THE DATA IS TO
BE OBTAINED/STORRED. 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 DATA,
TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DATA
SUBSCRIPTED OR ZERO IF NOT SUBSCRIPTED,
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DATA
OR ZERO IF OUTPUT DATA
IDIN - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DATA OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIPTED OUTPUT.
DSPFY - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIPTED DATA ELEMENTS ELSE ZERO.
RETURNED IN DSPFY.

EXTERNAL SYMBOLS
(SEE XPGET)

INTERNAL VARIABLES

NOTES
USES XPATH, XPE13(XPGET), XPINI(XPGET), XPFMT(XPGET), XPREQ,
APFSUC(XPGET), XPATH, XPATH1

XPATH IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY POINTS XPATH, XPATH1 AND XPATH2
1 BEGIN XPATH
2 PERFORM XPMIC(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
3 EXIT TO XP:15: XPGET IF PARAMETER IS OUT OF RANGE
4 EXTRACT NAME FROM INTERFACE TABLE
5 SET DSPTT TO ZERO
6 IF LITERAL PARAMETER (NAME IS ZERO)
7 THEN
8 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
9 ELSE
10 IF SUBSCRIPTED (INTERFACE TABLE DISP OR $ FIELDS ARE NON-ZERO)
11 THEN
12 PERFORM XPBC(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
13 STORE TYPE, SIZE, IDIM AND DSPTT
14 ELSE
15 ENDIF
16 IF DROE
17 THEN
18 PERFORM XPFN(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 DSPTT FROM TOC ENTRY
24 ELSE
25 ENDIF
26 SET TYPE, SIZE AND IDIM TO ZERO
27 ENDIF
28 ENDIF
29 END XPATH
FORTRAN CALLING PROCEDURE

CALL XPGET (LU, INTBUF, INTLNG, MRUFF, N, INUMS, IN(1), ...)

XPGET ALLOWS PROCESSORS TO OBTAIN DATA FROM THE FD'S AVA. IT
VERIFIES THE EXISTENCE OF THE DATA ELEMENTS AND ORDE'S
REFERENCED BY INTERFACE TABLE AND RETRIEVES INPUT DATA
BY ONE OF THE FOLLOWING WAYS:

1) REQUESTED DATA FROM LITERAL DATA STORED WITHIN THE
INTERFACE TABLE IS RETURNED OR

2) REQUESTED DATA FROM DATA ELEMENT REFERENCED BY
THE INTERFACE TABLE IS RETURNED OR

3) THE NAME OF THE ORDE 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 BODE NAMES ARE SEARCHED FOR IN THE AVA TOC.
IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING TERMINATED.

-------

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INTBUF - INPUT/OUTPUT BUFFER OF 7 OR 8 WORDS.

INTLNG - LENGTH OF INTBUF

MRUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCATCH
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 INPUT DATA IS TO BE
STORED.

-------

INTERNAL VARIABLES

AMULT - ARRAY OF 10 BYTES CONTAINING MULTIPLIERS FOR USE BY THE
PROCEDURE SUMCPX IN DETERMINING WORD DISPLACEMENTS FOR
VARIOUS TYPES OF SUBSCRIPTED ARRAYS

CLSTL - 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)

I - COUNTER OF THE NUMBER OF PASSES THROUGH THE PARAMETER
155 1 *04 PROCESSING LOOP
156 1 *04 MP  = VALUE OF "P", WHERE P IS THE NUMBER OF PARAMETERS IN THE
157 1 *04 INTERFACE TABLE
158 1 *04 NAMFL = ADDRESSES OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
159 1 *04 ARAY REQST
160 1 *04 NARY = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
161 1 *04 SEQUENCE
162 1 *04 RERST = EIGHT WORD ARRAY USED FOR CONSTRUCTING ANA MANAGEMENT
163 1 *04 REQUESTS FOR XPRED
164 1 *04 USRID = USER FILE IDENTIFIER CHARACTER (#IPARM(3))
165 1 *04 EXTERNAL VARIABLES (SEE XPRED)
166 1 *04 XPCLS
167 1 *04 XPLU
168 1 *04 XPRQ0
169 1 *04 XPRQA
170 1 *04 NOTES
171 1 *05 USES .ENTRY, EXEC, XPRED, XPRIT, XVSTB
172 1 *05
173 1 *05 XPGET MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.
174 1 *05 XMLTR INTBUF MUST BE USED BY XPGET, XPPUT, AND XPATH AND XEOS
175 1 *05 TO BE INITIALIZED ONLY ONCE BY ANY OF THE THREE ROUTINES.
176 1 *05 SINCE REQUESTS FOR INPUT DATA FROM THE ANA MAY BE MADE FOR UP TO
177 1 *05 EIGHT AT A TIME, XPGET RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
178 1 *05 REQUESTED IN MULTIPLES OF EIGHT.
179 1 *05
180 1 *05
181 1 *05
182 1 *05
183 1 *05
FORTRAN CALLING PROCEDURE

CALL XPUT (LU, INBUF, INTLNK, NRFU, N, INUMS, OUT(1), ..., OUT(N))

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

XPUT 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. 
**
** 1) IF FOUND, DATA IS STORED 
**
** 1) 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. 
**
** 1) **
**
** INPUT 
**
** 1) LOGICAL UNIT NUMBER OF USER TERMINAL 
**
** 1) **
**
** INBUF - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 13 WORDS), 
**
** ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE 
**
** INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON 
**
** FIRST USE TO CAUSE INITIALIZATION. 
**
**
** INLNK - LENGTH OF INBUF 
**
** NRFU - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE 
**
** WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH 
**
** AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS. 
**
** N - NUMBER OF PARAMETERS TO BE STORED. IF N=0, ALL 
**
** OUTPUT PARAMETERS ARE ASSUMED TO BE SUPPLIED. 
**
** INUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE 
**
** INTERFACE TABLE REFERENCING DATA ELEMENTS AND ORDER'S 
**
** WHERE THE OUTPUT DATA IS TO BE STORED. THE ORDER OF THE 
**
** OUT(N) - LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE 
**
** STORED FROM. 
**
**
** SHARED EXTERNAL SYMBOL 
**
** XPENF - FLAG INDICATING XPUT/IPO NORMAL PROCESSING (-1) OR 
**
** XPST/IPO SPECIAL PROCESSING BY-PASSING SUBSCRIPT 
**
** RESOLUTION (3-19) 
**
**
** OUTPUT 
**
** NONE 
**
** INTERNAL VARIABLES 
**
** (SEE XPUT) 
**
**
** NOTES 
**
** (SEE XPUT)
243 1 BEGIN XGET
244 2 SET FOR 'GET'
245 3 PERFORM ACCESS TO RETRIEVE DATA
246 4 END XGET
247 1 BEGIN XPUT
248 2 SET FOR 'PUT'
249 3 PERFORM ACCESS TO STORE DATA
250 4 END XPUT
251 1 BEGIN XPNI
252 2 PERFORM XPNI TO INITIALIZE GLOBALS AND INTERFACE TABLE
253 3 DO FOR EACH PARAMETER REQUESTED
254 4 IF SELECTED PARAMETER IS OUT OF RANGE
255 5 CALL XPROC TO PURGE QUEUED REQUESTS
256 6 EXIT TO :XPE13:
257 7 ENDIF
258 8 IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/'PUT' PROCESSING
259 9 THEN
260 10 CALL XPROC TO PURGE QUEUED REQUESTS
261 11 EXIT TO :XERR12:
262 12 ENDIF
263 13 IF OVERWRITING CALLING SEQUENCE
264 14 THEN
265 15 CALL XPROC TO PURGE QUEUED REQUESTS
266 16 EXIT TO :XPE13:
267 17 ENDIF
268 18 IF PROCESSING FOR 'GET'
269 19 THEN
270 20 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
271 21 THEN
272 22 IF INPUT IN LITERAL FORM
273 23 THEN
274 24 BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT
275 25 ELSE IF NORMAL XGET/PUT PROCESSING (XPGPF = -1)
276 26 THEN
277 27 IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
278 28 THEN
279 29 PERFORM XPSBC TO COMPUTE DISPLACEMENT; = F(DIM, SUBS, TYPE)
280 30 ELSE DISPLACEMENT IS ZERO
281 31 ENDIF
282 32 USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
283 33 ENDIF
284 34 BUILD REQUEST WITH DE NAME AND DISPLACEMENT
285 35 ENDIF
286 36 CALL XPROC TO QUEUE RETRIEVAL AND STORAGE OF INPUT
287 37 ELSE PARAMETER IS DDE (CLASS 3)
288 38 THEN
289 39 CALL XPROC TO IMMEDIATELY RETRIEVE TDC ENTRY
290 40 PERFORM XPSFM TO CONSTRUCT AND STORE QUALIFIED FILE NAME
291 41 ENDIF
292 42 ELSE PROCESSING FOR 'PUT'
293 43 IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
294 44 THEN
295 45 IF NORMAL XGET/PUT PROCESSING (XPGPF = -1)
296 46 THEN
BEGIN XPINI
  INITIATE GLOBAL VALUES FROM LU AND XVS TB
  TERMINATE PROCESSOR WITH XP10 ERROR IF LU NOT IN XVS TB
  IF INTERFACE TABLE BUFFER NOT INITIALIZED
    THEN
      RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER
      IF RETRIEVAL NOT SUCCESSFUL
        THEN
          TERMINATE PROCESSOR WITH 'XP10 PROCESSOR INITIALIZATION ERROR'
        ENDIF
      ENDIF
    ENDIF
  EXIT TO :XPE13: IF N < 0
END XPINI

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 XPSRC
  IF DOUBLE SUBSCRIPTED
    THEN
      BUILD REQUEST FOR SUBSCRIPTS STORED IN LITERAL BLOCK
      CALL XPRER TO QUEUE RETURN OF SUBSCRIPTS
      ELSE
      ENDIF
      SINGLE SUBSCRIPT IS CONTAINED IN DISPLACEMENT FIELD & J-SUBSCRIPT IS = 1
      CALL XPRER TO RETURN 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
        ENDIF
      ENDIF
    ENDIF
  END ACCESS

EXECUTE
FORTAN CALLING PROCEDURE

CALL XPGTI (LU, INBUF, INTLNG, MBUFF, INUM, IN, SIZE, DISP)

**********

XPGTI 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 7*(# PARAMETERS + 1) WORDS,
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION

INTLNG - LENGTH OF INBUF

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

INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM

WHICH TO RETURN DATA

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

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

JGETPT - TWO WORD VECTOR CONTAINING 'JSB XPGT' OR 'JSB XPFU'

INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGT AND
XPFU CALLS

**********

USES EXEC, ENTR, XPATR, XPE13(XPATR), XPF(XPATR), XPP11(XPATR)
USES EXEC, ENTR, XPATR, XPE(XPATR), XPP11(XPATR), XPF(XPATR),
XPPUT(XPATR)

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

CALL XPTI (LU, INBUF, INLNG, MRBUFF, INUM, OUT, SIZE, DISP)

GENERAL DESCRIPTION

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

INPUT:

- LU - logical unit number of user terminal
- INBUF - input/output buffer of 7*(# parameters + 1) words
- INLNG - interface table header. First word must be zero only on 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 which to store data
- SIZE - total number of words to be stored
- DISP - displacement from the beginning of the output parameter at which to begin data storage. A value of zero indicates the beginning of the area specified by the user in the interface table. Negative values of DISP must not be specified, i.e., storage into an area previous to the user's subscripts is not supported.

OUTPUT:

- NONE

INTERNAL:

- SEE XPGTI

NOTES:

- XPTI is an entry point into XPGTI
1 BEGIN XPGTI
2 SET FOR GET PROCESSING
3 PERFORM XPGXI TO RETRIEVE DATA
4 END XPGTI
5 BEGIN XPGTI
6 SET FOR PUT PROCESSING
7 PERFORM XPGXI TO STORE DATA
8 END XPGTI
9 BEGIN XPGTI
10 SET XPGPF(XPGF*) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
11 CALL XPATR TO ASSURE Initialization OF INBUF AND RETURN BASE DISPLACEMENT
12 EXIT TO :XPE13: (XPATR) IF INDICATED PARAMETER IS A DROG FILE
13 SAVE INTERFACE TABLE ENTRY CLASS/TYPW WORD, FLAG/DISP WORD AND SIZE WORD
14 IF LITERAL (NAME = Q)
15 THEN
16 EXIT TO :ERR15: IF DISP + SIZE > SAVED SIZE
17 ELSE
18 INCREMENT BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
19 SET TYPE FIELD TO FREE
20 EKDF
21 CLEAR ENTRY SUBSCRIPT BIT
22 SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
23 SET SIZE FIELD TO SIZE
24 CALL XPGGET/XPPUT TO TRANSFER DATA
25 RESTORE ORIGINAL INTERFACE TABLE ENTRY
26 RESTORE XPGPF(XPGF) TO NOMINAL VALUE
27 EXIT XPGXI
28 :ERR15: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
29 END XPGXI
CALLING PROCEDURE

JSB XPREQ

DEF 'W' = WHERE 'W' IS THE NUMBER OF ACTUAL ARGUMENTS

DEF OPRN = REQUIRED OPTION

DEF REQU = REQUIRED REQUEST

DEF ADRS = OPTIONAL ADDRESS (SEE OPRN)

*********

XPLO/XPPUT BUFFERED AM/AM MANAGEMENT SERVICE

*********

INPUT:

OPTN = OPTION WORD

SIGN BIT = 0 QUEUE REQUEST AND RETURN

1 QUEUE REQUEST AND CLOSE BUFFER

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

ADRES

1 QUEUE REQUEST ONLY

REQS = EIGHT WORD REQUEST (SEE SDD 6.2.6.2)

ADRES = STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT

VALUES CUSED ONLY WHEN OPRN(RIGHT BIT) = 0

*********

EXTERNAL SYMBOLS

XPLU = TERMINAL LOGICAL UNIT NUMBER

XPRB = ADDRESS OF 64 WORD MANAGER REQUEST BUFFER

XPCLS = CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

*********

OUTPUT

IF A REQUEST FAILS, A MESSAGE IS ISSUED AND PROCESSING IS

TERMINATED, OTHERWISE INCOMING DATA WILL BE STORED IN THE

ADDRESSES SUPPLIED.

*********

LOCAL VARIABLES

ADDS = ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDs)

ADDS = TABLE OF ADDRESSES FOR STORAGE OF FETCHED DATA

APRMS = ADDRESS OF PARMS AREA

CLASS = CLASS NUMBER USED TO TRANSMIT DATA TO MANAGER

PARAMETER RETURN AND SCRATCH AREA

PTR = SAVE AREA FOR ADDS AND XPRB POINTER

*********

NOTES

USES ENTER, EXEC., XPLXIT, XPAW

IF DATA IS TO BE TRANSFERRED TO/FROM 'ADRES', THEN 'REQS' SIZE

FIELD (WORD 0) MUST BE SET.
1 BEGIN XPREQ
2 MOVE REQUEST INTO BUFFER
3 IF OPTION IS TO TRANSFER DATA
4 THEN
5 IF REQUEST IS TO RETRIEVE DATA (8)
6 THEN
7 STORE ADDRESS IN TABLE
8 ELSE SHOULD BE A REQUEST TO STORE DATA (7)
9 OUTPUT DATA TO CLASS I/O
10 STORE CLASS NUMBER IN REQUEST WORD
11 ENDIF
12 ENDF
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 PAW MANAGER WITH REQUEST FOR AMA MANAGEMENT
19 RETRIEVE RETURN PARAMETERS
20 IF REWRITE FLAG SET (PARMS)
21 THEN
22 RETRIEVE REQUEST BUFFER
23 ENDIF
24 IF REQUESTS WERE SUCCESSFUL
25 THEN
26 DO FOR EACH REQUEST IN BUFFER
27 IF REQUEST TO RETURN DATA (8, 9 OR 16)
28 THEN
29 RETRIEVE AND STORE DATA IN ADDRESS CONTAINED IN TABLE
30 ENDIF
31 ENDDO
32 CLEAR POINTER AND LOCAL CLASS NUMBER
33 ELSE
34 OUTPUT FAILURE MESSAGE (XP11)
35 DO FOR EACH REQUEST IN BUFFER
36 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
37 THEN
38 FREE CLASS I/O NUMBER AND SAM BUFFER
39 ELSE
40 ENDDO
41 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
42 ENDIF
43 ENDF
44 ENDDO
45 1 END XPREQ
FORTAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE

CALL XP30S (LU,PRMLEN,PROMT,TYPE,DATEN,1DIM,DATA,RETS)

CALLS XPROM, 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
PROMT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING USED AS THE USER PROMPT
TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
0 - FREE
1 - INTEGER
2 - REAL
3 - DOUBLE PRECISION
4 - CHARACTER STRING LENGTH
5 - CHARACTER STRING LENGTH
6 - CHARACTER STRING LENGTH
7 - CHARACTER STRING LENGTH
8 - CHARACTER STRING LENGTH
DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE NUMBER OF WORDS.
1DIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0 IF IT IS A VECTOR.

OUTPUTS FROM CALLING SEQUENCE:

DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE) AREA TO CONTAIN THE USERS RESPONSE
RETC - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO CALLER:
0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
1 - USER ENTERED LESS THAN COLUMNS 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. 1DIM LESS THAN ZERO
B. DATLEN LESS THAN ONE
C. TYPE: VALID TYPE SPECIFIED.

INTERNAL VARIABLES

5-207
DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT
ELEMENT IS TO BE STORED
EFFTPY - 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
STKPRM - 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
WHERE EACH REPEAT GROUP BEGINS
TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED
TYPEI - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE
T:TYPE = TYPE +1

SUBROUTINES AND FUNCTIONS CALLED:
EXEC, XPRDS, ERRMSG

POL ROUTINES INCLUDED:
XPRDS, STRING, SUBSCR, REPET, ERRMSG
1 BEGIN XPRODS
2 IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3 CALL XPRM TO PROMPT USER AND RECEIVE RESPONSE
4 DO UNTIL EOS TOKEN IS FOUND
5 USE MESSAGE NUMBER TO XPO2
6 CASE TOKEN (NUMBER:, NUMBER:, NUMBER:, STRING:, NULL,:
7 ;SUBSCR:, ;REPET:)
8 :NUMBER:
9 EREREIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERMSG
10 SET MESSAGE NUMBER TO XPO2
11 EREREIT IF THERE IS NO ROOM IN DATA AREA PERFORM ERMSG
12 SET PREVIOUS TOKEN TO DATA
13 CALL ERMOV TO MOVE DATA INTO DATA AREA
14 INCREMENT POINTER IN DATA AREA
15 INCREMENT POINTER TO NEXT TOKEN
16 :STRING:
17 PERFORM STRING
18 :NULL:
19 SET MESSAGE NUMBER TO XPO2
20 EREREIT IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERMSG
21 SET PREVIOUS TOKEN TO DATA
22 INCREMENT POINTER IN DATA AREA
23 INCREMENT POINTER TO NEXT TOKEN
24 :SUBSCR:
25 PERFORM SUBSCR
26 :REPET:
27 PERFORM REPET
28 ENDCASE
29 DO UNTIL TOKEN IS NOT A CLOSED PARENTHESIS
30 IF STACK IS NOT EMPTY AND
31 (PREVIOUS TOKEN IS DATA AND TOP OF STACK PAREN FLAG = 0) OR
32 (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG = 1) THEN
33 DECREMENT TOP OF STACK REPEAT COUNT BY 1
34 IF TOP OF STACK REPEAT COUNT > 0 THEN
35 RESET TOKEN POINTER TO TOP OF STACK INDEX
36 ELSE
37 POP TOP ENTRY ON STACK
38 IF TOKEN IS A CLOSED PAREN THEN
39 INCREMENT TO NEXT TOKEN
40 ENDIF
41 ENDIF
42 ENDOD
43 ENDIF
44 ENDOD
45 ENDIF
46 SET XPRM RETURN CODE = XPRM RETURN CODE
47 SET XPRM RETURN CODE TO SAY INVALID PARAMETER LIST
48 ENDIF
49 ENDOD
50 ENDOD
51 END XPRODS
BEGIN STRING
ERREXIT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERMSG
DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
ERREXIT IF TYPE OF RESPONSE > TYPE REQUESTED AND
ERREXIT IF TYPE IS NOT FREE THEN
SET EFFECTIVE LENGTH = LENGTH REQUESTED
END IF
SET MESSAGE NUMBER TO XPO7
ERREXIT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERMSG
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
1 END STRING
1
1
1
1
BEGIN SUBSCR
INCREMENT POINTER TO NEXT TOKEN
SET MESSAGE NUMBER TO XPO7
ERREXIT IF TOKEN IS NOT AN INTEGER TO PERFORM ERMSG
IF 1-DIMENSION > 1 THEN
SFT 1 TO INTEGER VALUE
INCREMENT POINTER TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT AN INTEGER OR
ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERMSG
SET MESSAGE NUMBER TO XPO7
ERREXIT IF INTEGER > 1-DIMENSION TO PERFORM ERMSG
ERREXIT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERMSG
ELSE
ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERMSG
SET MESSAGE NUMBER TO XPO7
ERREXIT 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 XPO7
ERREXIT IF TOKEN IS AN EOS OR
ERREXIT IF TOKEN IS A REPEAT OR
ERREXIT IF TOKEN IS A CLOSE PAREN OR
ERREXIT IF TOKEN IS A SUBSCRIPT TO PERFORM ERMSG
1 END SUBSCR
1
1 BEGIN REPET
2 SET MESSAGE NUMBER TO XPDS
3 ERREXIT IF TOKEN IS AN EOS TO PERFORM ERRMSG
4 INCREMENT STACK POINTER
5 SET MESSAGE NUMBER TO XPDS
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 1 OR
17 1 BEGIN ERRMSG
18 CALL EXEC TO WRITE ERROR MESSAGE
19 PERFORM XPDS - NO RETURN
20 END ERRMSG
FORTRAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE

CALL XPRDM (LU,PRLLEM,PROMPT,COMLEN,COMBUF,RET)

WRTIES "PRLLEM" WORDS OF "PROMPT" TO USER READS THE USER'S
RESPONSE; CONVERTS RESPONSE TO TOKENS IN "COMBU" AND PASSES
BACK A RETURN CODE "RET" INDICATING THE SUCCESS OF XPRDM
AND THE USEFULNESS OF "COMBU".

INPUTS FROM CALLING SEQUENCE:

LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
PRLLEM - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
PROMPT - (INTEGER, PRLLEM WORDS) IS THE CHARACTER STRING
USED AS THE USER PROMPT.
COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE
COMMUNICATIONS BUFFER (COMBU)

OUTPUTS FROM CALLING SEQUENCE:

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

INTERNAL VARIABLES

COUNT - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN
A CHARACTER STRING AND NUMER DIGITS IN A NUMBER.
DBLHT - DOUBL WORD USED TO ACCUMULATE AN INTEGER VALUE
DBLVD - DOUBL WORD USED TO ACCUMULATE A REAL OR DOUBLE
VALUE
FLGCOM - COMA FLAG
O - LAST CHARACTER NOT A COMA
1 - LAST CHARACTER WAS A COMA
FLGCOM - CONTINUE FLAG
O - THIS IS NOT A RESPONSE TO A CONTINUE
1 - THIS IS A CONTINUED RESPONSE
FLGNUM - NUMBER FLAG
O - POSITIVE NUMBER
T - NEGATIVE NUMBER
FLGPOW - POWER FLAG
O - POSITIVE POWER
1 - NEGATIVE POWER
2 - SINGLE PRECISION
3 - DOUBLE PRECISION

SUBROUTINES AND FUNCTIONS USED:
EXEC, KCVT, XRMOV, XRRC, XRUPK, OV

PDL ROUTINES USED:
XPRM, TOKENS, QUOTE, DIGIT, DCOL, DECP, EORD,
INTEGR, REAL, DBL, INVAL, COMPUL
E98 1 BEGIN XPRDM
E99 2 IF PROMPT IS NOT TOO LONG THEN
E100 3 CALL XMOV "\) MOVE PROMPT INTO OUTPUT AREA
E101 4 CALL XDEC TO WRITE PROMPT
E102 5 SET XPRDM RETURN CODE TO NORMAL RETURN
E103 6 SET CONTINUE FLAG OFF
E104 7 INITIALIZE COMMUNICATIONS BUFFER
E105 8 :COMLOP:
E106 9 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
E107 10 CALL EXEC TO READ RESPONSE
E108 11 CALL XRUPK TO CONVERT A2 RESPONSE TO R1 FORMAT
E109 12 IF NUMBER OF WORDS READ IS NOT ZERO THEN
E110 13 SET COMMA FLAG ON
E111 14 DO WHILE CONTINUE FLAG IS OFF
E112 15 IF INPUT BUFFER IS COMpletely SCANNED THEN
E113 16 SET CONTINUE FLAG ON
E114 17 IF COMMA FLAG IS ON THEN
E115 18 CALL EXEC TO WRITE CONTINUE
E116 19 GO TO :COMLOP:
E117 20 ENDIF
E118 21 ELSE
E119 22 EXIT IF COMBUF IS FULL PERFORM COMFUL
E120 23 IF INPUT CHARACTER IS A COMMA THEN
E121 24 IF COMMA FLAG IS ON THEN
E122 25 EXIT IF COMBUF CANNOT HOLD TOKEN PERFORM COMFUL
E123 26 STORE NULL FIELD TOKEN IN COMBUF
E124 27 INCREMENT WORDS IN COMBUF BY 1
E125 28 INCREMENT TOKENS IN COMBUF BY 1
E126 29 ENDIF
E127 30 SET COMMA FLAG ON
E128 31 SET NEXT INPUT CHARACTER
E129 32 ELSE
E130 33 SET COMMA FLAG OFF
E131 34 PERFORM TOKENS
E132 35 ENDIF
E133 36 ENDIF
E134 37 ELSE
E135 38 IF CONTINUE FLAG IS OFF THEN
E136 39 SET XPRDM RETURN CODE TO SAY USER ENTERED CR
E137 40 ENDIF
E138 41 ENDIF
E139 42 ELSE
E140 43 SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
E141 44 ENDIF
E142 45 END XPRDM
BEGIN TOKENS
SET NEGATIVE NUMBER FLAG OFF
SET NEGATIVE POWER FLAG OFF
SET POWER = 0
IF INPUT CHARACTER IS A DIGIT THEN
  PERFORM DIGIT
ELSE
  :A:
    SET XPROM RETURN CODE TO SAY I ENTERED
  :B:
    PERFORM QUOTE
  :C:
    SET PAREN INDICATOR = 0
    GO TO :E:
  :D:
    SET PAREN INDICATOR = 1
    GO TO :E:
  :E:
    ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN TO PERFORM COMFUL
    SPACE TOKEN (OPAR + PAREN INDICATOR) IN COMBUF
    INCREMENT #WORDS IN COMBUF BY 1
    INCREMENT #TOKENS IN COMBUF BY 1
    GET NEXT CHARACTER
  :F:
    ERREXIT IF NEXT CHARACTER IS NOT A DIGIT PERFORM INVAL
    SET INTEGER = 0
    PERFORM DECP
  :G:
    IF INPUT CHARACTER IS A - THEN
      SET NEGATIVE NUMBER FLAG ON
      ENDIF
    GET NEXT CHARACTER
    IF INPUT CHARACTER IS A DIGIT THEN
      PERFORM DIGIT
    ELSE
      IF INPUT CHARACTER IS A . THEN
        GO TO :F:
      ELSE
        PERFORM INVAL - NO RETURN
      ENDIF
    ENDIF
  END CASE
ENDIF
END CASE
END TOKENS
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 ERREXIT 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 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1022 3 IF INPUT CHARACTER IS A "0" THEN
1023 4 PERFORM DECPT
1024 5 ELSE IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1025 6 PERFORM EORD
1026 7 ELSE IF INPUT CHARACTER IS AN "R" THEN
1028 8 EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1030 9 EXIT IF INTEGER IS ZERO PERFORM INVALID
1031 6 STORE REPEAT TOKEN IN COMBUF
1032 6 INCREMENT #WORDS IN COMBUF BY 2
1033 6 INCREMENT TOKENS BY 1
1034 6 GET NEXT CHARACTER
1035 5 ELSE
1036 6 PERFORM INTEGR
1037 5 ENDIF
1038 4 ENDIF
1039 3 ENDIF
1040 2 ELSE
1041 3 PERFORM INTEGR
1042 2 ENDIF
1043 1 END DIGIT
1044 1 *
1045 1 *
1046 1 *
1047 1 BEGIN DCOL
1048 2 SET INTEGER = 0
1049 2 SET COUNTER = 0
1050 2 DO WHILE CHARACTER IS A DIGIT AND
1051 3 WHILE INPUT BUFFER IS NOT EXHAUSTED
1052 5 SET INTEGER = (INTEGER * 10) + INPUT CHARACTER - 48
1053 5 EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1054 5 INCREMENT COUNTER BY 1
1055 5 GET NEXT CHARACTER
1056 2 ENDDO
1057 1 END DCOL

ORIGINAL FILE FROM
1059 1 BEGIN DECP
1060 2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061 3 GET NEXT CHARACTER
1062 4 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063 5 IF INPUT CHARACTER IS A DIGIT THEN
1064 6 PERFORM DCON
1065 7 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066 8 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067 9 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 DECP
BEGIN INTEGER

EXIT IF NEXT TOKEN IS NOT A COMMA AND
EXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
EXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
CONVERT DOUBLE TO INTEGER
EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
EXIT IF NEGATIVE NUMBER FLAG IS ON THEN
SET INTEGER = -INTEGER
ENDIF
STORE INTEGER TOKEN IN COMBUF
INCREMENT #WORDS IN COMBUF BY 2
INCREMENT #TOKENS IN COMBUF BY 1
END INTEGER

BEGIN REAL

EXIT IF NEXT TOKEN IS NOT A COMMA AND
EXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
EXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
SET REAL = DOUBLE * 10 ** POWER
EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
EXIT IF NEGATIVE NUMBER FLAG IS ON THEN
SET REAL = -REAL
ENDIF
STORE TOKEN IN COMBUF
INCREMENT #WORDS IN COMBUF BY 3
INCREMENT #TOKENS IN COMBUF BY 1
END REAL
1153 1 BEGIN INVAL
1154 2 CALL KCVT TO CONVERT OCTAL CHARACTER NUMBER TO ASCII
1155 2 CALL EXEC TO WRITE ERROR MESSAGE
1156 2 PERFORM XPARM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1157 1 END INVAL
1158 1 *
1159 1 *
1160 1 *
1161 1 BEGIN CONFUL
1162 2 CALL EXEC TO WRITE ERROR MESSAGE
1163 2 PERFORM XPARM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1164 1 END CONFUL
1165 1 *
1166 1 *
1167 1 *
1168 1 BEGIN OVFLOW
1169 2 CALL KCVT TO CONVERT OCTAL TO ASCII
1170 2 CALL EXEC TO WRITE ERROR MESSAGE
1171 2 PERFORM XPARM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1172 1 END OVFLOW
FORTRAN CALLING PROCEDURE
CALL XPXIT (LU, RPARAMS)

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

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

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

OUTPUT
RPARMS IS RETURNED TO THE FATHER TASK

NOTES
USES EXEC. PRMT

XPXIT DOES NOT RETURN TO THE CALLER.

BEGIN XPXIT
IF LU IS NON-ZERO
THEN
WAIT ON ANY CLASS I/O TO COMPLETE (CLASS I/O CONTROL THEN GET)
ENDIF
RETURN PARAMETERS TO FATHER
TERMINATE PROGRAM
END XPXIT
FORTRAN CALLING PROCEDURE

CALL XBIT (BIT, BITNUM, STRING)

*********

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

*********

INPUT

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

*********

OUTPUT

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

*********

NOTES

USES .ENTR, XRSER

*********

BEGIN XBIT

TRANSFER CALLING SEQUENCE

COMPUTE ADDRESS OF WORD CONTAINING BITNUM

COMPUTE BIT NUMBER WITHIN WORD

CALL XRSER TO SET/CLEAR BIT

END XBIT
INTEGER FUNCTION
1 000  XRCPR(LENGTH, ARRAY1, ARRAY2)
1 000  
1 001  ************
1 001  COMPAR LENT'' WORDS OF 'ARRAY1' TO 'ARRAY2'. IF THE ARRAYS
1 001  ARE IDENTICAL A VALUE OF 0 IS RETURNED. IF THE COMPARE FAILS
1 001  A VALUE IS RETURNED INDICATING WHETHER THE FIRST UNEQUAL
1 001  ELEMENT OF ARRAY1 IS LESS THAN ARRAY2 (-1) OR GREATER (+1)
1 001  
1 001  ************
1 002  INPUT
1 002  LENGTH - POSITIVE INTEGER NUMBER OF WORDS TO BE COMPARED
1 002  ARRAY1 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY2
1 002  ARRAY2 - ARRAY OF INTEGERS OR CHARACTERS TO BE COMPARED TO ARRAY1
1 002  
1 002  ************
1 003  OUTPUT
1 003  FUNCTION VALUE - 0, ARRAYS EQUAL
1 003  -1, ARRAYS NOT EQUAL, ARRAY1 < ARRAY2
1 003  +1, ARRAYS NOT EQUAL, ARRAY1 > ARRAY2
1 003  
1 003  ************
1 005  NOTES
1 005  USES .ENTR
1 005  
1 005  ************
1 006  
1 006  BEGIN XRCPR
1 006  TRANSFER CALLING SEQUENCE
1 006  SETUP COMPARE
1 006  COMPARE ARRAY1 AND ARRAY2
1 006  RETURN RESULT FLAG
1 006  END XRCPR
80 1 CDO  FORTRAN CALLING PROCEDURE.
81 1 CDO  CALL XR018 (DOUBLE, ASCII)
82 1 CDO
83 1 CDO
84 1 CDO
85 1 CDO  CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN
86 1 CDO  FP018.1Y FORMAT
87 1 CDO
88 1 CDO
89 1 CDO
90 1 CDO  INPUT
91 1 CDO  DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED
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, IADD, IDINT, IOR, KCVT, XNEXT
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 EMD00
124 1 EMD XR018
FORTRAN CALLING PROEDURE

CALL XREQ

MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE

INPUT

COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPTR

ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER

OUTPUT

COMMON XE - REQPTR

CLASS I/O WRITE/READ TO CLASS 'CLASNO'

REQBUF AND RESPONSE IS PRINTED IF REQUESTED

COMMUNICATES WITH FDS MANAGER FATHER TASK

USES EXEC, IAND, XRMOV, XRMSG, XR6, XRFL, XUPL, XPAW

BEGIN XREQ

PERFORM TRACE

OUTPUT REQUESTS TO MANAGER

REQUEST AWA MANAGEMENT AND WAIT FOR RESPONSE

RETRIEVE MANAGER RESPONSE

PERFORM TRACE

RETURN RESPONSE IN REQPTR

END XREQ

BEGIN TRACE

IF TRACE REQUESTED

THEN

DO FOR EACH REQUEST

CALL XUPL TO FORMAT LINE

OUTPUT LINE

ENDDO

OUTPUT PARM1 AND PARM2

ENDIF

END TRACE
INTEGER FUNCTION
XEXT(START, LENGTH, SOURCE)

**********

**********

**********

**********

**********

BEGIN XEXT
Transfer calling sequence
If START NOT = 0
Then
Constructure shift
Load a with source
Shift b a left start bits
Else
Load a with source
Endif
Save a
Clear b
Constructure shift
Restore a
Shift b a left length bits
Move result from b to a
End XEXT
FORTRAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

CONVERT A SINGLE PRECISION REAL NUMBER TO AN ASCII STRING IN 1PE14.4 FORMAT

REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED

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

WORKING LOCATION FOR ABSOLUTE VALUE OR '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
263 1 CD0   FORTRAN CALLING PROCEDURE
264 1 CD0
265 1 CD0   CALL XR16 (INTEGER, ASCII)
266 1 CD0
267 1 C*****
268 1 CD1   CONVERT A SIXTEEN BIT SIGNED BINARY INTEGER TO A SIX CHARACTER
269 1 CD1   ASCII STRING
270 1 CD1
271 1 C*****
272 1 CD2   INPUT
273 1 CD2   INTEGER - SIXTEEN BIT INTEGER TO BE CONVERTED
274 1 CD2
275 1 C*****
276 1 CD3   OUTPUT
277 1 CD3   ASCII - THREE WORD CHARACTER STRING REPRESENTATION OF 'INTEGER'
278 1 CD3
279 1 C*****
280 1 CD4   LOCAL
281 1 CD4   I - INTERNAL LOCATION FOR 'INTEGER' REPEATEDLY MODIFIED TO
282 1 CD4   PRODUCE 'ASCII'
283 1 CD4
284 1 CD4   WRK - SEVEN WORD WORKING BUFFER FOR CONSTRUCTION OF 'ASCII'
285 1 CD4
286 1 C*****
287 1 CD5   NOTES
288 1 CD5   USES XRMV AND XRCK
289 1 CD5
290 1 C*****
291 1 *
292 1 *
293 1 *
294 1 *
295 1 BEGIN XR16
296 2   BEGIN WORKING SPACE
297 2   CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING
298 2   SET SIGN OF 'INTEGER' IN 'ASCII'
299 2   CALL XRCK TO CONVERT FROM R1 TO A2 FORMAT
300 1 END XR16
*FORTRAN CALLING PROCEDURES*

1 00 CALL XRLCK (RCODE)
2 01 CALL XRULK (RCODE)

1 ************
2 01 XRLCK AND XRULK PROVIDE A MECHANISM FOR SERIALIZING THE UPDATE
3 01 OF FDS GLOBAL SYSTEM TABLES AND FILES. THE RESOURCE NUMBER
4 01 STORED IN THE XVSTB RESIDENT STATUS TABLE IS USED AS THE
5 01 LOCKING MECHANISM
6 01 ************
7 02 INPUT
8 02 'VSTB RESOURCE NUMBER
9 02 ************
10 03 OUTPUT
11 03 RCODE - INTEGER RETURN CODE (0 - SUCCESSFUL, 1 - FAILURE)
12 03 ************
13 04 LOCAL
14 04 STAT - STATUS OF THIS COPY OF XEXEC USES XVSTB RN
15 04 1 - RN LOCKED
16 04 4 - RN UNLOCKED
17 04 ************
18 05 NOTES
19 05 USES .ENA, RNR.
20 05 ************
21 05 THIS ROUTINE MAY NOT BE OVERLAYERED
22 05 ************
337 1 BEGIN XRLC
338 2 SET REQUEST FOR LOCK FUNCTION
339 2 PERFORM RLOCK
340 1 END XRLC
341 1 BEGIN XRLK
342 2 SET REQUEST FOR UNLOCK FUNCTION
343 2 PERFORM RLOCK
344 1 END XRLK
345 1 BEGIN RLOCK
346 2 IF REQUEST IS CONSISTENT WITH STATUS
347 2 THEN
348 3 SET NEW STATUS
349 3 IF RM IN STB IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
350 3 THEN
351 4 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
352 4 ENDF
353 3 CLEAR RETURN CODE
354 2 ELSE
355 3 SET RETURN CODE
356 2 ENDF
357 1 END RLOCK
 INTEGER FUNCTION

 XRLOC(A)

 **********

 RETURN THE 16-BIT MAPPED ADDRESS OF A

 **********

 INPUT

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

 **********

 OUTPUT

 XRLOC - 16-BIT ADDRESS OF A

 **********

 NOTES

 NO EXTERNAL REFERENCES

 **********

 BEGIN XRLOC

 TRANSFER CALLING SEQUENCE

 LOAD THE ADDRESS OF THE CALLING PARAMETER

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

CALL XMSG (NUMBER, LOCATE, LENGTH, SOURCE)

BEGIN 'NUMBER' BEGINNING AFTER 'LOCATE' WORDS, CONCATENATE TO PREFIX,
TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL.

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 - XX
  8 - XL
  9 - BF
  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)

LU - USERS LOGICAL UNIT NUMBER

COMMON

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 FDS SYSTEM MESSAGE FILE XMSG
USES CLOSE, EXEC. IAMS, KCVT, OPEN, READ, XMOV, XUDG

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

XNXTF(BIT, BITNUM, STRING)

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

**INPUT**

*BIT* - INTEGER VALUE THE LAST BIT OF WHICH IS TO BE COMPARED TO BITS OF 'STRING' FOR A MATCH.

*BITNUM* - UNSIGNED SIXTEEN BIT INTEGER INDICATING THE BIT NUMBER IN 'STRING' WITH WHICH TO BEGIN THE SEARCH (FIRST BIT OF 'STRING' IS BIT NUMBER ZERO).

*STRING* - BIT STRING TO BE SEARCHED. SEARCH WILL CONTINUE THROUGH MEMORY UNTIL A VALUE OF 'BIT' IS DETECTED.

**OUTPUT**

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

**NOTES**

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

A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4096 WORDS) CAN BE MEANINGFULLY ACCOMMODATED.
1 BEGIN XNKB
2   COMPUTE STARTING WORD NUMBER
3   COMPUTE STARTING BIT NUMBER
4   LOAD STARTING WORD
5   SHIFT WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
6   INITIALIZE BIT COUNT
7   DO WHILE SIGN BIT IS NOT EQUAL TO BIT
8      IF WORD NOT FINISHED
9         THEN
10            SHIFT WORD LEFT
11            INCREMENT BIT COUNT
12      ELSE
13         DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
14   LOAD NEXT WORD
15   ENDDO
16   CLEAR BIT COUNT
17   ENDF
18   ENDDO
19   RETURN VALUE OF MATCHING BIT NUMBER
20 END XNKB
If I get it, I convert a word from binary to six character octal representation.

Input: Binary - Binary word to be converted

Output: Octal - Three word array containing octal representation of "Binary" in ASCII format (06)

Notes:

1. USES_21TR
2.idos

BEGIN XRO6

Transfer calling sequence:

Form 18-bit word using 8 and 2 most significant bits of A
Set byte flag high
Do for each pair of octal digits (3)
Set prefix bits (10000000)
Shift in digit
If byte flag set high
Then
Shift for accumulation of low byte
Else
Store pair of digits in octal(c)
Clear for next pair of digits
Endif
Flip byte flag
Enddo
Enddo

1. MD XRO6
FORTRAN CALLING PROCEDURE

*00 CALL XRPK (LENGTH, UNPKED, PACKED)
*00
*00 ************
*00 1* INPUT
*00 1*1 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED
*00 1*2 UNPKED - ARRAYS OF CHARACTERS IN R1 FORMAT
*00 1*2 ************
*00 1** OUTPUT
*00 1**1 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS
*00 1**2 ODD, THE LAST WORD WILL BE BLANK FILLED.
*00 1** ************
*00 1**5 NOTES
*00 1**5 USES .ENTR
*00 1**5 ************
*00 1* BEGIN XRPK
*00 2 TRANSFER CALLING SEQUENCE
*00 2* SET BYTE FLAG FOR HIGH BYTE
*00 2* INITIALIZE PACKED POINTER
*00 2* DO FOR EACH CHARACTER IN UNPKED
*00 2*3 IF BYTE FLAG SET HIGH
*00 2*4 THEN
*00 2*4 1 LOAD A WITH NEXT WORD OF UNPKED
*00 2*4 2 SHIFT CHARACTER INTO HIGH BYTE
*00 2*4 3 ELSE
*00 2*4 4 INCLUSIVE OR NEXT WORD OF UNPKED INTO A
*00 2*4 5 STORE A IN PACKED
*00 2*4 6 INCREMENT POINTER
*00 2*4 7 ENDF
*00 2*4 8 FLIP BYTE FLAG
*00 2*5 ENDDO
*00 2*5 ENDDO
*00 2*5 IF BYTE FLAG SET LOW
*00 2*5 THEN
*00 2*5 1 INCLUSIVE OR BLANK INTO LOW BYTE
*00 2*5 2 ENDF
*00 2*5 3 STORE A IN PACKED
*00 1* END XRPK
FORTRAN CALLING PROCEDURE

CALL XRQFM (PREFIX, NAME4, NAME6)

XRQFM 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 INPUT NAME.

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 XRQFM

STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING
MOVE NAME3 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 XRQFM
FORTRAN CALLING PROCEDURE

CALL XRSET (START, LENGTH, SOURCE, OBJECT)

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

***********

INPUT

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

LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED

SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED

***********

OUTPUT

OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED

***********

NOTES

USES .ENTR

***********

BEGIN XRSET

TRANSFER CALLING SEQUENCE

CONSTRUCT SHIFT INSTRUCTIONS

SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER

SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD

CONSTRUCT MASK AND CLEAR FIELD OF OBJECT

INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT

END XRSET
INTEGER FUNCTIONS

1  *DO
2  XSFL(COUNT, SOURCE)
3  XSSF(COUNT, SOURCE)

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  **********
66  **********
67  **********
68  **********
69  **********
70  **********
71  **********
72  **********
73  **********
74  **********
75  **********
76  **********
77  **********
78  **********
79  **********
80  **********
81  **********
82  **********
83  **********
84  **********
85  **********
86  **********
87  **********
88  **********
89  **********
90  **********
91  **********
92  **********
93  **********
94  **********
95  **********
96  **********
97  **********
98  **********
99  **********
100  **********
101  **********
102  **********
103  **********
104  **********
105  **********
106  **********
107  **********
108  **********
109  **********
110  **********
111  **********
112  **********
113  **********
114  **********
115  **********
116  **********
117  **********
118  **********
119  **********
120  **********
121  **********
122  **********
123  **********
124  **********
125  **********
126  **********
127  **********
128  **********
129  **********
130  **********
131  **********
132  **********
133  **********
134  **********
135  **********
136  **********
137  **********
138  **********
139  **********
140  **********
141  **********
142  **********
143  **********
144  **********
145  **********
146  **********
147  **********
148  **********
149  **********
150  **********
151  **********
152  **********
153  **********
154  **********
155  **********
156  **********
157  **********
158  **********
159  **********
160  **********
161  **********
162  **********
163  **********
164  **********
165  **********
166  **********
167  **********
168  **********
169  **********
170  **********
171  **********
172  **********
173  **********
174  **********
175  **********
176  **********
177  **********
178  **********
179  **********
180  **********
181  **********
182  **********
183  **********
184  **********
185  **********
186  **********
187  **********
188  **********
189  **********
190  **********
191  **********
192  **********
193  **********
194  **********
195  **********
196  **********
197  **********
198  **********
199  **********
200  **********
201  **********
202  **********
203  **********
204  **********
205  **********
206  **********
207  **********
208  **********
209  **********
210  **********
211  **********
212  **********
213  **********
214  **********
215  **********
216  **********
217  **********
218  **********
219  **********
220  **********
221  **********
222  **********
223  **********
224  **********
225  **********
226  **********
227  **********
228  **********
229  **********
230  **********
231  **********
232  **********
233  **********
234  **********
235  **********
236  **********
237  **********
238  **********
239  **********
240  **********
241  **********
242  **********
243  **********

BEGIN XRSFL
1  2  SET FOR LEFT SHIFT
2  2  PERFORM SHIFT (FLAG, COUNT, SOURCE)
3  END XRSFL
4  BEGIN XRSFR
5  2  SET FOR RIGHT SHIFT
6  2  PERFORM SHIFT (FLAG, COUNT, SOURCE)
7  END XRSFR
8  BEGIN SHIFT
9  2  TRANSFER CALLING SEQUENCE
10  2  CONSTRUCT SHIFT INSTRUCTION
11  2  LOAD A WITH SOURCE
12  2  CLEAR B
13  2  SHIFT BA AS SPECIFIED
14  END SHIFT
FORTRAN CALLING PROCEDURE

CALL XRUNE (NAME6, NAME4)

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

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

XRUNE - THREE TO SIX CHARACTER PACKED QUALIFIED NAME

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

NAME4 - ONE TO FOUR CHARACTER PACKED NAME WITH PREFIX AND SUFFIX REMOVED

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

NOTES

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

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

BEGIN XRUNE

MOVE CHARACTERS 2-5 OF NAME6 INTO NAME4

IF SIXTH CHARACTER IS BLANK, THEN

LOCATE LAST NON-BLANK CHARACTER OF NAME4

BLANK THAT CHARACTER

ENDIF

END XRUNE
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 IN A2 FORMAT

OUTPUT

UNPKED - ARRAY OF NON-BLANK CHARACTERS IN R1 FORMAT
COUNT - NUMBER OF CHARACTERS IN UNPKED

********

NOTES

USES ENTR

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

** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER

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

CALL XRISP (CHSTR, LENTH)

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

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

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

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

*************
EXTERNAL REFERENCES

*************
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 'O' TO INDICATE
BEGIN/END OF A CHARACTER STRING_THIS BIT IS TURNED 'OFF'
BY THIS PROGRAM PRIOR TO EXIT THIS CHANGE MADE TO
INCORPORATE QUOTE MARKS W/I CHARACTER STRINGS.
OTHER PROCESSORS CHANGED FOR THIS MODIFICATION WERE:
A. XTLAN, XPARM
B. XILSS/XILSD

*************
1 BEGIN XR1SP
2 CONVERT WORD COUNT INTO CHARACTER COUNT
3 SET STORE INDEX TO FIRST POSITION IN CHARACTER STRING
4 CLEAR CHARACTER STRING FLAG
5 DO FOR EACH CHARACTER IN STRING
6 IF CHARACTER IS SPECIAL CHARACTER INDICATING CHARACTER STRING 'THF'
7 REPLACE CHARACTER STRING INDICATOR WITH QUOTE MARK
8 FLIP CHARACTER STRING FLAG
9 CLEAR BLANK FLAG
10 ELSE
11 IF CHARACTER STRING FLAG IS CLEAR, THEN
12 IF CHARACTER IS A BLANK, THEN
13 IF BLANK FLAG IS SET (AT LEAST ONE PREDECESSING BLANK), THEN
14 SKIP THIS CHARACTER (EXIT TO ENDDO)
15 ELSE
16 SET BLANK FLAG
17 ENDF
18 ELSE
19 CLEAR BLANK FLAG
20 ENDF
21 ENDF
22 ENDDO
23 IF NUMBER OF CHARACTERS STORED 'S 000
24 STORE ONE MORE BLANK
25 ENDF
26 CONVERT STORE INDEX TO WORD COUNT AND RETURN
27 END XR1SP
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR : 495</td>
</tr>
<tr>
<td>BLOCK 345</td>
</tr>
<tr>
<td>SHIFT 746</td>
</tr>
<tr>
<td>TRACE 463</td>
</tr>
<tr>
<td>XRBIT 332</td>
</tr>
<tr>
<td>XRCPB 73</td>
</tr>
<tr>
<td>XRD18 112</td>
</tr>
<tr>
<td>XRE8 155</td>
</tr>
<tr>
<td>XREXT 199</td>
</tr>
<tr>
<td>XRE14 249</td>
</tr>
<tr>
<td>XRI6 295</td>
</tr>
<tr>
<td>XRLCK 337</td>
</tr>
<tr>
<td>XRLQ 384</td>
</tr>
<tr>
<td>XMMOV 415</td>
</tr>
<tr>
<td>XMMSG 470</td>
</tr>
<tr>
<td>XRMX 541</td>
</tr>
<tr>
<td>XR66 588</td>
</tr>
<tr>
<td>XR6D 634</td>
</tr>
<tr>
<td>XRFR 687</td>
</tr>
<tr>
<td>XRS 725</td>
</tr>
<tr>
<td>XRSF 757</td>
</tr>
<tr>
<td>XRSD 761</td>
</tr>
<tr>
<td>XRULX 341</td>
</tr>
<tr>
<td>XRUNG 800</td>
</tr>
<tr>
<td>XRUPK 839</td>
</tr>
<tr>
<td>XRISP 912</td>
</tr>
</tbody>
</table>

BXGT F.POLIST
FORTRAN CALLING PROCEDURE

CALL XNLBS(XSENE)

XSENE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR

INPUT

COMMON XE - LU

COMMON XB - DEBUG, INETC, NEWTAB, NUMENT, PRMLEN
  PROMPT, PRMTND, WKBUFF

OUTPUT

COMMON XE - RENBUFF, REAPTR

COMMON XB - NUMENT, PRMLEN, PRMTND, PROMPT, WKBUFF

NOTES

USES ROUTINES

EXEC

XERTM

XREM

XREM

XREM

XREM

XREM

XIMP

XIPCE

XIPDM

XICOM

XICOM

XICOM
1 BEGIN XSER
2 DO UNTIL A % OR 'EXIT' IS ENTERED
3 CALL XSPEN TO BUILD A PROMPT BASED ON PROMT NODE FOR THE
4 NEXT TABLE ENTRY (INDICATED BY TABNDX)
5 CALL XITCOM TO ISSUE THE PROMPT AND RETURN RESPONSE
6 ERREXIT IF ERROR IN XITCOM :ERR10:
7 EXIT XITEN IF X WAS ENTERED
8 IF NOTHING (ONLY CR) ENTERED, THEN
9 IF PROMPT MODE IS NOT 'ALL', THEN
10 CALL XRSN - 'INVALID INPUT'
11 ENDIF
12 ELSE
13 CALL XSPNPT TO PROCESS INPUT BASED ON PROMPT MODE,
14 CURRENT TABLE ENTRY (TABNDX), AND PROMPTED SCIENCE
15 NUMBER (PRNUMBER)
16 ENDIF
17 ENDDO
18 BUILD AWA REQUEST TO DELETE/VERIFY ABSENCE OF NDTAB
19 CALL XSPCK TO PACK THE TABLE BUFFER (REMOVE DELETED ENTRIES)
20 BUILD AWA REQUEST TO ALLOCATE NDTAB
21 IF NUMBER OF TABLE ENTRIES (NENTAB) > 0, THEN
22 CALL XRENC TO PROCESS THE REQUESTS
23 IF THE ALLOCATE REQUEST FAILED, THEN
24 CALL XRSN - 'ANA/DDA FULL, SEQUENCE TABLE NOT STORED'
25 CALL EXEC TO FREE CLAS NO. AND SAM BUFFER
26 ELSE
27 CALL ERRLSN - 'VACUOUS TABLE -- NOT STORED'
28 ENDIF
29 ENDF
30 1 EXIT XSER
31 2 :ERR10: CALL XRSN - 'SYSTEM ERROR'
32 1 END XSER
FORTAN CALLING PROCEDURE

CALL ISPRM

XSPRM BUILDS SEQUENCE TABLE EDITOR PROMPTS.

INPUT

COMMON XE - LU

COMMON XB - DEBUG, NUMENT, PRNTMD, TABNDX, WKBUF

OUTPUT

COMMON XB - PRMLEN, PROMPT, SERN0, TABNDX

USES ROUTINES

XRES

XMOV

XRMSG

XRES

XRDEG
BEGIN XSPRM

IF PROMPT MODE IS ALL, THEN

IF NUMBER OF ENTRIES (NUMENT) > 0, AND
ENTRIES EXIST BEYOND TABLE ENTRY INDEX (TABNDX), THEN
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT TABLE ENTRY INDEX (TABNDX) TO NEXT ENTRY (+7)
ENDO

BUILD PROMPT OF THE FORM ' MMNMM=PROC,INT'
SET PROMPT SEQUENCE NUMBER (SEGNO) TO SEQUENCE NO. OF ENTRY
ELSE
SET PROMPT NO. TO CREATE
ENDIF

ENDIF

IF PROMPT MODE IS CREATE, THEN
SET TABLE ENTRY INDEX (TABNDX) TO NEXT ENTRY (NUMENT * 7 + 1)
IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
IF SEQUENCE NO. OF LAST ENTRY > 32690, THEN
CALL XSPRM - 'UNABLE TO BUILD SEQUENCE NO. > 32700'
SET PROMPT MODE TO UPDATE
ELSE
SET PROMPT SEQUENCE NUMBER (SEGNO) TO NEXT MULTIPLE OF 100
BEYOND SEQUENCE NUMBER OF LAST TABLE ENTRY
ENDIF
ELSE
SET PROMPT SEQ. NO. (SEGNO) TO BE 100
ENDIF

IF PROMPT MODE IS NOT UPDATE, THEN
BUILD PROMPT OF THE FORM ' MMNMM=
ENDIF

ENDIF

IF PROMPT MODE IS UPDATE, THEN
BUILD PROMPT OF THE FORM
SET PROMPT LENGTH TO 0 CAUSING #: PROMPT TO BE ISSUED
ENDIF

1 END XSPRM
156 1 CD0     FORTRAN CALLING PROCEDURE
157 1 CD0
158 1 CD0
159 1 CD0     CALL XSNPT
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 CD1
167 1 CD2
168 1 CD2     INPUT
169 1 CD2
170 1 CD2     COMMON XB - COMBUF, COMPR, LU, TOKENS
171 1 CD2
172 1 CD2     COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, PRMTMD
173 1 CD2     SEENO, TABNOX, WKBUF
174 1 CD2
175 1 CD3
176 1 CD3     COMMON XB - COMPR
177 1 CD3
178 1 CD3     COMMON XB - Insert, IRETC, NUMENT, PMTMD, SEENO,
179 1 CD3     TABNOX, TABSIZ, WKBUF
180 1 CD3
181 1 CD3
182 1 CD3
183 1 CD3
184 1 CD3
185 1 CD3
186 1 CD3     NOTEs
187 1 CD3
188 1 CD5     USES ROUTINES
189 1 CD5
190 1 CD5     XRSNG
191 1 CD5
192 1 CD5     XSED
193 1 CD5
194 1 CD5     XSNM
195 1 CD5
196 1 CD5
197 1 CD5
198 1 CD5
C**********
BEGIN XSNPT
.IO IF PROMPT MODE IS UPDATE, THEN

IF TOKEN INPUT IS AN INTEGER, THEN

ERREXIT IF INTEGER < 1 :ERROR:

RETAIN INTEGER AS SEQUENCE NO. (SEQNO)

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT "=" :ERROR:

INCREMENT TO NEXT TOKEN

START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED

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

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 (SEQNO)

SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY

SET INSERT FLAG TO 1 INDICATING INSERT NEEDED

ORELSE

INCREMENT TO NEXT TABLE ENTRY

END LOOP

SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE

SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE

END SEARCH

CALL XSEN TO REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG

ELSE

ERREXIT IF TOKEN IS NOT A NAME :ERROR:

START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED

EXIT IF NAME INPUT IS DIRECTIVE

SET INDEX TO DIRECTIVE LIST ENTRY

ORELSE

INCREMENT TO NEXT DIRECTIVE

END LOOP

ERREXIT :ERROR:

END SEARCH

CASE (XSLIS, XDDEL, XSPMT, XSUM, XSMRG), INDEX

ENDIF

ELSE, PROMPT MODE IS NOT UPDATE

IF TOKEN IS #, THEN

SET PROMPT MODE TO UPDATE

"F AN & IS INPUT, THEN

ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR:

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 XSEN TO BUILD ENTRY BASE ON INSERT FLAG

ENDIF

ENDIF

EXIT XSNPT

:ERROR: CALL XRMNG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
**FORTRAN CALLING PROCEDURE**

**CALL XSENT**

**XSENT CONSTRUCTS SEQUENCE TABLE ENTRIES**

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

**COMMON XB - DEBUZ, LIBDSZ, NUMENT, PRMTMD, SEGO, TABNDX, WBUF**

**OUTPUT**

**COMMON XE - COMPR**

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

**NOTES**

**USES ROUTINES**

**XRCFR, XREXT, XRMV, XRNSG, XRSET, XRPC, XUE4G**

**BEGIN XSENT**

**ERRXIT IF TOKEN INPUT IS NOT A NAME :ERR01:**

**RETAIN THIS NAME AS PROC. NAME**

**INCREMENT TO NEXT TOKEN**

**START SEARCH UNTIL ALL ENTRIES OF XLIB SEARCHED**

**EXIT IF XLIB ENTRY = PROC. NAME**

**ORELSE**

**INCREMENT TO NEXT ENTRY**

**ENDLOOP**

**ERRXIT :ERR03:**

**ENDSEARCH**

**IF COMMA IS NEXT TOKEN, THEN**

**ERRXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERR04:**

**INCREMENT TO NEXT TOKEN**

**ERRXIT IF NEXT TOKEN IS NOT A NAME :ERR01:**

**RETAIN THIS NAME AS INTERFACE TABLE NAME**

**INCREMENT TO NEXT TOKEN**

**ELSE**

**SET INTERFACE TABLE NAME TO ZERO**
322 3 ENDF
323 3 ERREXIT IF NEXT TOKEN IS NOT END :ERROR:
324 3 IF INSERT FLAG DOES NOT INDICATE REPLACE, THEN (I.E. INSERT OR ADD)
325 4 IF NUMBER OF TABLE ENTRIES (NUMENT) = 150, THEN
326 5 CALL XSPCK TO PACK TABLE BUFFER (REMOVE DELETED ENTRIES)
327 5 IF NUMBER OF TABLE ENTRIES STILL = 150, THEN
328 6 SET PROMPT MODE TO UPDATE
329 6 ERREXIT :ERROR:
330 5 ENDF
331 4 ENDF
332 4 IF INSERT FLAG INDICATES INSERT (+1), THEN
333 5 IF ENTRY ABOVE INDEXED ENTRY (TABNDX) IS MARKED DELETED, THEN
334 6 SET TABLE ENTRY INDEX (TABNDX) TO BE THIS DELETED ENTRY
335 6 SET INSERT FLAG TO 0 INDICATING ENTRY REPLACEMENT
336 6 ELSE
337 6 SET MOVLEN = MIN (5, 150-NUMENT) + 7
338 6 DO FOR ALL TABLE ENTRIES FROM BOTTOM OF TABLE TO TABNDX
339 7 MOVE THE ENTRY DOWN MOVLEN WORDS
340 6 ENDDO
341 6 IF MOVLEN > 7 (I.E. MORE THAN 1 ENTRY), THEN
342 7 MARK FOLLOWING ENTRIES AS DELETED
343 6 ENDF
344 5 ENDF
345 4 ENDF
346 4 SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SEQNO) INPUT/PROMPTED
347 3 ENDF
348 3 MOVE PROC. NAME AND INT. NAME INTO TABLE ENTRY AT TABNDX
349 3 IF INSERT FLAG DOES NOT INDICATE ENTRY REPLACEMENT, THEN
350 4 INCREMENT NUMBER OF ENTRIES (NUMENT) BY 1
351 3 ENDF
352 2 EXIT XSENT
353 3 :ERROR: CALL XRMNG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
354 3 :ERROR: CALL XRMNG - '**** IS NOT A VALID PROCESSOR NAME'
355 3 :ERROR: CALL XRMNG - '*** DOES NOT USE AN INTERFACE TABLE'
356 3 :ERROR: CALL XRMNG - 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
357 2 END XSENT
359  2  CD0  FORTRAN CALLING PROCEEDURE
360  2  CD0  CD0  CALL XSPCK
361  2  CD0  CD0  ********
362  2  CD1  XSPCK COMPACTS THE WORKING BUFFER BY REMOVING ALL SEQUENCE
363  2  CD1  TABLE ENTRIES MARKED FOR DELETION
364  2  CD0  ********
365  2  CD2  INPUT
366  2  CD2  CD2  COMMON IE - LU
367  2  CD2  CD2  COMMON XB - DEBUG, NUMENT, TABNOX, WBUF
368  2  CD2  CD2  ********
369  2  CD3  OUTPUT
370  2  CD3  CD3  COMMON XB - NUMENT, TABNOX, WBUF
371  2  CD3  CD3  ********
372  2  CD5  NOTES
373  2  CD5  CD5  USES Routines
374  2  CD5  CD5  XRMOV
375  2  CD5  CD5  XRMSE
376  2  CD5  CD5  XUDGE
377  2  ********
378  2  CD5  CD5  XSPCK
379  2  CD5  CD5  XSPCK
380  2  CD5  CD5  XSPCK
381  2  CD5  CD5  XSPCK
382  2  CD5  CD5  XSPCK
383  2  CD5  CD5  XSPCK
384  2  CD5  CD5  XSPCK
385  2  CD5  CD5  XSPCK
386  2  CD5  CD5  XSPCK
387  2  CD5  CD5  XSPCK
388  2  CD5  CD5  XSPCK
389  2  CD5  CD5  XSPCK
390  2  CD5  CD5  XSPCK
391  2  CD5  CD5  XSPCK
392  2  CD5  CD5  XSPCK
393  2  CD5  CD5  XSPCK
BEGIN XSPCK

DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED

IF THIS ENTRY IS MARKED DELETED, THEN
SET MOVE LENGTH (MOVLEN) TO 7
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT MOVLEN BY 7

ENDDO

MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
THE DELETED ENTRY
DECIMATE NUMENT BY MOVLEN/7

IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
DECREMENT TABLE INDEX (TABNOX) BY MOVLEN

ENDDO

ENDDO

END XSPCK
CD0  FORTRAN CALLING PROCEDURE
CD0  CALL XSLST
CD0  C******
CD1  XSLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
CD1  CONTAINED IN THE WORKING BUFFER
CD1  C******
CD2  INPUT
CD2  COMMON XE - LU, RERBUF, SUBSTA
CD2  COMMON XB - BENO, DEBUG, ENDO, LISTLU,
CD2  MENTAB, MUMENT, WKBUF
CD2  C******
CD5  NOTES
CD5  USES ROUTINES
CD5  XREG
CD5  XRMOV
CD5  XGET
CD5  XUDBG
CD5  C******
CD5  BEGIN XSLST
CD6  IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
CD6  DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
CD6  SET LIMITS (BENO AND ENDO) OF SEQ. ENTRIES LISTED
CD6  SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
CD6  ENDF
CD7  WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
CD8  IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
CD8  DO FROM BENO TO ENDO
CD8  MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
CD9  IF INT. TABLE NAME = 0, THEN
CD9  SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARs.)
CD9  ELSE
CD9  SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARs.)
CD9  ENDF
CD9  CALL XREG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
CD9  WRITE PRINT BUFFER
CD9  ENDF
CD9  ENDXSLST
BEGIN XSCAN
SET LIST LIMITS (BEGINO AND ENDNO) TO ZERO
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERREXIT IF VALUE IS < 1 :ERRO6:
SET BEGIN LIMIT (BEGINO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERREXIT IF VALUE IS < 1 :ERRO6:
SET END LIMIT (ENDNO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
IF BEGIN LIMIT (BEGINO) = 0, THEN
SET BEGIN LIMIT (BEGINO) TO 1 (BEGINO IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGINO)
SET BEGIN LIMIT (BEGINO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERRO6:
END SEARCH
ENDIF
IF END LIMIT (ENDNO) = 0, THEN
SET END LIMIT (ENDNO) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGINO) TO LAST TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDNO)
SET END LIMIT (ENDNO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERRO6:
END SEARCH
ENDIF
SET RETURN CODE TO INDICATE NO ERROR
SET RETURN CODE TO INDICATE AN ERROR
EXIT XSLIB
:ERRO1: CALL XRMSE - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERRO6: CALL XRMSE - 'INVALID SEQUENCE NUMBER'
:ERRO8: CALL XRMSE - 'INVALID SEQUENCE NUMBER RANGE'
END XSCAN
**FORTRAN CALLING PROCEDURE**

**CALL XSDEL**

**XSDEL IS THE SEQUENCE TABLE EDITOR DELETE DIRECTIVE PROCESSOR**

**INPUT**

**COMMON XE - LU**

**COMMON XB - BEGNO, DEBUG, ENDMOD, IRET, NUMEN**

**OUTPUT**

**COMMON XB - NUMEN, WKBIF**

**COMMON**

**NOTES**

**USES Routines**

**XSCAN**

**XDEBUG**

**BEGIN XSDEL**

**CALL XSCAN TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE**

**IF NO ERROR INDICATED, THEN**

**DO FROM THE BEGIN LIMIT (BEGMO) TO THE ENDMOD LIMIT (ENDMD)**

**MARK THIS SEQ. TABLE ENTRY AS DELETED**

**ENDDO**

**DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR UNTIL A NON-DELETED ENTRY FOUND**

**IF ENTRY IS MARKED DELETED, THEN DECREMENT NUMBER OF TABLE ENTRIES (NUMEN) BY ONE**

**ENDIF**

**ENDIF**

**END XSDEL**
C05
C0000000
BEGIN XSNUM
EXIT IF TOKEN IS NOT EOS :ERROR1:
IF NUMBER OF TABLE ENTRIES (NEMENT) > 0, THEN
SET SEQUENCE NUMBER (SEQNO) TO 100
DO FOR ALL ENTRIES IN TABLE
IF TABLE ENTRY IS NOT MARKED DELETED, THEN
SET SEQ. NO. FIELD OF ENTRY TO SEQUENCE NUMBER (SEQNO)
INCREMENT SEQUENCE NUMBER (SEQNO) BY 100
ENDIF
ENDIF
EXIT XSNUM
:ERROR1: CALL XRMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
END XSNUM
**FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:**

```fortran
CALL XTCOM (PROMPT, PRMLN, RETCOD)
```

**INPUTS FROM CALLING SEQUENCE:**

- **PROMPT** - (INTEGER, PRMLN WORDS) AN ARRAY OF PRMLN WORDS
  USED AS A PROMPT TO THE USER

- **PRMLN** - (INTEGER, 1 WORD) THE LENGTH IN WORDS OF
  THE PROMPT ARRAY. MAXIMUM LENGTH IS 38 WORDS.

**OUTPUTS IN CALLING SEQUENCE:**

- **RETCOD** - (INTEGER, 1 WORD) IS A COMPLETION CODE DEFINED
  AS FOLLOWS:
  0 - NORMAL RETURN, BUFFER CONTAINS RESPONSE
  1 - USER RESPONDED YES, BUFFER CONTAINS RESPONSE UP TO AND INCLUDING %.
  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**

- **COMMSG** - 6 WORD ARRAY CONTAINING THE CONTINUATION MESSAGE
- **PREFIX** - 4 CHARACTERS USED AS PREFIXES TO PROMPT.
- **CORRESPONDING TO CODES IN XE(5)
- **RETCED** - RETURN CODE FROM XIPRM EXTENDED PROMPTING
- **RETC13A** - RETURN CODE FROM XILAN LEXICAL ANALYSIS
- **SUFFIX** - CHARACTERS APPENDED TO END OF PROMPT

**XE COMMON USED:**

```
EQUIVALENCE (XE(1), LUL), (XE(2), ICLAS),
XE(5), (XE(5), LFVL), (XE(145), LONRIP),
XE(145), (XE(145), NOTOK), (XE(146), NOWDS)
```

**XS COMMON USED:**

```fortran
```
EQUIVALENCE (XS(1), INBUF), (XS(21), MOIN),
+ (XS(22), OUTBUF), (XS(122), SSFLAG),
+ (XS(123), INLEM)

INBUF - 80 WORD ARRAY THAT CONTAINS THE USER'S RESPONSE
IN R1 FORMAT
IMLEN - LENGTH OF ORIGINAL PROMPT PASSED TO XPARM
MOIN - 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
NZERO = VALUE REPRESENTING THE INDEX INTO COMBUF
WHERE LENGTH OF SS IS TO BE STORED

SUBROUTINES AND FUNCTIONS CALLED:
EXEC, KCVT, XRMV, XRMG, XRMPK, XTLAM, XPARM

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
  ENDLOOP

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

ENDIF

IF LA RETURN CODE SAYS COMPUTE IS FULL THEN
  CALL XRMSC 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 XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
133 3 CALL EXEC TO READ RESPONSE TO PROMPT
134 4 CALL XRUPK Routines TO CONVERT A2 FORMAT BUFFER TO A1
135 5 IF NUMBER OF WORDS READ IS NOT ZERO THEN
136 6 CALL XTLAN Routines 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
Fortran Calling Procedure for Lexical Analysis:

CALL XTLAM (RETC)

Convert 'INBUF' User's Response to 'COMBUF' of Tokens indicating Characters, Integers, Reals, etc.

Outputs in Calling Sequence:

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

0 - Normal return. Buffer contains response.
1 - User responded 1. Buffer contains response up to and including X.
5 - User requested a continuation.
6 - Extended prompting request was received. Buffer contains response up to and including the request.
7 - Communications buffer is full.
1XX - Error in response at or beyond character XX.
2XX - Overflow/Underflow detected at or beyond character XX.

Internal Variables

COMLEN - Length in words of COMBUF = 256
DBLINT - Double precision location to accumulate an integer value
DBLWD - Double precision location to accumulate an integer and fractional value for double precision or real.
FLGCOM - Comma flag
LCL - Last character was not a comma
LCH - Last character was a comma
FLGEN - End loop flag
CIN - Continue loop
END - End loop
FLGNEG - Negative exponent flag
EXPW - Exponent was positive
EXPN - Exponent was negative
FLGTYP - Type of real value
SGN - Single precision
DP - Double precision
POW - Exponent part of a real number
RELWD - Single precision location for real value
SPCHAR - 25 special character array containing

The hi format representation for:

"+-*/<>%=#?(!)"XX.YXR = DEW
X is a closed bracket
Y is an open bracket
Z is a back slash
1 BEGIN XITLAN
2   BEGIN XITLAN
3   SET END FLAG OFF
4   SET LAST CHARACTER WAS A COMMA ON
5   INITIALIZE INDEX INTO INPUT BUFFER
6   INITIALIZE RETURN CODE TO NORMAL RETURN
7   DO WHILE END FLAG IS OFF
8     IF INPUT BUFFER HAS BEEN COMPLETELY SCANNED THEN
9      IF LAST CHARACTER WAS A COMMA ON A SYMBOLIC STRING IS STILL OPEN THEN
10     SET RETURN CODE TO SAY CONTINUATION REQUESTED
11     ENDIF
12     SET END FLAG ON
13   ELSE
14     ERREXIT IF COMBUF IS FULL PERFORM COMFUL
15     IF INPUT CHARACTER IS A COMMA THEN
16       PERFORM COMPA
17     ELSE
18       SET LAST CHARACTER WAS A COMMA OFF
19       IF INPUT CHARACTER IS AN ALPHABETIC CHARACTER THEN
20         PERFORM ALPHA
21       ELSE
22         IF INPUT CHARACTER IS A DIGIT THEN
23           PERFORM DIGIT
24         ELSE
25           PERFORM SCHARS
26         ENDIF
27     ENDIF
28     ENDIF
29   END
30   ENDDO
31   STORE END OF BUFFER TOKEN IN COMBUF
32   INCREMENT #TOKENS BY 1
331 END XITLAN
34   1 * 35   1 * 36   1 * 37   1 * 38   1 * 39   1 * 40   1 * 41   1 * 42   1 *
43   1 BEGIN COMM
44   SET LAST CHARACTER WAS A COMMA ON
45   ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
46   STORE COMMA TOKEN IN COMBUF
47   INCREMENT #WORDS IN COMBUF BY 1
48   INCREMENT #TOKENS BY 1
49   GET NEXT INPUT CHARACTER
501 END COMM
1 BEGIN ALPHA
2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
3 CALL XRMOV TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
4 DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
5 INPUT CHARACTER IS A NUMERIC OR
6 INPUT CHARACTER IS AN EXCLAMATION POINT) AND
7 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
8 MOVE CHARACTER INTO TEMPORARY BUFFER
9 SET NEXT INPUT CHARACTER
10 ENDDO
11 SET CHARACTER COUNT = 6
12 STORE CHARACTER NAME TOKEN IN COMBUF
13 CALL XRPACK ROUTINE TO PACK CHARACTERS INTO COMBUF
14 INCREMENT WORDS IN COMBUF BY 4
15 INCREMENT TOKENS BY 1
16 END ALPHA
17
18 BEGIN DIGIT
19 INITIALIZE POWER TO ZERO
20 PERFORM DCOL
21 IF INPUT BUFFER IS NOT EXHAUSTED THEN
22 IF INPUT CHARACTER IS A . THEN
23 PERFORM DECPT
24 ELSE
25 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
26 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
27 PERFORM EORM
28 ELSE
29 IF INPUT CHARACTER IS AN "R" THEN
30 PERFORM REPET
31 ELSE
32 PERFORM INTEGR
33 EMDIF
34 ENDIF
35 ELSE
36 PERFORM INTEGR
37 EMDIF
38 END DIGIT
1 BEGIN DCOL
2   SET INTEGER = 0
3   SET COUNTER = 0
4   DO WHILE CHARACTER IS A DIGIT AND
5       INPUT BUFFER IS NOT EXHAUSTED
6       INTEGER = (INTEGER + 10) + CURRENT CHARACTER - 48
7       EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
8       INCREMENT COUNTER BT1
9       GET NEXT CHARACTER
10  END DO
11  1 END DCOL
16  1 *
17  1 *
22  1 BEGIN DECP
23   1 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
24   2 GET NEXT INPUT CHARACTER
25   3 IF INPUT BUFFER IS NOT EXHAUSTED THEN
26     4 IF INPUT CHARACTER IS A DIGIT THEN
27       5 PERFORM DCOL
28       6 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
29     7 EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
30   8 ENDIF
31   3 IF INPUT CHARACTER IS AN "E" OR A "0" THEN
32     4 PERFORM EORD
33     3 ELSE
34     4 PERFORM REAL
35     3 ENDIF
36   3 ELSE
37     3 PERFORM REAL
38     2 ENDIF
39   1 END DECP
1 BEGIN INTEGR
2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFLU
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 INTEGR
9
10 BEGIN REAL
11 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFLU
12 SET REAL = DOUBLE PRECISION * 10 ** POWER
13 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
14 STORE REAL TOKEN IN COMBUF
15 INCREMENT WORDS IN COMBUF BY 3
16 INCREMENT TOKENS BY 1
17 END REAL
18
19 BEGIN DDL
20 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFLU
21 SET DOUBLE = DOUBLE PRECISION * 10 ** POWER
22 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
23 STORE DOUBLE TOKEN IN COMBUF
24 INCREMENT WORDS IN COMBUF BY 4
25 INCREMENT TOKENS BY 1
26 END DDL
27
28 BEGIN REPEP
29 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFLU
30 ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVL
31 STORE REPEAT TOKEN IN COMBUF
32 INCREMENT WORDS IN COMBUF BY 2
33 INCREMENT TOKENS BY 1
34 GET NEXT CHARACTER
35 END REPEP
421 1 BEGIN SCARS
422 2  * CHARACTER TABLE:
423 3  * "-#<>@#=$?() busiest
424 4  N M ARE INVALID CHARACTERS HERE
425 5  X IS A CLOSED BRACKET
426 6  Y IS AN OPEN BRACKET
427 7  Z IS A BACK SLASH
428 8  SET J=1
429 9  STARTSEARCH WHILE J<> #CHARACTERS IN TABLE
430 10  EXIT IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
431 11  INCREMENT J BY 1
432 12  ENDLOOP
433 13  PERFORM INVAL - NO RETURN
434 14  ENDSEARCH
435 15  SET NEXT_FLAG ON
436 16  CASE J:A:;F:+F:;F:;F:;F:;F:;F:;F:;F:;F:;F:;B:;
437 17  * ( ) * = 2  S  T  X  N  N  ;
438 18  (:=F:;F:;C:;C:;E:;E:;F:;F:;F:;INVAL;:=INVAL;:=F:)
439 19  :A:
440 20  SET NEXT_FLAG OFF
441 21  PERFORM QUOTE
442 22  SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED
443 23  SET END_FLAG ON
444 24  SET SYMBOLIC STRING FLAG TO CURRENT COMMAND INDEX + 1
445 25  FALSE
446 26  IF SYMBOLIC STRING FLAG IS OFF THEN
447 27  SET SYMBOLIC STRING FLAG TO CURRENT COMMAND INDEX + 1
448 28  ELSE
449 29  SET COMBUF(SYMBOLIC STRING FLAG) = CURRENT COMMAND INDEX - SYMBOLIC STRING FLAG
450 30  SET J = J+1 TO STORE SYMBOLIC STRING CLOSE TOKEN
451 31  SET SYMBOLIC STRING FLAG OFF
452 32  ENDIF
453 33  :D:
454 34  SET RETURN CODE TO SAY X ENTERED
455 35  SET END_FLAG ON
456 36  :E:
457 37  IF FOLLOWING CHARACTER IS A DIGIT THEN
458 38  SET NEXT_FLAG OFF
459 39  SET INTEGER = 0
460 40  SET POWER = 0
461 41  PERFORM DECPT
462 42  ENDIF
463 43  :F:
464 44  ENDCASE
465 45  IF NEXT_FLAG IS ON
466 46  STORE TOKEN (J) IN COMBUF
467 47  INCREMENT WORDS IN COMBUF BY 1
468 48  IF TOKEN IS BEGIN SYMBOLIC STRING THEN
469 49  INCREMENT WORDS IN COMBUF BY 1
470 50  ENDIF
471 51  INCREMENT #TOKENS BY 1
472 52  SET NEXT CHARACTER
BEGIN QUOTE
GET NEXT CHARACTER
GET #CHARACTERS = 0
DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
(INPUT CHARACTER IS A QUOTE AND
NEXT CHARACTER IS A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
INCREMENT #CHARACTERS BY 1
MOVE CHARACTER INTO TEMPORARY BUFFER
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
ENDDO
ENDOQ
EXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
EXIT IF LENGTH OF CHARACTER STRING IS 0 OR
EXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
STORE CHARACTER STRING TOKEN IN COMBUF
CALL XPCK TO PACK CHARACTERS INTO COMBUF
INCREMENT #WORDS IN COMBUF BY 2+((#CHARACTERS+1)/2)
INCREMENT #TOKENS BY 1
GET NEXT CHARACTER
END QUOTE
FORTAN CALLING PROCEDURE

CALL XTPRM

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

********

INPUT

2D COMMON - CARTR, COMBUF, FLAGS, LU, NOPROC, PRCKAM, SUBSTA,

2D TOKENS

2D XB COMMON - XEPTR, XBUF (FROM INTERFACE TABLE EDITOR)

2D XS COMMON - PRMLEM (FROM XTCOM)

2D VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)

2D LISTING OF APPROPRIATE EXTENDED PROMPTS

********

INTERNAL VARIABLES

2D CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY

2D FILE - FILE NAME OF CURRENT TABLE ENTRY

2D INDEX - INDEX TO CURRENT TABLE ENTRY

2D L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO

2D FIRST LIST ITEM IN RECORD 1 (SEE SDD SECTION 6.2.4.3)

2D MESS - MESSAGE NUMBER, IF ANY, FOR CURRENT TABLE ENTRY

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

2D SECTION 6.2.4.3)

2D RECORD - RECORD NUMBER WITH WHICH TO BEGIN PROCESSING FOR CURRENT

2D TABLE ENTRY

2D SEARCH - LIST SEARCH FLAG (1) FOR CURRENT TABLE ENTRY

2D SIZE - LIST ITEM SIZE OF CURRENT TABLE ENTRY (NEGATIVE INDICATES

2D LAST CHARACTER TO BE MASKED)

2D TABLE - PROCESSING CONTROL TABLE FOR VARIOUS SYNTAX CONDITIONS

********

ENTRY SYNTAX CONDITION INDEX FILE REC SIZE SRC MSG CONTINUE

2D 1 I X: 1 1 I >XPRM 1 2 NO NONE NO I

2D 2 I XR: 1 9 I >XPRM 1 2 YES XTO6 NO I

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

2D 4 I S:, $P:, $P: 1 25 I XLIBD 2 -3 NO NONE NO I

2D 5 I #: 1 33 I >XPRM 1 2 YES NONE YES I

2D 6 I S:, $P:, $P: 1 41 I XPPDC 3 126 NO XTO7 NO I

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

2D 8 I 1: 1 57 I >PROC 1 3 NO NONE NO I

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

2D 10 I $P:?, $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 :ERROR9: IF FAILURE
7   IF TABLE SIZE FIELD < 128 (NOT A LIST RECORD), then
8       IF SIZE > 0 (NO LAST CHARACTER MASKING & POSSIBLE SPANNING), then
9           IF RECORD SPANNED (N-SIZE > 128), then
10              READ SECOND RECORD AND APPEND TO FIRST RECORD DATA
11              EXIT TO :ERROR9: IF FAILURE
12 ENDIF
13 ELSE
14   SET SIZE POSITIVE
15   DO FOR EACH LIST ITEM (1-M)
16      BLANK LAST CHARACTER
17 END DO
18 ENDIF
19 IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), then
20   START SEARCH WHILE LIST ITEMS REMAIN TO BE EXAMINED
21   EXIT IF TOKEN LOCATED IN LIST
22   POSITION TO APPROPRIATE RECORD (I+L-1) AND READ
23   EXIT TO :ERROR9: IF FAILURE
24   SET SIZE TO 128 (ENTIRE RECORD TO BE DisplayED)
25 END LOOP
26 IF TABLE MESSAGE NUMBER FIELD > 0, then
27       CALL XRMNG TO DISPLAY 'NOT VALID ...' MESSAGE
28 ENDIF
29 EXIT TO ENDDO
30 END SEARCH
31 ENDIF
32 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
33       CALL XRMNG TO DISPLAY 'NOT VALID ...' MESSAGE
34 END IF
35 :ERROR9: CALL XRMNG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
36 CLOSE FILE
37 ENDIF
38 ENDDO
39 END XTPRM
BEGIN SETUP
EXIT TO :ERROR2: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
CASE (^): (>: ^: ^:) SUBSTA

:SI:
IF FIRST TOKEN = ?, THEN
SET TABLE INDEX TO FIRST ENTRY
ELSE
SET TABLE INDEX TO SECOND ENTRY
ENDIF

:SI:
IF FIRST TOKEN = ?, THEN
SET TABLE INDEX TO FOURTH ENTRY
ELSE
SET TABLE INDEX TO SIXTH ENTRY
ENDIF

ENDIF

IF PROMPT LENGTH = 0, THEN
ASSUME TABLE INDEX OF THIRD ENTRY
ELSE
ASSUME TABLE INDEX OF FOURTH ENTRY
ENDIF

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

FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN

:SI:
FORM TENTH ENTRY FILE NAME FROM > AND IT EDITOR CURRENT PROCESSOR NAME
ASSUME TABLE INDEX OF TENTH ENTRY
IF FIRST TOKEN = ?, THEN
IF PROMPT LENGTH = 0, THEN
CHANGE TABLE INDEX TO SEVENTH ENTRY
COPY FILE NAME FROM ENTRY TEN TO ENTRY EIGHT
ELSE
CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY IT EDITOR
ENDIF
ELSE
IF PROMPT LENGTH = 0, THEN
CHANGE TABLE INDEX TO NINTH ENTRY
ENDIF
ENDIF
END CASE
EXIT SETUP
:ERROR2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
END SETUP
BEGIN DISPLAY
IF SIZE < 128, THEN
DO UNTIL ALL LIST ITEMS DISPLAYED
BLANK LINE BUFFER
MOVE EIGHT (OR REMAING) ITEMS INTO BUFFER
DISPLAY LINE
ENDDO
ELSE
DISPLAY EXTENDED PROMPT
ENDIF
END DISPLAY
FORTRAN CALLING PROCEDURE

CALL XUDBG (I,U, ID)

* * * * * *

CD3

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

* * * * * *

CD3

INPUT (CALLING SEQUENCE)

CD3

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

* * * * * *

CD3

INPUT (INTERACTIVE)

CD3

OPTION (S,M,E) - ONE ASCII CHARACTER IDENTIFYING XUDBG OPTION
S = SNAP OUT (DUMP) MEMORY
M = MODIFY MEMORY
E = EXIT XUDBG

CD3

START - OCTAL MEMORY ADDRESS IN THE USERS MAP OF FIRST WORD TO BE SNAPPED OR MODIFIED

CD3

STOP - OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPPED

CD3

OUTPUT UNIT - LOGICAL UNIT NUMBER TO WHICH SNAP IS TO BE WRITTEN

CD3

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.

* * * * * *

CD3

OUTPUT (TO 'LU')

CD3

HEADER - *** XUDBG FROM IDIDID

CD3

PROMPTS - (SEE INPUT)

CD3

OUTPUT (TO 'OUTPUT UNIT')

CD3

HEADER - *** XUDBG FROM IDIDID

CD3

SNAP - 102 WORD DUMP FORMATTED LINE (SEE XUPL)

* * * * * *

CD4

BASE - NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE

CD4

CLASS - CLASS I/O NUMBER FOR TERMINAL INPUT

CD4

LENGTH - LENGTH OF CHARACTER STRING BEING MANIPULATED

CD4

LINE - EIGHT CHARACTER BUFFER OF WORD TO BE SNAPPED

CD4

LU - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET

CD4

LUG - LOGICAL UNIT FOR SNAP OUTPUT

CD4

OPTR - ONE CHARACTER EXECUTION OPTION CODE

CD4

ORIGIN - REFERENCE POINT FOR MEMORY ACCESS OFFSET COMPUTATION

CD4

ORG - ADDRESS OF 'ORIGIN'

CD4

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

CD4

STR - 'ORIGIN' RELATIVE BEGINNING OF MEMORY TO BE ACCESSED

CD4

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

* * * * * *

CD3

NOTES

* * * * * *
BEGIN XUDG
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 FOR OUTPUT UNIT
        WRITE SNAP HEADER TO OUTPUT UNIT
        DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
        CALL XUDPL TO FORMAT LINE
        OUTPUT LINE
  END DO
ENDIF
END DO
END XUDG
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**

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

**OUTPUT**
- FORMATTED DUMP OF THE INDICATE PORTION OF THE INDICATED FILE

**NOTES**
- USES EXEC, MAXO, OPEN, POINT, READF, RMPAR, XPRBS, XRNRY, XUPFL
- ANY FILE WITH VARIABLE LENGTH RECORDS WILL BE DUMPED USING A RECORD BUFFER OF 1024 WORDS THUS LIMITING THE MAXIMUM DUMPABLE RECD LENGTH.

**BEGIN** XUPPF

**DO FOREVER**

**READ FILE NAME**

**EXIT XUPPF 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**

**ENDDO**

**END XUPPF**
FORTRAN CALLING PROCEDURE

CALL XUDPL (ADDRESS, LINE, BUFFER)

**********

ADDRESS - TWO WORD INTEGER ARRAY CONTAINING THE ABSOLUTE AND
RELATIVE ADDRESS TO BE FORMATTED WITH THE LINE
LINE - EIGHT WORD ARRAY TO BE CONVERTED TO OCTAL AND ASCII
BUFFER - FIFTY-ONE WORD BUFFER TO HOLD FORMATTED PRINT LINE. MUST
BE BLANKED PRIOR TO FIRST CALL TO XUDPL AND NOT STORED
INTO BETWEEN CALLS TO XUDPL.

**********

BUFFER - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE

COLUMNS CONTENTS
3-6 FIRST ADDRESS
11-16 SECOND ADDRESS
21-32 OCTAL REPRESENTATION OF 'LINE'
87-102 ASCII REPRESENTATION OF 'LINE'

**********

NOTES
USES XREST, XRO6, XRSET

**********

BEGIN XUDPL
CALL XRO6 TO CONVERT EACH WORD OF ADDRESS TO OCTAL
DO FOR EACH WORD OF LINE
CALL XRO6 TO CONVERT WORD TO OCTAL
DO FOR EACH BYTE OF WORD
IF BYTE < 409 OR BYTE > 1368
THEN
REPLACE BYTE WITH ASCII PERIOD
ENDIF
END DO
ENDDO
END XUDPL
CD1          FORTRAN MAIN PROGRAM XVMT IS SCHEDULED BY XUOMP TO PRINT
CD1          A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD1          CVSTT      CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
CD1          CD2          LU NUMBER OF WHERE DUMP IS ON DISK
CD1          CD2          TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
CD1          CD2          CONTAINING THE DUMP
CD1          CD1          OUTPUT
CD1          CD3          FORMATTED DUMP TO LU 6
CD1          CD1          EXTERNAL REFERENCES
CD1          CD5          EXEC
CD1          CD5          RMPAR
CD1          CD5          XRCPR
CD1          CD5          XRMOV
CD1          CD5          XUOPL
CD1          BEGIN XVMT
CD2          CALL RMPAR TO GET LU AND STARTING TRACK NOS.
CD2          READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
CD2          COUNT OF ID-SEGS TO BE DUMPED
CD2          UP TO 7 ID-SEGMENT ADDRESSES
CD2          LOW AND HIGH BASE PAGE ADDRESSES
CD2          LOW AND HIGH MAIN MEMORY ADDRESSES
CD2          POINT TO 2ND SECTOR OF DUMP DATA
CD2          DO UNTIL ALL ID-SEGS PRINTED
CD2          DO UNTIL A EIGHT-WORD LINES PRINTED
CD2          PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
CD2          ENDO
CD2          INCREMENT TO NEXT SECTOR OF DUMP DATA
CD2          ENDOD
CD2          COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
CD2          PERFORM COMPARE AND PRINT FUNCTION
CD2          COMPUTE NTRKS, NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ
CD2          DO UNTIL NTRKS ARE READ
CD2          READ NEXT TRACK FROM DISK
CD2          COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP
CD2          PERFORM COMPARE AND PRINT FUNCTION
CD2          ENDO
CD2          RELEASE THE GLOBALLY ALLOCATED TRACKS
CD2          EXIT XVMT
CD2          BEGIN COMPARE AND PRINT FUNCTION
CD2          DO UNTIL N LINES PROCESSED
CD2          IF NOT 1ST LINE, THEN
CD2          CALL XRCPR TO COMPARE WITH PREVIOUS LINE
CD2          IF LINES ARE IDENTICAL, THEN
CD2          IF THIS IS 1ST OF A SERIES, THEN
CD2          WRITE 'DUPLICATE LINE'
CD2          ENDIF
CD2          ELSE
CD2          CALL XUOPL TO FORMAT THE DUMP LINE
CD2          WRITE FORMATTED DUMP LINE
CD2          ENDIF
CD2          ELSE
CD2          CALL XUOPL TO FORMAT THE DUMP LINE
CD2          WRITE FORMATTED DUMP LINE
CD2          ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 *01 ENTRY XVABN
2 *02 TYPE 1A ROUTINE TO ABORT CURRENT ID AND
3 *01 REQUEST A DUMP FOR THE ID.
4 *01 ENTRY XUOMP
5 *01 TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
6 *01 THE REQUESTED DATA AREAS AND THE CURRENT
7 *01 ID SEGMENT AND ITS FATHER CHAIN SEGMENTS
8 *01 BEGIN XVABN ABEND
9 *02 INPUTS: NONE
10 *02 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
11 * EXTERNALS: $ABRT,$ABRE,$XEN,$LIBR
12 *03 CALL SLIBR REENTRANT ROUTINE CALL
13 *03 GET CURRENT ID FROM XERT (LOCATION 1717)
14 *03 INCREMENT TO ADDRESS BOUNDARIES
15 *03 GET ADDRESS BOUNDARIES
16 *03 STORE := XUOMP PARAMETER LIST
17 *03 PERFORM XUOMP DUMP THIS PARTITION
18 *03 CALL $ABRT ABORT THIS PARTITION
19 *03 CALL $ABRE FREE CURRENT REENTRANT DATA BLOCK
20 1 EXIT TO $XEN GO TO DISPATCHER
21 1 END XVABN
22 1 *
23 1 *
24 1 *
25 1 *
26 1 *
27 1 *
28 1 *
29 1 *
30 1 *
31 1 *
32 1 *
33 1 *
34 1 *
35 1 *
36 1 *
37 1 *
38 1 *
39 1 *
40 1 *
41 1 *
42 1 *
43 1 *
44 1 *
45 1 *
46 1 *
47 1 *
48 1 *
49 1 *
50 1 *
51 1 *
52 1 *
53 1 *
54 1 *
55 1 *
56 1 END XUOMP
58 1 01 TYPE 1A ROUTINE TO CONTROL COMMUNICATION BETWEEN AND
59 1 EXECUTION OF FDS MANAGER AND IT'S ASSOCIATED TASKS
60 1 EXECUTIVE, PROCESSORS, AND UTILITIES)_
61 1 ENTRY XVPM AND XVSTB
62 1 INPUTS
63 1 FROM AN ASSOCIATED TASK
64 1 CALL XVPM(PARMS)
65 1 ASSEMBLY FORM
66 1 JSB XVPM
67 1 DEF #2 RETURN ADDRESS
68 1 DEF PARMS A(PARMS)
69 1 WHERE PARMS ARE P1,P2,P3,P4,P5
70 1 P1 IS THE SERVICE REQUEST
71 1 P2= NORMAL TERMINATION (P2-P5 NOT USED)
72 1 WORK AREA REQUEST (P2-P5 NOT USED)
73 1 EXECUTE A SEQUENCE TABLE
74 1 (P2-P4 HAS TABLE NAME)
75 1 (P5 INDICATES EXECUTION CONTROL IN CLASS I/O BUFFER)
76 1 RESET SEQUENCE POINTES
77 1 (P2 HAS SEQUENCE NUMBER)(P3-P5 NOT USED)
78 1 TERMINATE SEQUENCE (P5-P5 NOT USED)
79 1 TERMINATE FDS FUNCTION (P2-P5 NOT USED)
80 1 ABNORMAL TERMINATION OF ASSOCIATED TASK
81 1 FROM AN FDS MANAGER
82 1 ASSEMBLY FORM
83 1 JSB XVPM
84 1 DEF (RETURN POINT)
85 1 OCT 0
86 1 DEF PARMS A(FDS MANAGER RESPONSE)
87 1 OUTPUTS
88 1 TO AN FDS MANAGER
89 1 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
90 1 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
91 1 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
92 1 MANAGER IS ACTIVATED VIA SLIST
93 1 TO AN ASSOCIATED TASK
94 1 RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
95 1 RETURN ADDRESS IS MOVED INTO ID-SEGMENT Word 9(XSUSP)
96 1 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
97 1 ASSOCIATE TASK IS ACTIVATED VIA SLIST
103 1 BEGIN XVPAM
104 2 CALL BL01RN 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.O.A (PARMS)
109 3 PERFORM XVPAM POST REQUESTOR AND WAIT
110 3 ELSE SET UP TO ACTIVATE MANAGER AND SUSPEND ASSOCIATED TASK
111 3 * CALL SEQUENCE IS RETURN.A (PARMS)
112 3 PERFORM XVPAM POST MANAGER AND WAIT
113 2 ENDF
114 2 * STOP-ID HAS ID-SEGMENT TO BE SUSPENDED,
115 2 * AWAKEN-ID HAS ID-SEGMENT TO BE ACTIVATED.
116 2 CALL $LIST (SCHEDULE, AWaken-ID)
117 2 MOVE RETURN ADDRESS TO XSUSP OF STOP-ID.
118 2 CALL $LIST (WAIT, STOP-ID)
119 1 EXIT TO :$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 MGRFND(FATHER-ID-SEGMENT,COUNT)
129 2 EXITIF COUNT .GT. 0
130 2 SET TARGET-ID TO FATHER-ID-SEGMENT
131 2 ENDDO
132 2 IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
133 2 CALL SYSMG (12,*XV03,SEGMENT-NAME) 'XV03,NAM' REQUESTING PROG
134 2 PERFORM PUMP
135 2 EXIT TO &LIX TO ENABLE AND REDISPATCH
136 2 ENDF
137 2 SET AWAKEN-ID FROM FDS-ENTRY SYSMG
138 2 GET REQUEST PAMRS MOVE INTO ID-SEGMENT
139 2 SET SUBAT FROM CURRENT-ID
140 1 FND XVPMAN
141 1 *
142 1 *
143 1 *
144 1 *
145 1 DECFM XVPMAN
146 2 * DETERMINE IF CALLER IS A VALID FDS MANAGER
147 2 *
148 2 * PERFORM MGRFND (CURRENT-ID,COUNT)
149 2 * COUNT WILL BE 0 FOR NO MATCH.
150 2 * COUNT NOT EQUAL ZERO IMPLIES A MATCH
151 2 * AND FDS-ENTRY HAS MATCHING FDSRC ENTRY ADDRESS
152 2 IF COUNT .LT. 0 THEN CALLER IS NOT A FDS MANAGER
153 2 CALL SYSMG (12,*XV01,SEGMENT-NAME) 'XV01 PROGRAM' REQUESTING PROG.
154 2 PERFORM PUMP
155 2 EXIT TO :SKEP THE DISPATCHER
156 2 ENDF
157 2 FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
158 2 * SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
159 2 IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
160 2 CALL SYSMG (12,*XV02,SEGMENT-NAME) 'XV02 PROGRAM' ASSOCIATED PROG.
161 2 PERFORM PUMP
162 2 EXIT TO :SKEP THE DISPATCHER
163 2 ENDF
164 2 AWAKEN-ID-SEGMENT=SUBAT
165 2 IF MANAGER HAS REQUEST FOR ABORT, THEN
166 2 CALL SABORT FOR CURRENT AF
167 2 ENDIF
168 2 *
169 2 MOVE FDS MANAGERS INPUT PAMRS TO ASSOCIATED TASK ID SEGMENT
170 1 END XVPMAN
172 1 BEGIN PDUMP
173 2  *
174 2  SET A CALL TO XVABM AT CALLER'S SUSPEND POINT
175 2  *
176 2  END PDUMP
177 1 *
178 1 *
179 1 *
180 1 *
181 1 *
182 1 BEGIN MGRFND INPUT IS TARGET-ID
183 2  GET XVSIB TABLE OF ACTIVE FDS MANAGERS
184 2  SET COUNT TO NUMBER IN TABLE NUMBER
185 2  SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
186 2  DO WHILE COUNT .LT. 0
187 2  EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMENT
188 3  SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
189 3  SET COUNT = COUNT - 1
190 2  ENDDO
191 2  * FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
192 2  * COUNT = 0 MEANS NO MATCH
193 1 END MGRFND
194 1 *
195 1 *
196 1 *
197 1 *
198 1 BEGIN XVSIB
199 2  N  EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSIB
200 2  STBMIN DEF N NUMBER-IN-TABLE
201 2  STBAC DEF O NUMBER-ACTIVE
202 2  STBAN DEF O STB RESOURCE NUMBER
203 2  STBES EQU ENTRY START
204 2  STBLU DEF O LUC(TALLY)
205 2  STBLA DEF O LUC(ASCII)
206 2  STBMS DEF O AFDS-MANAGER-ID-SEGMENT
207 2  STBEX DEF O AFDS-EXECUTIVE-ID-SEGMENT
208 2  STBEC DEF O CLASS NUMBER FOR EXEC
209 2  STBAT DEF O A(CURRENT-ASSOCIATED-TASK)
210 2  STBPC DEF O CLAGE-NUMBER FOR PROCESSOR
211 2  STBGS DEF O ENTRY SAVE AREA
212 2  STBEE EQU ENTRY "END"
213 2  STBEL EQU STBEE-STBES ENTRY LENGTH
214 2  STBONE 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 A ENTRY
218 2  DEF 0
219 1 END XVSIB
FORMAN CALLING PROCEDURE:

CALL IXAUT

IXAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE

INPUTS IN COMMON:

XE(5) MASSTA, XE(10) RSTSTR, XE(11) SENDER, XE(12) SEPTEK, XE(14) TABEND, XE(15) NPROC, XE(2) LIBD,
XE(249) SENDO, XE(250) SENLEN, XE(251) SERTAB

OUTPUTS IN COMMON:

XE(5) MASSTA, XE(1) FLGTAB

COMMON USED:

EQUIVALENCE (XE(5), MASSTA)

+XE(10), RSTSTR (XE(11), SENDER)

+XE(12), SEPTEK (XE(14), TABEND)

+XE(15), NPROC (XE(2), LIBD)

+XE(249), SENDO (XE(250), SENLEN)

+XE(251), SERTAB (XE(19), FLGTABLE)

FBS ROUTINES CALLED:

XCEPR, XEXT, XRMOV, XMSG,

XSEXE, XSTO, XSTMP

RTE ROUTINES CALLED:

IOE
1 BEGIN XSAUT
2 IF ENTRY IS FROM A DIRECTIVE THEN
3    SET MASTER STATE TO INDICATE REENTRY
4    DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
5    SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
6    ERREXIT IF PROCESSOR NOT FOUND TO :ERR1:
7    STUFF INTERFACE tableau BIT AND VERSION INTO SEQUENCE TABLE ENTRY
8    ENDDO
9    CALL XSTO TO STORE REVISED SEQUENCE TABLE IN AW ASEXSETB
10   ELSE (/I AM BEING REENTERED FROM INT./)
11    CALL XSTMP TO SET UP TEMPORARY EN-RTM EN-RTAB
12    CALL XsxexA TO EXECUTE FROM TEMPORARY ENTRY
13    IF RESET SEQUENCE NUMBER IS NOT REQUESTED THEN
14      EXIT XSAUT IF TERMINAL ENTRY WAS JUST EXECUTED
15      SET STARTING ENTRY TO NEXT ENTRY
16    ENDDO
17    ENDF
18   ENDIF
19   DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
20    CALL XsxexA TO EXECUTE REMAINDER OF TABLE
21   ENDDO
22 1 EXIT XSAUT
23
24 :ERR1:
25  CALL XRMSTG TO DISPLAY INVALID PROCESSOR NAME
26 1 END XSAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELS (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 XELS 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 (X(E), MARSTA)

FCE ROUTINES USED
XCHR, XCHT, XMMSG, XHMAN

NOTE: CONTAINS DUMMY CALL TO XEXEC
119 1 BEGIN XECNT
120 2 SET MODE TO XREXT OF BITS 12 AND 13 OF MASSTA
121 3 CASE MODE (:MANU:, :SEMI:, :AUTT:, :AUTO:)
122 3 :MANU: CALL XXMAN
123 3 :SEMI: CALL XXSEM
124 3 :AUTT: CALL XXMSG TO DISPLAY INVALID OPTION
125 3 :AUTO: CALL XXAUT
126 2 END CASE
127 2 SET MASTER STATE TO DIRECTIVE LEVEL
128 2 CALL XERTH TO RETURN TO XEXEC **NO RETURN**
129 2 DUMMY CALL XEXEC
130 1 END XECNT
132 1 C0************
133 1 C0
134 1 C0 FORTRAN CALLING SEQUENCE:
135 1 C0 CALL XXDEC (REC)
136 1 C0
137 1 C0************
138 1 C0
139 1 C0 XIDEC DECODES A RESPONSE OF PROCESSOR NAME (INT TABLE NAME)
140 1 C0 INTO A SEQUENCE TABLE ENTRY.
141 1 C0
142 1 C0
143 1 C0************
144 1 C0
145 1 C0 INPUTS IN COMMON:
146 1 C0
147 1 C0 XE(IS) TOKENS, XE(145) COMBUF, XB(1) NOPROC, XB(2) LIBD
148 1 C0
149 1 C0************
150 1 C0
151 1 C0 OUTPUTS IN CALLING SEQUENCE:
152 1 C0
153 1 C0 RETC - RETURN CODE (O IS NORMAL RETURN)
154 1 C0
155 1 C0 OUTPUTS IN COMMON:
156 1 C0
157 1 C0 XE(16) PROCAM, XB(251) SEQTAB
158 1 C0
159 1 C0************
160 1 C0
161 1 C0 COMMON USED:
162 1 C0
163 1 C0 EQUIVALENCE (XE(16), PROCAM),
164 1 C0 * (XE(IS), TKENN), (XE(145) T0KPTR),
165 1 C0 * (XE(145), COMBUF),
166 1 C0 * (XB(1), NOPROC), (XB(2), LIBD ),
167 1 C0 * (XB(251), SEQTAB)
168 1 C0
169 1 C0 FDS ROUTINES USED:
170 1 C0
171 1 C0 XRCPR, XREXT, XRM0V, XRMSG
172 1 C0
173 1 C0 RTE ROUTINES USED:
174 1 C0
175 1 C0 IAND
176 1 C0
177 1 C0************
179 1 BEGIN XXDEC
180 2 INITIALIZE RETURN CODE TO ZERO
181 2 SET SEQUENCE ENTRY TO ZEROS
182 2 ERREXIT IF FIRST TOKEN IS NOT A PROCESSOR NAME TO :ERR1:
183 2 SEARCH LIBRARY DIRECTORY FOR PROCESSOR
184 2 ERREXIT IF NAME NOT FOUND TO :ERR1:
185 2 MOVE PROCESSOR NAME, IT BIT AND VERSION INTO SEQUENCE ENTRY
186 2 IF AN INTERFACE TABLE NAME WAS ENTERED THEN
187 2 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
188 2 ENDIF
189 2 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
190 2 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
191 2 IF AN INTERFACE TABLE IS REQUIRED BUT NOT SPECIFIED THEN
192 2 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'INTAB'
193 2 ENDIF
194 1 EXIT XXDEC
195 2 :ERR1:
196 2 CALL XRNSG TO DISPLAY ERROR
197 2 SET RETURN CODE TO SAY ERROR
198 1 END XXDEC
200 1 CD************
201 1 CD0
202 1 CD0 FORTRAN CALLING PROCEDURE:
203 1 CD0
204 1 CD0 CALL XXDEF
205 1 CD0
206 1 CD************
207 1 CD1 XXDEF READS IN THE DEFAULT INTERFACE TABLE FOR A PROCESSOR
208 1 CD1 AND STORES IT IN THE AWS AS BINTAB
209 1 CD1
210 1 CD1
211 1 CD************
212 1 CD2 INPUTS FROM COMMON:
213 1 CD2 PROCNAM - (INTEGER, 3 WORDS) NAME OF PROCESSOR IN SERTAB
214 1 CD2 FOR WHICH NO INTERFACE TABLE WAS SUPPLIED
215 1 CD2
216 1 CD2
217 1 CD2
218 1 CD************
219 1 CD4 INTERNAL VARIABLES:
220 1 CD4
221 1 CD4
222 1 CD4 DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE
223 1 CD4 DEFAULT INTERFACE TABLE CAN BE READ INTO
224 1 CD4 DEFNAM - (INTEGER, 3 WORDS) ARRAY WHERE INTERFACE TABLE
225 1 CD4 NAME IS CREATED FROM PROCESSOR NAME
226 1 CD4
227 1 CD4
228 1 CD5 COMMON USED:
229 1 CD5
230 1 CD5
231 1 CD5 EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA),
232 1 CD5 + (XE(13), INTNAM), (XE(16), PRCNAM),
233 1 CD5 + (XE(19), REPTR), (XE(20), REBBUF),
234 1 CD5 + (XE(142),ICR ),
235 1 CD5 + (XS(6), TMPTAB), (XS(14), DEFNAM),
236 1 CD5 + (XS(18), LEN1 ), (XS(19), LEN2 ),
237 1 CD5 + (XS(20), IDC), (XS(200),RET$ )
238 1 CD5
239 1 CD5 RTE ROUTINES USED:
240 1 CD5
241 1 CD5 CLOSE, EXEC, KCVT, OPEN, READF
242 1 CD5
243 1 CD5 FDS ROUTINES USED:
244 1 CD5
245 1 CD5 XERT, XREX, XREX, XRMV, XRMV, XRPCK, XRUPK
246 1 CD5
247 1 CD************
BEGIN XDEF
CREATE DEFAULT INTERFACE TABLE NAME FROM PROCESSOR NAME
CALL OPEN TO OPEN FILE
ERREXIT IF OPEN ERROR TO :FILERR:
CALL READ TO READ RECORD 1
ERREXIT IF READ ERROR TO :FILERR:
IF THERE IS LITERAL DATA THEN
CALL READ TO READ RECORD 2
ERREXIT IF READ ERROR TO :FILERR:
ENDIF
CALL CLOSE TO CLOSE FILE
ERREXIT IF CLOSE ERROR TO :FILERR:
CALL XREG TO ALLOCATE & STORE GINTAB
ERREXIT IF RETURN CODE IS NOT ZERO TO :NGRERR:
IF DEFAULT TABLE IS INCOMPLETE THEN
SET INTERFACE TABLE NAME IN XE TO GINTAB
SET SUBSTATE TO INTERFACE TABLE EDITOR
CALL XERTN TO IMPICITLY CALL INT EDITOR **NO RETURN**
ENDIF
EXIT XDEF

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

:NGRERR:
CALL XRMSG TO DISPLAY SPACE ERROR
CALL EXEC TO FREE CLASS NUMBER
SET MASTER STATE TO DIRECTIVE LEVEL
CALL XERTN TO RETURN TO EXEC **NO RETURN**
END XDEF
BEGIN XXMAN

1. IF ENTRY IS FROM A DIRECTIVE THEN
2. SET HASHTO TO INDICATE RE-ENTRY
3. DO UNTIL PERCENT IS ENTERED

4. :PROMPT: CALL XICOM TO PROMPT FOR PRNAME, *ITNAME
5. IF PERCENT IS NOT ENTERED THEN
6. ERREXIT IF CR ENTERED TO :PROMPT:
7. CALL XXDEC TO DECODE RESPONSE
8. ERREXIT IF INVALID RESPONSE TO :PROMPT:
9. SET SER #5 IN ;E: ) ZEROS
10. SET #ENTRIES IN SERTAC IN XE TO 1
11. CALL XXSTO TO STORE SEQUENCE TABLE
12. IF IT NAME IN SERTAB IS &INTAB THEN
13. PROCESSOR REQUIRES AN IT THEN
14. CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
15. ENIF
16. CALL XXERE TO EXECUTE SERTAB
17. ENDDO
18. ELSE
19. AM BEING REENTERED FROM INTERFACE TABLE EDITOR
20. CALL XXMP TO SET UP TO EXECUTE A TEMPORARY TABLE
21. PERFORM XXMAN **NO RETURN**
22. ENIF
23. RETURN

24. :PRERR: CALL XRMSG TO DISPLAY ERROR
25. PERFORM XXMAN **NO RETURN**
26. END XXMAN
**FORTRAN CALLING SEQUENCE:**

```fortran
CALL XXSEM
```

**INPUTS IN COMMON:**

- XE(5) MASTA, XE(10) SERST, XE(11) SEREM, XE(12) SERET
- XE(140) TABEND, XE(1) NOPROC, XE(2) LIND
- XE(245) SERNO, XE(250) SERLEM, XE(251) SERTAB

**OUTPUTS IN COMMON:**

- XE(5) MASTA, XE(13) FLGTAB

**INTERNAL COMMON USED:**

- XE(139) EXEMD - ENDING SEQUENCE NUMBER USED TO TERMINATE SEQUENCE
- XE(161) CURIND - INDEX TO THE CURRENT ENTRY BEING EXECUTED
- XE(235) RESIND - INDEX TO RESET CURRENT INDEX TO
- XE(236) ASCENT - ASCENT OF XXII SEQUENCE TABLE ENTRY USED TO PROMPT THE USER
- XE(246) OLDIND - INDEX TO THE LAST ENTRY EXECUTED IN THE SEQUENCE TABLE

**COMMON USED:**

- + (XE(10), SERST), (XE(11), SEREM), (XE(12), SERET)
- + (XE(14), SEREM), (XE(245), SERNO), (XE(250), SERLEM), (XE(251), SERTAB)
- XXSEM
- XXDEF, XXEXF

**FDS ROUTINES USED:**

- ZRCPX, ZRXT, ZRIO, ZRMDV
- ZMAGP, ZSETV, ZXDEF, ZXSEM
1 BEGIN XSEM:
2   IF ENTRY IS FROM A DIRECTIVE THEN
3     SET MASTER STATE TO SAT REENTRY
4     DO FOR # ENTRIES IN SEQUENCE TABLE
5     SEARCH LOAD FOR PROCESSOR NAME
6     EXIT IF NOT FOUND TO :ERR:
7     MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
8   ENDIF
9   SAVE ORIGINAL ENDING SEQUENCE #
10  SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
11  SET FLSTAB TO SAT EXECUTE ENTIRE SEQUENCES
12  CALL XSTO TO STORE READY IN AREA
13 ELSE (I AM BEING REENTERED FROM INTE)
14   CALL XETyp TO SET UP TEMPORARY EXECUTIN
15   CALL XERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQUENCES
16   IF RESET WAS REQUESTED THEN
17     RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
18   ELSE
19     EXIT XSTO IF TERMINAL ENTRY WAS JUST EXECUTED
20     SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
21   ENDIF
22 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
23   CALL XECOM TO PROMPT USER WITH CURRENT ENTRY
24 EXIT XSEM IF RESPONSE IS X
25   IF RESPONSE IS CR THEN
26     IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
27       CALL XIDF TO READ UP DEFAULT TABLE
28     ENDIF
29     SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
30     SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
31     CALL XERE TO EXECUTE
32     SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
33     ELSE IF RESPONSE WAS A SEQUENCE # THEN
34       EXIT IF NUMBER IS ZERO TO :ERR:
35       SEARCH SEQUENCE TABLE FOR SEQUENCE #
36       EXIT IF NUMBER IS NOT FOUND TO :ERR:
37       SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
38     ELSE
39       IF RESPONSE IS AN AMPSAND THEN
40         SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
41       ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
42         CALL XEDC TO DECODE PROCESSOR NAME, IT NAME RESPONSE
43       IF RESPONSE IS VALID THEN
44         CALL XSTO TO STORE OVERRIDING ENTRY
45         SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
46         SET CURRENT SEQUENCE # TO FIRST ENTRY
47       ENDIF
48       ENDIF
49     ENDIF
50   ENDIF
51   ENDIF
52 END DO
53 EXIT XSEM
54   :ERR:
55   CALL XEMSG TO DISPLAY ERROR MESSAGE
56 END XSEM
CALLING PROCEDURE FOR XXSTO:

CALL XXSTO

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

INPUTS FROM COMMON:

XB(250) SERLEN, XB(251) SECTAB, X5(13) FLGTAB

OUTPUTS TO COMMON:

XE(5) MASSTA

COMMON USED:

EQUIVALENCE (XE(5), MASSTA),
+ (XE(19), REGPTR), (XE(20), RENBUF),
+ (XB(250), SECLEN), (XB(251), SECTAB),
+ (X5(13), FLGTAB)

FDS ROUTINES USED:

XREQ, XRN0V, XRM0G, XERTN

RTE ROUTINES USED:

EXEC
1 BEGIN X5STO
2 SET CLASS NUMBER TO ZERO
3 IF TABLE FLAG SAYS STORE ENTRIE TABLE THEN
4 CALL EXEC TO WRITE ENTIRE TABLE
5 SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN XB
6 CALL XMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER
7 ELSE
8 CALL EXEC TO WRITE ONLY FIRST ENTRY
9 SET LENGTHS IN REQUEST BUFFER TO 7 WORDS
10 CALL XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER
11 ENDBF
12 CALL XERGTO REQUEST MANAGER TO STORE .XERTB
13 IF RETURN CODE IS NOT ZERO THEN
14 CALL XMOV TO WRITE SPACE ERROR
15 CALL EXEC TO RELEASE CLASS 0
16 SET MASTER STATE TO ZERO
17 CALL XERGTO RETURN TO EXEC **NO RETURN**
18 ENDF
19 2 RETURN
20 1 END X5ST0
FORTRAN CALLING SEQUENCE:

CALL XITMP

XITMP SETS UP A ONE ENTRY SEQUENCE TABLE USING LINTAB AND
STORES IT IN THE AWA TO EXECUTE WHEN EXECUTION WITH A
TEMPORARY ENTRY IS NECESSARY

INPUTS FROM COMMON:
N1(12) SERPTR, XB2(250) SEGLEM, XB2(251) SERTAB

OUTPUTS TO COMMON:
XE(6) SBSBA, XB2(249) SEANO, XB2(250) SEGLEM,
XB2(251) SERTAB, XS(13), FLGTAB

COMMON USED:
EQUIVALENCE (XE(6), SBSBA), (XEC(12), SERPTR),
+ (XEC(19), SERPTR), (XEC(20), REBUF),
+ (XB2(249), SEGNO), (XB2(250), SEGLEM),
+ (XB2(251), SERTAB), (XE(6), TMPTAB),
+ (XS(13), FLGTAB)

FDS ROUTINES USED:
XREG, XHMOV, XHSTO

RTE ROUTINES USED:
EXEC
<table>
<thead>
<tr>
<th>SYMBOL DEFINITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO : 125</td>
</tr>
<tr>
<td>AUTT : 124</td>
</tr>
<tr>
<td>ERR1 : 70</td>
</tr>
<tr>
<td>ERR2 : 243</td>
</tr>
<tr>
<td>ERR3 : 195</td>
</tr>
<tr>
<td>ERR4 : 565</td>
</tr>
<tr>
<td>ERR5 : 347</td>
</tr>
<tr>
<td>ERR6 : 269</td>
</tr>
<tr>
<td>INT1 : 339</td>
</tr>
<tr>
<td>INT2 : 344</td>
</tr>
<tr>
<td>MANU : 122</td>
</tr>
<tr>
<td>MGRERR : 273</td>
</tr>
<tr>
<td>PMERR : 439</td>
</tr>
<tr>
<td>PROMPT : 418</td>
</tr>
<tr>
<td>RESET : 349</td>
</tr>
<tr>
<td>SEMI : 123</td>
</tr>
<tr>
<td>XIAUT : 49</td>
</tr>
<tr>
<td>XIENT : 119</td>
</tr>
<tr>
<td>XIDEC : 179</td>
</tr>
<tr>
<td>XIDEF : 249</td>
</tr>
<tr>
<td>XIEXEC : 327</td>
</tr>
<tr>
<td>XIDMAN : 414</td>
</tr>
<tr>
<td>XISEM : 509</td>
</tr>
<tr>
<td>XISTO : 609</td>
</tr>
<tr>
<td>XITMP : 675</td>
</tr>
</tbody>
</table>

*DXOT F.POLIST*
CCD************

ASSGN - DATA ASSIGNMENT PROCESSOR

- SCHEDULED BY FDS

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

ASSGN ALLOWS THE FDS USER TO COMPUTE VALUES AND STORE THEM IN
AN EXISTING DATA ELEMENT IN THE ANA. ASSGN SUPPORTS EXTENDED
FORTRAN TYPE MIXED-MODE EXPRESSIONS AND FUNCTIONS AND ALLOWS
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-NAPIER DEFINITION OF VALID
SYNTAX)

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

OUTPUTS TO THE ANA:

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

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

INTERNAL VARIABLES:

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

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIMENSION</th>
<th>START</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARMS</td>
<td>5</td>
<td>1</td>
<td>PARMS(1) - LU, PARMS(7) = DEBUG FLAGS</td>
</tr>
<tr>
<td>TOKERS</td>
<td>32</td>
<td>6</td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STDERR</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>
</tbody>
</table>
| SYMTAB | 12,81 | 41 | SYMBOL TABLE (WORDS 1-8 = TBC ENTRY OR
APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR
SUBSCRIPTED DATA ELEMENT) |
| SSTRNG | 247 | 1013 | RESULT STACK USED DURING POST-
FIX STRING EVALUATION (EACH EN-
TRY: WORDS 1-3 CONTAIN VALUE; |
<p>| RESULT | 4,35 | 1260 | WATER 4 = DATA TYPE). DATA TYPE= |
| 1-2,3 | FIXED DATA |
| -1 | SYMBOL TABLE INDEX |</p>
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>1 CD4</td>
</tr>
<tr>
<td>62</td>
<td>1 CD4</td>
</tr>
<tr>
<td>63</td>
<td>1 CD4</td>
</tr>
<tr>
<td>64</td>
<td>1 CD4</td>
</tr>
<tr>
<td>65</td>
<td>1 CD4</td>
</tr>
<tr>
<td>66</td>
<td>1 CD4</td>
</tr>
<tr>
<td>67</td>
<td>1 CD4</td>
</tr>
<tr>
<td>68</td>
<td>1 CD4</td>
</tr>
<tr>
<td>69</td>
<td>1 CD4</td>
</tr>
<tr>
<td>70</td>
<td>1 CD4</td>
</tr>
<tr>
<td>71</td>
<td>1 CD4</td>
</tr>
<tr>
<td>72</td>
<td>1 CD4</td>
</tr>
<tr>
<td>73</td>
<td>1 CD4</td>
</tr>
<tr>
<td>74</td>
<td>1 CD4</td>
</tr>
<tr>
<td>75</td>
<td>1 CD4</td>
</tr>
<tr>
<td>76</td>
<td>1 CD4</td>
</tr>
<tr>
<td>77</td>
<td>1 CD4</td>
</tr>
<tr>
<td>78</td>
<td>1 CD4</td>
</tr>
<tr>
<td>79</td>
<td>1 CD4</td>
</tr>
<tr>
<td>80</td>
<td>1 CD4</td>
</tr>
<tr>
<td>81</td>
<td>1 CD4</td>
</tr>
<tr>
<td>82</td>
<td>1 CD4</td>
</tr>
<tr>
<td>83</td>
<td>1 CD4</td>
</tr>
<tr>
<td>84</td>
<td>1 CD4</td>
</tr>
<tr>
<td>85</td>
<td>1 CD4</td>
</tr>
<tr>
<td>86</td>
<td>1 CD4</td>
</tr>
<tr>
<td>87</td>
<td>1 CD4</td>
</tr>
<tr>
<td>88</td>
<td>1 CD4</td>
</tr>
<tr>
<td>89</td>
<td>1 CD4</td>
</tr>
<tr>
<td>90</td>
<td>1 CD4</td>
</tr>
<tr>
<td>91</td>
<td>1 CD4</td>
</tr>
<tr>
<td>92</td>
<td>1 CD4</td>
</tr>
<tr>
<td>93</td>
<td>1 CD4</td>
</tr>
<tr>
<td>94</td>
<td>1 CD4</td>
</tr>
<tr>
<td>95</td>
<td>1 CD4</td>
</tr>
<tr>
<td>96</td>
<td>1 CD4</td>
</tr>
<tr>
<td>97</td>
<td>1 CD4</td>
</tr>
<tr>
<td>98</td>
<td>1 CD4</td>
</tr>
<tr>
<td>99</td>
<td>1 CD4</td>
</tr>
<tr>
<td>100</td>
<td>1 CD4</td>
</tr>
<tr>
<td>101</td>
<td>1 CD4</td>
</tr>
<tr>
<td>102</td>
<td>1 CD4</td>
</tr>
<tr>
<td>103</td>
<td>1 CD4</td>
</tr>
<tr>
<td>104</td>
<td>1 CD4</td>
</tr>
<tr>
<td>105</td>
<td>1 CD4</td>
</tr>
</tbody>
</table>

Note: Stacks used in the Assgn processor are sized for the maximum possible and overflow is not tested.

Table Definitions:
## Syntax - Table

<table>
<thead>
<tr>
<th>Token</th>
<th>Input Length</th>
<th>Output Length</th>
<th>Parentheses</th>
<th>Precedent Type</th>
<th>Priority Precedents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>3</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>4</td>
</tr>
<tr>
<td>)</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>10000</td>
</tr>
<tr>
<td>-</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>10000</td>
</tr>
<tr>
<td>~</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>10000</td>
</tr>
<tr>
<td>=</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>10000</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>300000</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
<td>4</td>
<td>SAME</td>
<td>SAME</td>
<td>200000</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>200000</td>
</tr>
<tr>
<td>OPERATOR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>200000</td>
</tr>
</tbody>
</table>

**Reproducibility of the original page is poor.**
<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>CODES (0000-1)</th>
<th>000000000000000000000000000000000000000000000000000000000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACTOR - 7.36</td>
<td>ABST</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>ALOG</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>ATAN</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>DBLQ</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>DBOT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>DLOG</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>DINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>FINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>GQ</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>HINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>LOG</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>LPINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>MINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>NINT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>NQ</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>SIGN</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>SQRT</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>TAN</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>TQ</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
<tr>
<td></td>
<td>Z</td>
<td>0B03 19048905990699079908990999109911991299139914991599169</td>
</tr>
</tbody>
</table>

5-328
Routines referenced - RMPAR, XPGET, Xamarin, ZIPS1, ZIPS2, XPLIT

Backus-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 //

<Additive Operator> ::= + // -

<Multiplicative Operator> ::= * //

<Operand> ::= FUNCTION NAME <LB> <Function List> <RB> //
<VARIABLE> //
<Constant>

<LB> ::= LEFT BRACKET
<RB> ::= RIGHT BRACKET

<Function List> ::= <Function List>,<Expression> //
<Expression>
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>272</td>
<td>BEGIN ASSGN</td>
</tr>
<tr>
<td>273</td>
<td>INITIALIZE COMMON</td>
</tr>
<tr>
<td>274</td>
<td>CALL XIPST TO BUILD POST-FIX STRING</td>
</tr>
<tr>
<td>275</td>
<td>CALL ZIPST2 TO EVALUATE EXPRESSION AND STORE VALUE(S)</td>
</tr>
<tr>
<td>276</td>
<td>CALL XPRIT TO EXIT PROFESSOR</td>
</tr>
<tr>
<td>277</td>
<td>END ASSGN</td>
</tr>
</tbody>
</table>

- ARENm
DDSP - DATA BOX DISPLAY PROCESSOR
- SCHEDULED BY FDS

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

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

XIDIM - READS AND EDITS INTERFACE TABLE

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

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

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

INPUTS TO DDSP FROM INTERFACE TABLE:

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

INPUTS TO DDSP FROM ORDER FILE:

RECORD 1

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

RECORD 2

(1M) - NO OF ENTRIES IN SUMMARY TABLE
(2) - X SCAN VARIABLE (6 CHAR)
(5) - X FIRST SUBSCRIPT (INT OR ZERO)
(6) - 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)
(21) - Y CENTROID (REAL)
(25) - YX INCREMENT (REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPENDANT VARIABLE NAME AND UNITS IN RECORDS 38 48 AND 5

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

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

RETC - RTW TOTP IN DBDSP. DBDSP. GOSUM. EXIT TO EXECUTION DBDSP, JERROR
STRING - CHAR STRING CONTAINING USER PROMPT MESSAGE
XCOORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
XSCNMN - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
397  1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHA)
398  1 CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 1 REAL)
399  1 CD 4 YSCANNED - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHA)
400  1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHA)
401  1 CD 4 ZTABLE - TABLE IN COMMON FOR SMTAB VARIABLE NAMES AND UNITS
402  1 CD 4 NAMVUL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN
403  1 CD 4 SMTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
404  1 CD 4 PARS - COMMUNICATION BUFFER FOR RMPAR - LU, USER ID, FLAGS
405  1 CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCAT
406  1 CD 4 LUSHP - DDBSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
407  1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRM
408  1 CD 4 DEBUG
409  1 CD 4 SELECT - SELECT =0 PROMPT ; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
410  1 CD 4 WITHOUT PROMPT
411  1 CD 4 CARTAG - CARTRIDGE USED TO LOCATE DATA BOX
412  1 CD 4 CALVE -
413  1 CD 4 ICMD -
414  1 CD 4 USES ROUTINES
415  1 CD 5
416  1 CD 5 XPRM, XELBS, XPIXIT, RMPAR
417  1 CD 5
418  1 CD 5
419  1 CD 5
420  1 CD 5
421  1 CD 5
DEFIN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

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

INPUTS FROM THE MANAGER:

LOGICAL UNIT OF THE USER'S TERMINAL
FLAGS FOR DEBUG

INPUTS FROM THE INTERFACE TABLE:

SYMBOLIC STRING CONTAINING DATA ELEMENT NAME(S), OPTIONAL I AND J DIMENSIONS AND A REQUIRED TYPE

OUTPUTS TO THE AWA:

SET OF DATA ELEMENT(S) REQUESTED

INTERNAL VARIABLES:

INTERFACE TABLE HEADER
LENGTH OF SYMBOLIC STRING
POSITION WITHIN THE SYMBOLIC STRING
BUFFER FOR VALID NAMES TO BE ALLOCATED
BUFFER AREA FOR AGET AND XPEN USE
NUMBER OF ENTRIES IN THIS AWA REQUEST
NUMBER OF NAMES IN SYMBOLIC STRING
TOKEN POSITION FOR NEXT NAME
SYMBOLIC STRING INPUT TO DEFINE

EXTERNAL ROUTINES USED:

EXEC, IAMD, KCYT, KMPAR, XPSET,
XPRES, XPXIT, XUNOG, XIOFT, XINSG
1 BEGIN DEFIN
2 CALL RMAR TO GET LU AND DEBUG FLAGS
3 CALL APGET TO RETRIEVE SYMBOLIC STRING
4 GET STRING LENGTH FROM INTERFACE TABLE HEADER
5 STARTSEARCH WHILE TOKEN-POSITION .LT. STRING-LENGTH, OR
6 WHILE CURRENT-TOKEN .NE. END-OF-STRING
7 PERFORM ZIDPM TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
8 PERFORM ZIDPM TO PROCESS THE CURRENT NAME
9 EXIT IF THERE WAS AN ERROR RETURN FROM ZIDPM
10 CALL ZIRNL TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
11 SET PROCESSOR RETURN CODE TO ABEND
12 ORELSE
13 INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
14 ENDDO
15 SET PROCESSOR RETURN FOR NORMAL EXIT
16 ENDS
17 SET OPTION TO CLOSE REQUEST BUFFER
18 ENDIF
19 COMPUTE INDEX TO THIS REQUEST
20 CALL XPTR TO QUEUE THIS REQUEST
21 ENDDO
22 CALL XPRIT TO RETURN TO THE MANAGER
23 1 END DEFIN
510 1 BEGIN XIDNN
511 2 SET NEXT NAME TO ZERO
512 3 DO WHILE NEXT NAME IS ZERO
513 4 CALL XIDFT TO FIND THE NEXT LEFT PAREN
514 5 CALL XIDFT TO FIND THE NEXT COMMA
515 6 IF THERE ARE NO SUBSCRIPTS (COMMA PRECEDES LEFT PAREN) THEN
516 7 SET NEXT NAME TO COMMA POSITION + 1
517 8 ELSE
518 9 CALL XIDFT TO FIND THE NEXT RIGHT PAREN
519 10 SET STARTING POSITION TO RIGHT PAREN POSITION + 1
520 11 IF STARTING POSITION > END OF STRING THEN
521 12 SET STARTING POSITION = END OF STRING
522 13 ENDIF
523 14 ENDF
524 15 ENDDO
525 1 END XIDNN
EMSC IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER.

EMSC PROCESSOR IS USED IN CONJUNCTION WITH SCAN ONLY. IT MARKS THE END OF THE SERIES OF PROCESSORS TO BE SCANNED. IT HAS NO INTERFACE TABLE.

INPUTS FROM THE MANAGER:

LU - LOGICAL UNIT OF THE USER'S TERMINAL
USEID - USER ID CODE
FLAGS - FLAGS FOR DEBUG

INPUTS FROM AWA:

&SCNTB - SCAN CONTROL TABLE CREATED BY SCAN

OUTPUT TO MANAGER:

XXRET - 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, RPMAP, WRIT

FDS ROUTINES USED:

XPXIT, XREX, XRMOV, XUDBG, XVPFM,
XIFCL, XIMSG
641 1 BEGIN ENDS
642 2 CALL XRPAH TO GET INPUTS FROM MANAGER
643 2 SAVE REQUEST TO RETRIVE $SCNTB FROM ASA
644 2 CALL XRPM TO REQUEST ANSI MANAGEMENT
645 2 ERREXIT IF $SCNTB NOT FOUND TO :ERR4:
646 2 CALL EXEC TO READ $SCNTB
647 2 SAVE REQUEST TO RETRIVE $SCNTAB IN REGBUF
648 2 CALL XRPM TO REQUEST ANSI MANAGEMENT
649 2 ERREXIT IF NOT FOUND TO :ERR4:
650 2 CALL EXEC TO READ $SCNTAB
651 2 IF SUMMARY TABLE IS LARGER THAN 32 ENTRIES THEN
652 2 SET SIZE OF $SCNTAB TO 32 ENTRIES
653 2 ENDF
654 2 CALL WRITE TO WRITE $SCNTAB TO DATBOX
655 2 ERREXIT IF WRITE ERROR TO :ERR4
656 2 IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
657 2 THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
658 2 CALL READ TO READ HEADER RECORD
659 2 ERREXIT IF READ ERROR TO :ERR4
660 2 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
661 2 CALL WRITE TO WRITE UPDATED HEADER
662 2 ERREXIT IF WRITE ERROR TO :ERR4
663 2 CALL CLOSE TO CLOSE DATBOX
664 2 SAVE REQUEST TO DELIVER ABS $SCNTB IN REGBUF
665 2 IF THERE ARE REMAINING SCANS I: $SCNTB THEN
666 2 CALL EXEC TO WRITE REMAINING $SCNTB
667 2 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCNTB
668 2 ENDF
669 3 SET RETURN PARAMETER TO NORMAL RETURN
670 ELSE
671 3 PERFORM SETXY
672 2 ENDF
673 2 CALL XRPM TO REQUEST ANSI MANAGEMENT
674 2 CALL XPXIT TO TERMINATE WITH RETURN PARAMETERS
675 1 EXIT ENDS
676 2 ERR4:
677 2 CALL XMSG TO DISPLAY ERROR
678 2 CALL XPXIT TO ABEND PROCESSOR
679 1 END ENDS
BEGIN SETXY
1 IF XCUR IS END STEP THEN
2 IF THERE IS 1 VARIABLE THEN
3 SET XCUR TO ZERO
4 SET X TO CENTROID
5 CALL POSN TO POSITION FILE TO CENTROID RECORD
6 ERREXIT IF POSN ERROR TO :ERR:
7 ELSE
8 IF YCUR IS END STEP THEN
9 SET YCUR TO ZERO
10 SET Y TO CENTROID
11 CALL POSN TO POSITION TO CENTROID RECORD
12 ERREXIT IF POSN ERROR TO :ERR:
13 ELSE
14 SET XCUR TO (-XSTEP)
15 IF XSTEP IS ZERO AND Y IS CENTROID THEN
16 INCREMENT YCUR BY 1
17 CALL WRTIF TO WRITE Dummy AS CENTROID RECORD
18 ERREXIT IF WRTIF ERROR TO :ERR:
19 ENDIF
20 COMPUTE X AS (XCENT + XINCR + float(XCUR))
21 COMPUTE Y AS (YCENT + YINCR + float(YCUR))
22 ENDIF
23 ELSE
24 INCREMENT XCUR BY 1
25 IF THERE IS 1 VARIABLE AND X IS THE CENTROID OR
26 THERE ARE 2 VARIABLES AND X IS THE CENTROID AND Y IS THE CENTROID THEN
27 INCREMENT XCUR BY 1
28 CALL WRTIF TO WRITE Dummy AS CENTROID RECORD
29 ERREXIT IF WRTIF ERROR TO :ERR:
30 ENDIF
31 COMPUTE X AS (XCENT + XINCR + float(XCUR))
32 COMPUTE Y AS (YCENT + YINCR + float(YCUR))
33 ENDIF
34 ENDIF
35 DJ "OR # SCAN VARIABLES
36 CALL EXEC TO WRITE VARIABLE
37 SAVE REQUEST TO STORE VALUES FOR VARIABLE IN RERBUF
38 END
39 CALL EXEC TO WRITE $SCNTB
40 SAVE REQUEST TO STORE NEW $SCNTB
41 SET RETURN PARAMETERS TO RESET SEQUENCE NUMBER
42 END SETXY
SCAN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

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

INPUTS FROM MANAGER:

LU - LOGICAL UNIT OF THE USER'S TERMINAL
USEID - USER ID CODE
FLAGS - FLAGS FOR DEBUG

ENTSUB-DISPLACEMENT OF THIS SCAN ENTRY IN BSEG TB

INPUTS FROM INTERFACE TABLE:

PROCON-CARTRIDGE # FOR DATA BOX FILE
SUMTAB-SUMMARY TABLE
*DATBOX-NAME OF DATA BOX ENTERED
*NOVAR - NUMBER OF SCAN VARIABLES
*SCAN-NAME OF X SCAN VARIABLE
*XUNIT-UNIT OF X VARIABLE
*XCENTR-CENTROID OF X VARIABLE
XINCX-INCREMENT FOR X
XSTEPS-NUMBER OF STEPS FOR X
*TSCAN-NAME OF Y SCAN VARIABLE
YUNIT-UNIT OF Y VARIABLE
YCENTR-CENTROID OF Y VARIABLE
YINCY-INCREMENT FOR Y
YSTEPS-NUMBER OF STEPS FOR Y

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

OUTPUTS TO MANAGER

XZRET-RETURN CODE TO MANAGER

OUTPUTS TO WORK AREA (CWA):

DATBOX-FILE WHERE SUMMARY TABLE IS WRITTEN
XSCAN-X SCAN VARIABLE
TSCAN-Y SCAN VARIABLE
BSCNTB-SCAN CONTROL TABLE (173 WORDS PER ACTIVE SCAN)

(1) SUMTAB & CHAR NAME OF SUMMARY TABLE
(2) DISP DISPLACEMENT FOR SUMMARY TABLE
(5) DATBOX & CHAR QUALIFIED NAME OF DATA BOX FILE
(6) SET SIG - RESET SEQUENCE NUMBER
(7) CENTRE CENTROID RECORD NUMBER
(10) XSCAN NAME OF X VARIABLE
(13) XDISP - DISPLACEMENT FOR X
(14) XCE-T CENTROID FOR X
(16) XINCX - INCREMENT FOR X
789  1 CO3
790  1 CO3
791  1 CO3
792  1 CO3
793  1 CO3
794  1 CO3
795  1 CO3
796  1 CO3
797  1 CO3
798  1 CO3***********
799  1 CO5
800  1 CO5
801  1 CO5
802  1 CO5
803  1 CO5
804  1 CO5
805  1 CO5
806  1 CO5
807  1 CO5
808  1 CO5
809  1 CO5
810  1 CO5
811  1 CO5
812  1 CO***********

(18) XSTEP  # STEPS OF X
(19) XCUR  CURRENT X STEP NUMBER
(20) YSCAN NAME OF Y VARIABLE
(21) YDISPL DISPLACEMENT FOR Y
(24) YCENT CENTROID OF Y
(26) YINCR INCREMENT FOR Y
(27) YSTEP # STEPS OF Y
(29) YCUR CURRENT Y STEP NUMBER
(30) IDCB  144 WORD DCB FOR DATBOX

RTE SUBROUTINES USED:
CLOSE, CREAT, EXEC, FLOAT, IAND,
KCVT, KOD, POSN, PURGE, RMPAR,
WRITF

FDT SUBROUTINES USED:
XTPATR, XPGET, XPPUT, XPYIT, XRCPR,
XNEXT, XMOV, XRFM, XRUN, XDDBG,
873 2  ERREXIT IF NO AWN SPACE TO :ERR1:
874 2  CALL 4EXIT TO EXIT NORMALY
875 1  EXIT SCAN
876 2  ERR:
877 2  IF THERE ARE MORE ACTIVE SCANS (# SCANS > 0) THEN
878 3  CALL 4EC TO READ IN NEW BSCNTB
879 3  CALL EXEC TO WRITE OUT ORIGINAL BSCNTB
880 3  SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL BSCNTB
881 2  ENDIF
882 2  ERR2:
883 2  CALL CLOSE TO CLOSE DATBOX
884 2  CALL PURGE TO PURGE DATBOX
885 3  SET VALUE FOR XPPUT
886 3  SAVE REQUEST TO DELETE DATBOX FROM AWN
887 3  CALL XPPAW TO REQUEST AWN MANAGEMENT
888 2  ERR3:
889 2  CALL 4MSG TO DISPLAY ERROR
890 2  CALL 4EXIT TO ABEND SCAN
891 1  END SCAN
925 1 BEGIN XICHR
926 2 2  SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
928 2 2  :FIXERR: :FIXERR:
929 2 2  
930 2 2  
931 3 3  :CHRSTR:
932 3 3  DETERMINE # WORDS IN CHARACTER STRING
933 3 3  MOVE CHARACTER STRING TO RESULT LOCATION
934 4 4  IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
935 4 4  BLANK FILL AFTER CHARACTER STRING
936 3 3  ENDIF
937 3 3  :CHRFFRE:
938 3 3  IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
939 4 4  POP DISPLACEMENT FROM RESULT STACK
940 3 3  ELSE
941 4 4  SET DISPLACEMENT = 0
942 3 3  ENDIF
943 3 3  POP RESULT OPERAND FROM STACK
944 3 3  IF RESULT OPERAND IS FREE, THEN
945 4 4  SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
946 3 3  ELSE CHARACTER = CHARACTER
947 4 4  SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
948 3 3  ENDIF
949 3 3  CALL XPRET TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
950 4 4  IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
951 4 4  BLANK FILL REMAINING DATA
952 3 3  ENDIF
953 2 3  ENDCASE
954 1 EXIT XICHR
955 2  :FIXERR:
956 2 2  SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO"
957 2 2  "NUMERICAL DATA"
958 2 2  CALL XEGS TO OUTPUT MESSAGE TO USER
959 2 2  CALL XLSS TO LIST SYMBOLIC STRING
960 2 2  CALL XEXIT TO EXIT PROCESSOR
961 1 END XICHR
FORTRAN CALLING SEQUENCE:

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

XIDFT SEARCHES 'ARRAY' BEGINNING AT 'START' AND ENDING AT 'END' FOR A SPECIFIED 'TOKEN'. IT RETURNS EITHER THE POSITION OF THE 'TOKEN' OR THE 'END' VALUE IN 'INDEX'.

INPUTS FROM CALLING SEQUENCE:

<table>
<thead>
<tr>
<th>ARRAY</th>
<th>SYMMBOLIC STRING TO BE SEARCHED</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>ENDING INDEX INTO 'ARRAY'</td>
</tr>
<tr>
<td>START</td>
<td>BEGINNING INDEX INTO 'ARRAY'</td>
</tr>
<tr>
<td>TOKEN</td>
<td>TOKEN TO BE SEARCHED FOR</td>
</tr>
</tbody>
</table>

OUTPUTS TO CALLING SEQUENCE:

<table>
<thead>
<tr>
<th>INDEX</th>
<th>POSITION OF 'TOKEN' IN SYMBOLIC STRING IF FOUND, OR 'END' IF NOT FOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>Code</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1003</td>
<td>BEGIN</td>
</tr>
<tr>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td></td>
</tr>
<tr>
<td>1014</td>
<td></td>
</tr>
<tr>
<td>1015</td>
<td></td>
</tr>
<tr>
<td>1016</td>
<td></td>
</tr>
<tr>
<td>1017</td>
<td></td>
</tr>
<tr>
<td>1018</td>
<td></td>
</tr>
<tr>
<td>1019</td>
<td></td>
</tr>
<tr>
<td>1020</td>
<td></td>
</tr>
<tr>
<td>1021</td>
<td></td>
</tr>
<tr>
<td>1022</td>
<td></td>
</tr>
<tr>
<td>1023</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>1025</td>
<td></td>
</tr>
<tr>
<td>1026</td>
<td></td>
</tr>
<tr>
<td>1027</td>
<td></td>
</tr>
<tr>
<td>1028</td>
<td></td>
</tr>
<tr>
<td>1029</td>
<td></td>
</tr>
<tr>
<td>1030</td>
<td></td>
</tr>
<tr>
<td>1031</td>
<td></td>
</tr>
<tr>
<td>1032</td>
<td></td>
</tr>
</tbody>
</table>

5=351
XZDP1 - DATA BOX DISPLAY OVERLAY - PREPARES DATA FOR DISPLAY
- SCHEDULED BY DBDSP

XZDP1 READS IN THE SCAN CONTROL INFORMATION AND THE SUMMARY
TABLES PRODUCED BY SCAN/ENDSCAN FOR FURTHER PROCESSING.
IN ADDITION, XZDP1 VERIFIES THE DISPLAY DEPENDANT VARIABLE
NAMES AND THE CONSTRAINT VARIABLE NAMES ARE CONTAINED IN THE
DEPENDANT VARIABLE NAME LIST GENERATED DURING THE SCAN.
XZDP1 ALSO GENERATES CONSTRAINT MASKS WITH SUBROUTINE XZDMK
FOR SUBSEQUENT DISPLAY BY XZDP2 AND XZDOT.

INPUTS FROM THE DATA BOX

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

RECORD 2
(11) - NO OF ENTRIES IN SUMMARY TABLE
(2) - X SCAN VARIABLE (6 CHAR)
(5) - X FIRST SUBSCRIPT (INT OR ZERO)
(6) - XSECOND SUBSCRIPT (INT OR ZERO)
(7) - X UNITS (6 CHAR)
(10) - X CENTROID (REAL)
(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)
(25) - YX INCREMENT (REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

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

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
output from XZDP1

mask tables containing the constraint masks for each
constraint which was violated and indication of
whether or not any constraint was violated for each
array coordinate

list of valid display dependent 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

mask1 - array containing constraints a thru d
mask2 - array containing constraints e thru g

name list for variables scanned by scan/endscn
number of constraints input by user (integer)
list of constraint relations input by user
list of constraint variable names input by user (32 max)
number of dep disp variable pairs for paged output (1-16pr)
number of dep disp var in mvar list (integer)
list of dep disp variable pair calc factors for each set of indicators for constraints violated or not
number of steps on either side of x 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)
list of x var values for y coordinates (1 - 11 real)
name of y var scanned to be placed on display (6 char)
name of y var units to be placed on display (6 char)
units list for variables scanned by scan/endscn
values for scan variables(s) - 1 to 32 values/record
communication buffer for rman - lu, user id, flag
logical unit # for xprom calling sequence - user locatn
ludsp - dbdsp will output display to this user supplied lu
prompt - table in common to communicate with xprom
select =0 prompt; select not 0 run all displays to o/p
XZDP2 函数进行的操作和描述

XZDP2 程序提示用户输入页面号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。

XZDP2 函数调用 DBDSP 时可以设置的行号。

1. XZDP2 函数调用 DBDSP 时可以设置的行号。
2. XZDP2 函数调用 DBDSP 时可以设置的行号。
3. XZDP2 函数调用 DBDSP 时可以设置的行号。
4. XZDP2 函数调用 DBDSP 时可以设置的行号。
5. XZDP2 函数调用 DBDSP 时可以设置的行号。
6. XZDP2 函数调用 DBDSP 时可以设置的行号。
7. XZDP2 函数调用 DBDSP 时可以设置的行号。
8. XZDP2 函数调用 DBDSP 时可以设置的行号。
9. XZDP2 函数调用 DBDSP 时可以设置的行号。
10. XZDP2 函数调用 DBDSP 时可以设置的行号。
1461 1 CD 4 NDVARL - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16 PR) XZDP2
1462 1 CD 4 NDVRL - NUMBER OF DEP DISP VAR IN NDVARL LIST (INTEGER) XZDP2
1463 1 CD 4 NDVRL - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P XZDP2
1464 1 CD 4 MBSKRE - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=O/NOT 0 XZDP2
1465 1 CD 4 NWSPM - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5) XZDP2
1466 1 CD 4 NWTMP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5) XZDP2
1467 1 CD 4 RETC - RTN CODE FM XPRDM; 0=NRML, 1=RTN TO EXEC, 2=NULL BFR, 3=ERR XZDP2
1468 1 CD 4 STRING - CHAR STRING CONTAINING USER PROMPT MESSAGE XZDP2
1469 1 CD 4 TORDER - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL) XZDP2
1470 1 CD 4 XSCMMN - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR) XZDP2
1471 1 CD 4 XUNIT - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR) XZDP2
1472 1 CD 4 XSCMMN - NAME OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL) XZDP2
1473 1 CD 4 YSCMMN - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR) XZDP2
1474 1 CD 4 YUNIT - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR) XZDP2
1475 1 CD 4 TSAFE - TABLE IN COMMON FOR SUNITB VARIABLE NAMES AND UNITS XZDP2
1476 1 CD 4 NVAR - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN XZDP2
1477 1 CD 4 SUNITB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD XZDP2
1478 1 CD 4 PARS - COMMUNICATION BUFFER FOR WPAR - LU, USER ID, FLAGS XZDP2
1479 1 CD 4 LU - LOGICAL UNIT # FOR XPRDM CALLING SEQUENCE - USER LOCATIN XZDP2
1480 1 CD 4 LDSP - DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU XZDP2
1481 1 CD 4 FSELECT - SELECT = 0 PRIORITY; SELECT NOT 0 RUN ALL DISPLAYS TO O/P XZDP2
1482 1 CD 4 CARTRG - CARTRIDGE USED TO LOCATE DATA BOX XZDP2
1483 1 CD 4 * * * * * Pchanges USES ROUTINES XZDP2
1484 1 CD 4 CARTGR - CARTRIDGE USED TO LOCATE DATA BOX XZDP2
1485 1 CD 4 * * * * * XPRDM, EXEC, XZDOT XZDP2
1486 1 CD 5 1 CD 5 1 CD 5 1 CD 5 1 CD 5anches
**REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR**

---

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>COMPO -- WHERE</th>
<th>UMN -- DEVL</th>
<th>TEMPL</th>
<th>XDEFU</th>
<th>TEMPL</th>
<th>XDEFU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**XZDOT -- DISPLAY OUTPUT ROUTINE**

**XZDOT IS CALLED ONCE BY XZSP2 TO OUTPUT THE DISPLAY PAGE**

---

**USES ROUTINES**

<table>
<thead>
<tr>
<th>EXEC</th>
<th>XZSP</th>
<th>XZSP</th>
<th>XZSP</th>
<th>XZSP</th>
<th>XZSP</th>
<th>XZSP</th>
<th>XZSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15531</th>
<th>15532</th>
<th>15533</th>
<th>15534</th>
<th>15535</th>
<th>15536</th>
<th>15537</th>
<th>15538</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**INPUTS**

<table>
<thead>
<tr>
<th>15541</th>
<th>15542</th>
<th>15543</th>
<th>15544</th>
<th>15545</th>
<th>15546</th>
<th>15547</th>
<th>15548</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**OUTPUTS**

<table>
<thead>
<tr>
<th>15551</th>
<th>15552</th>
<th>15553</th>
<th>15554</th>
<th>15555</th>
<th>15556</th>
<th>15557</th>
<th>15558</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15561</th>
<th>15562</th>
<th>15563</th>
<th>15564</th>
<th>15565</th>
<th>15566</th>
<th>15567</th>
<th>15568</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15571</th>
<th>15572</th>
<th>15573</th>
<th>15574</th>
<th>15575</th>
<th>15576</th>
<th>15577</th>
<th>15578</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15581</th>
<th>15582</th>
<th>15583</th>
<th>15584</th>
<th>15585</th>
<th>15586</th>
<th>15587</th>
<th>15588</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15591</th>
<th>15592</th>
<th>15593</th>
<th>15594</th>
<th>15595</th>
<th>15596</th>
<th>15597</th>
<th>15598</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15601</th>
<th>15602</th>
<th>15603</th>
<th>15604</th>
<th>15605</th>
<th>15606</th>
<th>15607</th>
<th>15608</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15611</th>
<th>15612</th>
<th>15613</th>
<th>15614</th>
<th>15615</th>
<th>15616</th>
<th>15617</th>
<th>15618</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15621</th>
<th>15622</th>
<th>15623</th>
<th>15624</th>
<th>15625</th>
<th>15626</th>
<th>15627</th>
<th>15628</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15631</th>
<th>15632</th>
<th>15633</th>
<th>15634</th>
<th>15635</th>
<th>15636</th>
<th>15637</th>
<th>15638</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15641</th>
<th>15642</th>
<th>15643</th>
<th>15644</th>
<th>15645</th>
<th>15646</th>
<th>15647</th>
<th>15648</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15651</th>
<th>15652</th>
<th>15653</th>
<th>15654</th>
<th>15655</th>
<th>15656</th>
<th>15657</th>
<th>15658</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**NOTES**

<table>
<thead>
<tr>
<th>15661</th>
<th>15662</th>
<th>15663</th>
<th>15664</th>
<th>15665</th>
<th>15666</th>
<th>15667</th>
<th>15668</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 CD******
2 CD0
3 CD0 FORTRAN CALLING PROCEDURE
4 CD0 CALL XZ1SP (STRING, LEN)
5 CD0
6 CD******
7 CD1 XZ1SP REMOVES DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
8 CD1 A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
9 CD1 WITH BLANKS
10
11 CD******
12 CD2 INPUT
13 CD2 CALLING SEQUENCE
14 CD2 STRING - INPUT CHARACTER STRING
15 CD2 LEN - NUMBER OF WORDS IN STRING
16
17 CD******
18 CD3 OUTPUT
19 CD3 CALLING SEQUENCE
20 CD3 STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
21 CD3 BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILLED
22 CD3 LEN - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
23
24 CD******
25 CD4 NOTES
26 CD4 USES ROUTINES
27 CD4 XR1SP
28
29 CD******
30 BEGIN XZ1SP
31 2 CALL XR1SP TO REMOVE DUPLICATE BLANKS FROM STRING
32 2 DC WHILE THERE ARE TRAILING WORDS IN STRING
33 3 SET THIS TRAILING WORD TO BLANKS
34 2 ENDXR1SP
35 1 END XZ1SP
FORTRAN CALLING SEQUENCE:

CALL XIFCL (LU)

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

INPUTS FROM CALLING SEQUENCE:

LU - LOGICAL UNIT OF TERMINAL BEING USED.

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

BEGIN XIFCL

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

CALL XIFNC(ENTRY)

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

INPUTS

ENTRY - FUNCTION TOKEN CURRENTLY BEING PROCESSED

FROM ASSGN - LU, SSTRNG, OPRNDS, OPIINFO

OUTPUTS TO ASSGN

RESULT, RSLTP, OPRNDS

EXTERNAL REFERENCES

RTE - ABS, AINT, ALOG, ALGOT, AMOD, ATAN, ATAN2, COS, DABS, DATAN,

DATN2, DBLE, DINT, DEEP, DLOG, DLGOT, DMOD, DS1-W, DSIN, DSQRT,

EXP, FLOAT, IABS, IDINT, IFIX, ISIGN, MOD, OVF, SIGN, SIM, SNGLE, SGRT,

TAN, TANH
1711 1 BEGIN XIFNC
1712 2 PERFORM FUNCTION INDICATED BY ENTRY
1713 2 EREXIT TO :OVER: IF OVERFLOW OR UNDERFLOW IS INDICATED
1714 2 PUSH RESULT AND TYPE ONTO RESULT STACK
1715 1 EXIT XIFNC
1716 2 :OVER:
1717 2 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
1718 2 CALL XIMSG TO OUTPUT MESSAGE TO USER
1719 2 CALL XLIST TO LIST SYMBOLIC STRING
1720 2 CALL XEXIT TO EXIT PROCESSOR
1721 1 END XIFNC
<table>
<thead>
<tr>
<th>_lineno</th>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1723</td>
<td>1 CDO</td>
<td>FORTRAN CALLING PROCEDURE:</td>
</tr>
<tr>
<td>1724</td>
<td>1 CDO</td>
<td>CALL XIFRE</td>
</tr>
<tr>
<td>1725</td>
<td>1 CDO</td>
<td>XIFRE IS USED BY THE ASSGN ROUTINE XIPS2 TO PROCESS DATA ASSIGNMENTS</td>
</tr>
<tr>
<td>1726</td>
<td>1 CDO</td>
<td>FOR FREE-TYPE OBJECT DATA ELEMENTS</td>
</tr>
<tr>
<td>1727</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1728</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1729</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1730</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1731</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1732</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1733</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1734</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1735</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1736</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1737</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1738</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1739</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1740</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1741</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1742</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1743</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1744</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1745</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1746</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1747</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1748</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1749</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1750</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1751</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1752</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
<tr>
<td>1753</td>
<td>1 CDO</td>
<td>bole</td>
</tr>
</tbody>
</table>
**REPRODUCIBILITY OF THE ORIGINAL PAGE IS poor**

```
1755  1 BEGIN XZFR
1756  2 * CASE (RESULT DATA TYPE ) :FRESTR:, :FREFR:, :FREFR:, :FREFX:, :
1757  3 * :FREXFR:, :FREFF:, :
1758  3 * :
1759  3 * :
1760  3 * :FRESTR:
1761  3 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1762  3 MOVE CHARACTER STRING TO RESULT LOCATION
1763  3 :FREFR:
1764  3 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1765  4 POP DISPLACEMENT FROM RESULT STACK
1766  4 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPTED
1767  4 SET DISPLACEMENT = 0
1768  3 ENDIF
1769  3 :FREFF:
1770  3 POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1771  3 GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
1772  4 CALL XPRCQ TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1773  4 SET # WORDS TO BE STORED IN OBJECT = 1
1774  4 ELSE FREE = CHARACTER DATA ELEMENT
1775  4 CALL XPRQ To RETRIEVE LOGICAL ELEMENT OF CHARACTER DATA FROM RESULT OPERAND
1776  4 AT DISPLACEMENT DETERMINED
1777  4 SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR RESULT OPERAND
1778  3 ENDIF
1779  3 :FREFX:
1780  3 CALL XZPCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781  3 SET # WORDS TO BE STORED = RESULT DATA TYPE
1782  2 END CASE
1783  1 END XZFR
```
FORTRAN CALLING PROCEDURE:

CALL XZFD

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

INPUTS FROM ASGCOM
LU,SYMTAB,STSTRNG,RESULT,RSLTPL,DATYPS,CLSTRN

OUTPUTS TO ASGCOM
RSLTPL,REQST,NUMWDS,OPRND5

FDS - XPREG,XPXIT,XRMOV,XZLSS,XXMSG,XZPCS
RTE - IAND

"0
1815 1 BEGIN XFDF
1816 2 SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1817 3 CASE (RESULT DATA TYPE ) :
1818 4 NOCHAR: 1 2
1819 5 FIXRE: 1 2
1820 6 FIXFX: 1 2
1821 7 FIXRE:
1822 8 ERRExit TO NOCHAR: IF TYPE IN SYMBOL TABLE FOR RESULT OPERAND IS NOT FREE
1823 9 IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1824 10 POP DISPLACEMENT FROM RESULT STACK
1825 11 ELSE FREE ELEMENT HAS NOT BEEN SUBSCRIPTED
1826 12 SET DISPLACEMENT = 0
1827 13 ENDF
1828 14 POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1829 15 CALL XPCX TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1830 16 (# WORDS RETRIEVED = OBJECT DATA TYPE)
1831 17 FIXFX:
1832 18 SET TARGET TYPE TO OBJECT DATA TYPE
1833 19 CALL XPCX TO POP RESULT VALUE, CONVERT IF NECESSARY, AND SET UP FOR STORE
1834 20 ENDCASE
1835 1 EXIT XFDF
1836 2 NOCHAR:
1837 3 SET MESSAGE TO BE OUTPUT TO "NUMERICAL DATA ELEMENT CANNOT BE SET EQUAL TO
1838 4 CHARACTER DATA"
1839 5 CALL XIMSG TO OUTPUT MESSAGE TO USER
1840 6 CALL XLIST TO LIST SYMBOLIC STRING
1841 7 CALL XPXR TO EXIT PROCESSOR
1842 1 END XFDF
FORTRAN CALLING PROCEDURE
CALL XILSS (LU, STRING, INDEX)
XILSS IS CALLED TO LIST A SYMBOLIC STRING AND AN INDICATOR TO A
PARTICULAR TOKEN IN THAT STRING

INPUT
LU - LOGICAL UNIT NO. FOR OUTPUT OF STRING
STRING - SYMBOLIC STRING TO BE LISTED
INDEX - SUBSCRIPT INTO STRING OF THE TOKEN TO BE INDICATED

OUTPUT
THE SYMBOLIC STRING IS OUTPUT TO THE LU FOLLOWED BY A LINE CONTAINING
AN INDICATOR (UP ARROW) TO THE DESIGNATED TOKEN.
<table>
<thead>
<tr>
<th>Line</th>
<th>COA</th>
<th>Internal Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1871</td>
<td>CD4</td>
<td>INTERNAL VARIABLES</td>
</tr>
<tr>
<td>1872</td>
<td>CD4</td>
<td></td>
</tr>
<tr>
<td>1873</td>
<td>CD4</td>
<td>CONTR = CONTROL TABLE DESCRIBING DISPOSITION AND PROCESSING</td>
</tr>
<tr>
<td>1874</td>
<td>CD4</td>
<td>FOR EACH OF THE TOKEN VALUES 1-32. EACH CONTROL TABLE</td>
</tr>
<tr>
<td>1875</td>
<td>CD4</td>
<td>ENTRY IS 3 WORDS:</td>
</tr>
<tr>
<td>1876</td>
<td>CD4</td>
<td></td>
</tr>
<tr>
<td>1877</td>
<td>CD4</td>
<td>WORD 1 (SIZE) = NO. OF WORDS IN PRINT BUFFER</td>
</tr>
<tr>
<td>1878</td>
<td>CD4</td>
<td>WORD 2 (FIELD) = CONTENTS TO GO INTO PRINT BUFFER</td>
</tr>
<tr>
<td>1879</td>
<td>CD4</td>
<td>OR FLAG DESCRIBING HOW TO COMPUTE</td>
</tr>
<tr>
<td>1880</td>
<td>CD4</td>
<td>THEN</td>
</tr>
<tr>
<td>1881</td>
<td>CD4</td>
<td></td>
</tr>
<tr>
<td>1882</td>
<td>CD4</td>
<td>WORD 3 (TOKSZ) = NO. OF WORDS IN SYMBOIC STRINGS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>COA</th>
<th>Entry</th>
<th>Token</th>
<th>Word 1 (Size)</th>
<th>Word 2 (Field)</th>
<th>Word 3 (TOKSZ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1884</td>
<td>CD4</td>
<td>1</td>
<td>INTEGER</td>
<td>3</td>
<td>CALL XRI6</td>
<td>2</td>
</tr>
<tr>
<td>1885</td>
<td>CD4</td>
<td>2</td>
<td>REAL</td>
<td>7</td>
<td>CALL XRI14</td>
<td>3</td>
</tr>
<tr>
<td>1886</td>
<td>CD4</td>
<td>3</td>
<td>DOUBLE</td>
<td>9</td>
<td>CALL XRI15</td>
<td>4</td>
</tr>
<tr>
<td>1887</td>
<td>CD4</td>
<td>4</td>
<td>NAME</td>
<td>3</td>
<td>USE 3 WORDS</td>
<td>4</td>
</tr>
<tr>
<td>1888</td>
<td>CD4</td>
<td>5</td>
<td>FOLLOWING TOKEN</td>
<td>0</td>
<td>ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>1889</td>
<td>CD4</td>
<td>6</td>
<td>CHARACTER STR. +1 USE VALUE</td>
<td>5 USE SIZE WORDS</td>
<td>9 USE SIZE+2 WORDS</td>
<td></td>
</tr>
<tr>
<td>1890</td>
<td>CD4</td>
<td>7</td>
<td>WORD FOLLOWING BEGINNING 2 PAST</td>
<td>0</td>
<td>ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>1891</td>
<td>CD4</td>
<td>8</td>
<td>TOKEN</td>
<td>0</td>
<td>ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>1892</td>
<td>CD4</td>
<td>9</td>
<td>+</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1893</td>
<td>CD4</td>
<td>10</td>
<td>-</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1894</td>
<td>CD4</td>
<td>11</td>
<td>*</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1895</td>
<td>CD4</td>
<td>12</td>
<td>/</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1896</td>
<td>CD4</td>
<td>13</td>
<td>&lt;</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1897</td>
<td>CD4</td>
<td>14</td>
<td>&gt;</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1898</td>
<td>CD4</td>
<td>15</td>
<td>#</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1899</td>
<td>CD4</td>
<td>16</td>
<td>?</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1900</td>
<td>CD4</td>
<td>17</td>
<td>@</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1901</td>
<td>CD4</td>
<td>18</td>
<td>!</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1902</td>
<td>CD4</td>
<td>19</td>
<td>$</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1903</td>
<td>CD4</td>
<td>20</td>
<td>%</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1904</td>
<td>CD4</td>
<td>21</td>
<td>(</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1905</td>
<td>CD4</td>
<td>22</td>
<td>)</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1906</td>
<td>CD4</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1907</td>
<td>CD4</td>
<td>24</td>
<td>END SYM. STR.</td>
<td>0</td>
<td>END SYM. STR.</td>
<td>0</td>
</tr>
<tr>
<td>1908</td>
<td>CD4</td>
<td>25</td>
<td>X</td>
<td>1</td>
<td>BACKSLASH</td>
<td>1</td>
</tr>
<tr>
<td>1909</td>
<td>CD4</td>
<td>26</td>
<td>BACKSLASH</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1910</td>
<td>CD4</td>
<td>27</td>
<td>#</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1911</td>
<td>CD4</td>
<td>28</td>
<td>LEFT BRACKET</td>
<td>1</td>
<td>LEFT BRACKET</td>
<td>1</td>
</tr>
<tr>
<td>1912</td>
<td>CD4</td>
<td>29</td>
<td>RIGHT BRACKET</td>
<td>1</td>
<td>RIGHT BRACKET</td>
<td>1</td>
</tr>
<tr>
<td>1913</td>
<td>CD4</td>
<td>30</td>
<td>4</td>
<td>1</td>
<td>CALL XRI6 AND APPEND &quot;^&quot;</td>
<td>2</td>
</tr>
<tr>
<td>1914</td>
<td>CD4</td>
<td>31</td>
<td>REPEAT</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1915</td>
<td>CD4</td>
<td>32</td>
<td>,</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
1 C********
2 |
3 | NOTES
4 | 1 1
5 | 1 1
6 | 1 1
7 | 1 1
8 | 1 1
9 | 1 1
10 | 1 1
11 | 1 1
12 | 1 1
13 | 1 1
14 | 1 1
15 | 1 1
16 | 1 1
17 | 1 1
18 | 1 1
19 | 1 1
20 | 1 1
21 | 1 1
22 | 1 1
23 | 1 1
24 | 1 1
25 | 1 1
26 | 1 1
27 | 1 1
28 | 1 1
29 | 1 1
30 | 1 1
31 | 1 1
32 | 1 1
33 | 1 1
34 | 1 1
35 | 1 1
36 | 1 1
37 | 1 1
38 | 1 1
39 | 1 1
40 | 1 1
41 | 1 1
42 | 1 1
43 | 1 1
44 | 1 1
45 | 1 1
46 | 1 1
47 | 1 1
48 | 1 1
49 | 1 1
50 | 1 1
51 | 1 1
52 | 1 1
53 | 1 1
54 | 1 1
55 | 1 1
56 | 1 1
57 | 1 1
58 | 1 1
59 | 1 1
60 | 1 1
61 | 1 1
62 | 1 1
63 | 1 1
64 | 1 1
65 | 1 1
66 | 1 1
67 | 1 1
68 | 1 1
69 | 1 1
70 | 1 1
71 | 1 1
72 | 1 1
73 | 1 1
74 | 1 1
75 | 1 1

1924 1 C*********
1925 1 C05
1926 1 C05
1927 1 C05
1928 1 C05
1929 1 C05
1930 1 C05
1931 1 C05
1932 1 C05
1933 1 C05
1934 1 C05
1935 1 C05
1936 1 C05
1937 1 C*********
1938 2 1 BEGIN XILSS
1939 2 2 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
1940 2 2 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
1941 2 2 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
1942 2 2 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
1943 2 2 EXIT TO ERROR 2 IF FIELD = 0
1944 3 3 IF SIZE < 0, THEN
1945 4 4 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
1946 3 4 ENDIF
1947 3 4 IF TOKSIZ < 0, THEN
1948 4 4 SET TOKSIZ TO SIZE + 2
1949 3 4 ENDIF
1950 3 4 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
1951 4 4 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
1952 4 4 SET BUFFER POINTER TO 1ST POSITION FOR DATA
1953 4 4 IF INDICATED TOKEN PROCESSED, THEN
1954 5 5 OUTPUT LINE WITH INDICATOR
1955 4 4 ENDIF
1956 3 4 ENDIF
1957 3 4 IF FIELD > 0, THEN
1958 4 4 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
1959 3 4 ELSE
1960 4 4 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
1961 5 5 :ONE: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING
1962 5 5 AND PUT RESULTS INTO PRINT BUFFER
1963 5 5 :TWO: CALL XRI14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
1964 5 5 AND PUT RESULTS INTO PRINT BUFFER
1965 5 5 :THREE: CALL XRI18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
1966 5 5 AND PUT RESULTS INTO PRINT BUFFER
1967 5 5 :FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
1968 5 5 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
1969 5 5 :SIX: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
1970 5 5 INTO PRINT BUFFER FOLLOWED BY AN "R"
1971 5 5 :EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
1972 5 5 INDEX BY 1
1973 5 5 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
1974 5 5 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
1975 6 6 INDICATOR LINE NOT YET OUTPUT, THEN
**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, SYNTAB, DATYPS, RSLTP, CLSTM, MAPHDS, RESULT**

**OUTPUTS TO ASSGN**

**RESULT, OPANDS, RESST, RSLTP**

**INTERNAL VARIABLES**

**MAPOP - MAPS OPERATOR TOKENS FOR EXECUTION**

**EXTERNAL REFERENCES**

**FDS - XPREN, XPXIT, XRMOV, XILSS, XIMSG**

**RTE - IAMC, OUF**
BEGIN XOPR

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

:EXPO:, :UMINUS:

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

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

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

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

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

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

:EXPO:
PERFORM EXPONENTIALIZATION
SET RESULT'S DATA TYPE AS APPROPRIATE

:UMINUS:
CHANGE SIGN OF OPERAND FOR RESULT
ENDCASE

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

SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
BEGIN XPCS
  POP OPERAND AND DATA TYPE FROM TOP ENTRY OF RESULT STACK
  IF TARGET TYPE AND DATA TYPE ARE NOT EQUAL, THEN
  CASE (TARGET TYPE) :INTS: :REAL: :DBLE:
  :INTS:
  CONVERT OPERAND TO INTEGER
  :REAL:
  CONVERT OPERAND TO SINGLE PRECISION REAL
  :DBLE:
  CONVERT OPERAND TO DOUBLE PRECISION REAL
  ENDCASE
  ENDF
  IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
  STORE OPERAND AND CURRENT DATA TYPE FOR USE BY FUNCTION OR ARITHMETIC OPER.
  EXIT XPCS
  SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
  CALL XMSG TO OUTPUT MESSAC" TO USER
  CALL XLESS TO LIST SYMBOLIC STRING
  CALL XPFIT TO EXIT PROCESSER
END XPCS
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

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

PRIORITY

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

TKNPTR - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING

TOKEN - TOKEN CURRENTLY BEING PROCESSED

EXTERNAL REFERENCES

RTE - IAMD, MINO

FDX - XPSII, XLSS, XMSG, XSYM
BEGIN SETUP

CASE TOKEN

IF TOKEN IS A NAME AND NEXT TOKEN IS A LEFT BRKT, THEN
START SEARCH UNTIL FUNCTION TABLE IS CHECKED
EXIT IF TOKEN NAME MATCHES FUNCTION NAME
CHANGE TOKEN TO FUNCTION'S INDEX IN TABLE + 128
PUSH COUNT: BINARY/UNARY FLAG AND LIMIT=0 ONTO GROUPING STACK
END LOOP
END SEARCH
ELSE TOKEN IS AN OPERAND
IF TOKEN IS A NAME AND NEXT TOKEN IS (, THEN
GET SUBSCRIPT FLAG
CALL XSYM TO STORE SYMBOL AND SUBSCRIPT FLAG
PUSH SYMBOL INDEX ONTO EXPRESSION STACK
ENDIF

:CHAR:
PUSH NEGATIVE POINTER TO CHARACTER STRING ONTO EXPRESSION STACK

:PLUS:
IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
INC FERN TO NEXT TOKEN AND EXIT TO :ENDO:
ENDIF

:MINUS:
IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
CHANGE TOKEN TO UNARY MINUS
ENDIF

:ASTER:
IF NEXT TOKEN IS *, THEN
CHANGE TOKEN TO EXPONENTIATION
ENDIF

:LPAR:
IF PRECEDING TOKEN WAS A VARIABLE (DATA ELEMENT NAME), THEN
PUSH COUNT AND LIMIT ALLOWING DOUBLE SUBSCRIPTING ONTO GROUPING STACK
ELSE EXPRESSION IS BEING PROCESSED
PUSH COUNT AND LIMIT ALLOWING NO SUBSCRIPTING ONTO GROUPING STACK
ENDIF

ENDCASE
2325 2 ENOCAS:
2326 1 EXIT SETUP
2327 2 :INVLD:
2328 2 SET MESSAGE TO BE OUTPUT TO "INVALID CHARACTER"
2329 2 :BADFUN:
2330 2 SET MESSAGE TO BE OUTPUT TO "FUNCTION NOT SUPPORTED BY THIS PROCESSOR"
2331 2 CALL ZXMSG TO OUTPUT MESSAGE TO USER
2332 2 CALL ZXLSG TO LIST SYMBOLIC STRING
2333 2 CALL XPXIT TO EXIT PROCESSOR
2334 1 END SETUP
BEGIN STRING  
SET INDEX INTO SYNTAX TABLE TO MIN(TOKEN, 40)  
ERREXIT TO :SYNTX1: IF THIS TOKEN IS NOT VALID ACCORDING TO SYNTAX TABLE  
IF TOKEN IS NOT AN OPERAND, THEN  
DO UNTIL TOKEN IS PUSHED ONTO OPERATOR STACK OR DISCARDED  
IF INPUT PRIORITY OF THIS TOKEN > OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR  
STACK, THEN  
IF TOKEN IS , THEN  
INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK  
ERREXIT TO :SYNTX1: IF COUNT > COMMA LIMIT FOR (TOP ENTRY IN GROUPING STK  
IF COMMA LIMIT SHOWS SUBSCRIPTING 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 :SYNTX1: IF INPUT PRIORITY = 0 AND OPERATOR STACK IS EMPTY  
IF TOP ENTRY OF OPERATOR STACK IS =, THEN  
ERREXIT IF THIS IS NOT LAST ENTRY ON OPERATOR STACK TO :SYNTX3:  
ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTX2:  
DISCARD TOKEN (; OR ESS)  
ENDIF  
POP OPERATOR STACK  
PUSH OPERATOR ONTO EXPRESSION STACK  
ELSE BRACKETS OR PARENTHESES HAVE BEEN MATCHED  
IF TOP ENTRY OF OPERATOR STACK IS (, THEN  
ERREXIT TO :SYNTX2: IF CURRENT TOKEN IS NOT )  
IF TOP ENTRY OF GROUPING STACK INDICATES SUBSCRIPTING (LIMIT > 0), THEN  
PUSH SUBSCRIPTING OPERATOR ONTO EXPRESSION ARRAY  
ENDIF  
ELSE TOP OPERATOR ENTRY IS OPEN BRACKET (FUNCTION LIST)  
ERREXIT IF CURRENT TOKEN IS NOT CLOSE BRACKET TO :SYNTX2:  
ERREXIT TO :SYNTX3: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GROUPING STK COUNT = 0)  
ENDIF  
POP OPERATOR STACK  
POP GROUPING STACK  
DISCARD CURRENT TOKEN  
ENDIF  
ENDIF  
END DO  
END:ENDIF  
INCREMENT TO NEXT TOKEN USING TOKEN LENGTH FROM SYNTAX TABLE  
EXIT STRING  
:SYNTX1:  
SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"  
:SYNTX2:  
SET MESSAGE TO BE OUTPUT TO "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY"  
:COMERR:  
SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
BEGIN RANGE
  INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
  IF LAST TOKEN INPUT HAS ; THEN
  DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
  ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
  CALL XSYM TO INSERT NAME IN SYMBOL TABLE, IF NECESSARY, FLAG ENTRY AS AN INDEX, AND RETURN SYMBOL TABLE INDEX
  ERREXIT IF NEXT TOKENS ARE NOT "INTEGER,INTEGER" TO :RNGSYM:
  IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
  SET INCREMENT TC -1
  EMDIF
  PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
  EMDIF DO
  ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
  EMDIF
  EXIT RANGE
  :RNGSYM:
  SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ";NAME=
  * INTEGER,INTEGER"
  CALL XMSG TO OUTPUT MESSAGE TO USER
  CALL XLIST TO LIST SYMBOLIC STRING
  CALL XPRT TO EXIT PROCESSOR
  END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

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

INPUTS FROM ASGCOM

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

OPINFO, NUMNDS, DATTPS, RSLTPT, EXPRX, CLSREQ, CLSTRM, DECLAS,

RNGSTR, POLISH, SYNTAX, FMCTBL

OUTPUTS TO ASGCOM

SYMTAB, RESULT, REQT, RSLTPT, OPINFO

INTERNAL VARIABLES

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

EXTERNAL REFERENCES

FD5 - XPRQ, XPYT, XRMOV, XZCNR, XZFCM, XZFre, XZFD, XZLSS, XZMSG,

XZOPR, XZPDS, XZRET

RTE - IABS, JANQ, MAXB

CD62 1 CD5

CD63 1 CD0

CD60 1 CD5
2465 1 BEGIN XIPS2
2466 2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
2467 2 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
2468 2 INITIALIZE RANGE VALUES FOR ITERATION
2469 3 DO UNTIL ALL RANGES ARE FINISHED
2470 3 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
2471 4 POP ENTRY FROM POLISH STRING
2472 4 IF ENTRY IS AN OPERAND, THEN
2473 4 PERFORM PUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH ONTO RESULT STACK
2474 4 ELSE ENTRY IS AN OPERATOR
2475 5 IF OPERATOR IS NOT =, THEN
2476 6 PERFORM EXEVAL TO EVALUATE POLISH STRING
2477 5 ELSE OPERATOR IS =
2478 6 PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
2479 5 ENDEF
2480 4 ENDF
2481 3 END DO
2482 3 PERFORM RNGSET TO DETERMINE CORRECT RANGE VALUES
2483 2 END DO
2484 1 END XIPS2
2486 1 BEGIN TOC
2487 2 DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
2488 3 IF SYMBOL NOT = 0 (SYMBOL IS DE OR INDEX), THEN
2489 4 IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN
2490 5 CALL XPREQ TO RETRIEVE TOC ENTRY
2491 6 ELSE SYMBOL IS A RANGE INDEX
2492 7 SET DATA TYPE TO INTEGER
2493 8 ENDIF
2494 3 ENDIF
2495 2 END DO
2496 2 CALL XPREQ WITH A CLOSE BUFFER REQUEST
2497 1 END TOC
BEGIN DATA
DO UNTIL ALL ENTRIES IN SCROLL TABLE ARE PROCESSED
IF A VARIABLE-LENGTH FIXED-LENGTH DATA ELEMENT THEN
CALL XBAR A FUNCTION REQUEST FOR DATA RETRIEVAL
END IF
CALL XBAR WITH A CLOSE BUFFER REQUEST
END DATA
1 BEGIN RPUSH
2 IF Operand IS A CHARACTER STRING POINTER (operand < 0), THEN
3 PUSH ABSOLUTE VALUE OF OPERAND AND #3 DATA TYPE ONTO RESULT STACK
4 ELSE OPERAND IS A SYMBOL TABLE INDEX
5 CALL SIZE TO RETRIEVE DATA AND DATA TYPE FOR OPERAND
6 PUSH RETURNED VALUE AND DATA TYPE ONTO RESULT STACK
7 ENDIF
8 END RPUSH
1 BEGIN EVAL
2 IF OPERATOR IS A FUNCTION, THEN
3 DETERMINE & OPERANDS FROM FUNCTION TABLE
4 ELSE
5 DETERMINE & OPERANDS FROM SYNTAX TABLE
6 ENDIF
7 DO FOR EACH OPERAND
8 IF RESULT STACK CONTAINS A SYMBOL INDEX, THEN
9 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
10 ENDIF
11 CALL ZIPS TO RETRIEVE DATA AND "PE FOR "T" ELEMENT OF ARRAY
12 STORE DATA AND TYPE IN RESULT STACK AT THIS ENTRY
13 ELSE
14 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
15 ENDIF
16 END DO
17 IF THE OPERAND IS A FUNCTION, THEN
18 DO FOR EACH FUNCTION OPERAND
19 SET TARGET TYPE FROM FUNCTION TABLE
20 CALL ZIPS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
21 END DO
22 CALL ZIFMC TO PERFORM FUNCTION FOR RESULTS
23 SET DATA TYPE FOR RESULTS FROM FUNCTION TABLE
24 ELSE
25 THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "="
26 DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
27 CASE (TYPE REQUIREMENT+1) :
28 SAME:
29 IF OPERANDS > 1, THEN
30 SET TARGET TYPE TO MAX OF TWO DATA TYPES
31 ELSE IF OPERANDS = 1
32 SET TARGET TYPE TO OPERAND'S DATA TYPE
33 ENDSF
34 :INTEGER:
35 SET TARGET TYPE TO INTEGER
36 ENDCASE
37 DO FOR EACH OPERAND
38 CALL ZIPS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
39 END DO
40 CALL ZIOPR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
41 ENDIF
42 EXIT EVAL
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
BEGIN REPLAC
IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
ENDIF
CASE (OBJECT DATA TYPE ) :
FREE; :FIXED; :FIXED; :CHAR; :CHAR;
FREE;
:CHAR; :CHAR; :CHAR;
CASE:
:FIXED;
CALL XFRER TO RETRIEVE DATA AND SET UP FOR STORE
:CHAR;
CALL CHROBJ TO RETRIEVE DATA AND SET UP FOR STORE
:CHAR;
CALL CHROBJ TO RETRIEVE DATA AND SET UP FOR STORE
ENDIF
IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT, THEN
POP OBJECT'S DISPLACEMENT FROM RESULT STACK
ELSE OBJECT HAS NOT BEEN SUBSCRIPTED
SET OBJECT'S DISPLACEMENT TO 0
ENDIF
BUILD XPREQ REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S
DISPLACEMENT
CALL XPREQ TO STORE DATA IN OBJECT
END REPLAC
1 BEGIN RNGSET
2  DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
3  * RANGES ARE PROCESSED
4  IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
5  INCREMENT RANGE VALUE
6  ELSE
7  SET RANGE INDEX VALUE TO BEGIN VALUE
8  ENDIF
9  END DO
10 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
FROM ASGCOM - SYMTAB,RSLTPT

OUTPUTS TO ASGCOM
RESULT,RSLTPT

EXTERNAL REFERENCES
FDS - XRMV
RTE - IAMD
XIRET
REPRODUCIBILITY OF
ORIGINAL PAGE IS POOR.
FORTRAN CALLING PROCEDURE

CALL XSYM (TKMPTR, FLAG, SYMIND)

XSYM PROVIDES SYMBOL TABLE BUILDING AND ACCESS FOR ASSGN. IT USES XSYM TO PERFORM GENERAL SYMBOL TABLE FUNCTIONS THEN ACCOMPLISHES RANGE AND SUBSCRIPT PROCESSING DIRECTLY.

INPUT

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

FLAG - SPECIAL PROCESSING FLAG

NORMAL PROCESSING

1, SYMBOL IS A RANGE

2, SYMBOL IS SUBSCRIBED

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

OUTPUT

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

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

ASGCOM COMMON - SYMTAB

INTERNAL

SYMPTR - INDEX TO FLAG WORD OF ENTRY IN SYMTAB. VALUE IS EQUAL TO SYMIND * STWIDE

EXTERNAL REFERENCES

XPIT, XILSS, XIMSG, XSYM
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

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

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

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

ENDIF

SET ENTRY FLAG WORD TO FLAG VALUE

ENDIF

ENDIF

EXIT XSYM

:BADRNG:

CALL XIMSG TO OUTPUT ERROR DESCRIPTION

CALL XILSS TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR

CALL XEXIT TO TERMINATE PROCESSOR

END XSYM
FORTRAN CALLING PROCEDURE

CALL XISYT (TKNPTR, SYMIND)

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

INPUT

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

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

OUTPUT

SYMIND - index into symbol table (SYMTAB) where symbol is located.

ASCOM COMMON - LASTSY, SYMTAB

INTERNAL

DISP - displacement into symbol table entry for field to be used as key. Names use a value of 1, constants a value of 8.

ELOOP - index of last allocated word in SYMTAB. Value is equal to LASTSY*STWIDE

IBUF - internal buffer for symbol value used to assure last words of integer and real constants are zero.

EXTERNAL REFERENCES

XRCPR, XRMOV

SPECIAL REMARKS

The required format of each symbol table entry is

I = token code (3 words) I reserved for toc entry (3 words) I other

I TYPE (1 word) I NAME I UNUSED (4 words) I VALUE I OTHER
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 ENDIF
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
DO - CONDITIONAL ITERATION (LOOPING) PROCESSOR

SCHEDULED BY FBS

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

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

INPUT FROM INTERFACE TABLE
DOTTYP - 4CH CHARACTER STRING SPECIFYING LOOP CONDITION TYPE
UNTIL - LOOP UNTIL RELATION BETWEEN OPNRD1 AND OPNRD2 IS
TRUE. TEST IS MADE AT THE END OF EACH LOOP
WHILE - LOOP WHILE RELATION BETWEEN OPNRD1 AND OPNRD2 IS
TRUE. TEST IS MADE PRIOR TO STARTING EACH LOOP

OPNRD1 - FIRST REAL QUANTITY TO COMPARE (VALUE NOT RETRIEVED,
CHARACTERISTICS PASSED TO ENDDO)
OPNRD2 - SECOND REAL QUANTITY TO COMPARE (SEE OPNRD1)

INPUT FROM AWA OM REQUEST TO MANAGER
INBUF - INTERFACE TABLE (LESS LITERAL AREA)
LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE
LITERAL AREA
SOSTK - SEE OUTPUT DEFINITION
BSERTB - EXEC COPY OF CURRENTLY EXECUTING SEQUENCE TABLE

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

INPUT/OUTPUT FROM/TO AWA
AOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(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
WITH THE FOLLOWING CHANGES (SEE FDS SD6,
VOLUME IV, FIGURE 1.2-17)
WORDS 8-14 - INTERFACE TABLE ENTRY FOR OPMD1 (SAME AS
IN ORIGINAL TABLE EXCEPT FOR POSSIBLE NEW
VALUES POINTING INTO LITERAL AREA)
WORDS 15-21 - INTERFACE TABLE ENTRY FOR OPMD2 (SEE
ABOVE)
WORDS 22-25 - LITERAL AREA FOR VALUE(s)/SUBSCRIPT(s) OF
OPMD1 & 2
WORD 26 - RELATION ID CODE AS FOLLOWS
0 - =
1 - >
2 - >= OR >=
3 - <
4 - <= OR <=
5 - <
WORD 27 - SEQUENCE NUMBER OF TOP OF LOOP
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CD********
2 CD0 FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
3 CD0
4 CD0
5 CD0
6 CD0
7 CD********
8 CD1
9 CD1 THE ELSE UTILITY PROCESSOR LOCATES THE ENDF CORRESPONDING TO
10 CD1 THIS ELSE AND SETS THE SEQUENCE RESET NUMBER (VIA XPXIT) SO
11 CD1 THE PROCESSOR FOLLOWING ENDF IS EXECUTED NEXT IN THE SEQUENCE.
12 CD0
13 CD1 CD********
14 CD1 CD2 INPUT
15 CD1 CD2 SCHEDULING PARAMETERS
16 CD1 CD2
17 CD2
18 CD2
19 CD2
20 CD2
21 CD2
22 CD2
23 CD2
24 CD2
25 CD2
26 CD2
27 CD2
28 CD2
29 CD2
30 CD2
31 CD2
32 CD2
33 CD2
34 CD2
35 CD2
36 CD2
37 CD2
38 CD2
39 CD2
40 CD2
41 CD2
42 CD2
43 CD2
44 CD2
45 CD2
46 CD2
47 CD2
48 CD2
49 CD2
50 CD2
51 CD2
52 CD2
53 CD2
54 CD2
55 CD2
56 CD2
57 CD2
58 CD2
59 CD2
60 CD2
61 CD2
62 CD2
63 CD2
64 CD2
65 CD2
66 CD2
67 CD2
68 CD2
69 CD2
70 CD2
71 CD2
72 CD2
73 CD2
74 CD2
75 CD2
76 CD2
77 CD2
78 CD2
79 CD2
80 CD2
81 CD2
82 CD2
83 CD2
84 CD2
85 CD2
86 CD2
87 CD2
88 CD2
89 CD2
90 CD2
91 CD2
92 CD2
93 CD2
94 CD2
95 CD2
96 CD2
97 CD2
98 CD2
99 CD2
100 CD2
101 CD2
102 CD2
103 CD2
104 CD2
105 CD2
106 CD2
107 CD2
108 CD2
109 CD2
110 CD2
111 CD2
112 CD2
113 CD2
114 CD2
115 CD2
116 CD2
117 CD2
118 CD2
119 CD2
120 CD2
121 CD2
122 CD2
123 CD2
124 CD2
125 CD2
126 CD2
127 CD2
128 CD2
129 CD2
130 CD2
131 CD2
132 CD2
133 CD2
134 CD2
135 CD2
136 CD2
137 CD2
138 CD2
139 CD2
140 CD2
141 CD2
142 CD2
143 CD2
144 CD2
145 CD2
146 CD2
147 CD2
148 CD2
149 CD2
150 CD2
151 CD2
152 CD2
153 CD2
154 CD2
155 CD2
156 CD2
157 CD2
158 CD2
159 CD2
160 CD2
161 CD2
162 CD2
163 CD2
164 CD2
165 CD2
166 CD2
167 CD2
168 CD2
169 CD2
170 CD2
171 CD2
172 CD2
173 CD2
174 CD2
175 CD2
176 CD2
177 CD2
178 CD2
179 CD2
180 CD2
181 CD2
182 CD2
183 CD2
184 CD2
185 CD2
186 CD2
187 CD2
188 CD2
189 CD2
190 CD2
191 CD2
192 CD2
193 CD2
194 CD2
195 CD2
196 CD2
197 CD2
198 CD2
199 CD2
200 CD2
201 CD2
202 CD2
203 CD2
204 CD2
205 CD2
206 CD2
207 CD2
208 CD2
209 CD2
210 CD2
211 CD2
212 CD2
213 CD2
214 CD2
215 CD2
216 CD2
217 CD2
218 CD2
219 CD2
220 CD2
221 CD2
222 CD2
223 CD2
224 CD2
225 CD2
226 CD2
227 CD2
228 CD2

ENDDO - LOOP TERMINATION PROCESSOR
SCHEDULED BY FDS

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FDS 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 AWA
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 FDS SYSTEM STATUS TABLE, XYSTB, IS ACCESS FOR THE MANAGER/
PROCESSOR CLASS I/O COMMUNICATIONS NUMBER

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

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

XPCS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

REFERENCES ROUTINES
EXEC, IAND, IRPAP, XPGET, XPREQ(XPGET), XPIXIT, XRLOC, XRMOV,
XUDBG, XVPAR, XIMSG
298  1 BEGIN ENDDO
299  2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XVS TB
300  2 CALL XPAN TO RETRIEVE $DOSTK
301       2 IF RETRIEVAL WAS SUCCESSFUL, THEN
302       3 WRITE $INTAB TO CLASS I/O FROM LAST $DOSTK ENTRY
303       3 WRITE NON-LITERAL PORTION OF $INTAB TO CLASS I/O (LEAVE FOR XPGET)
304       3 CALL XVPAN TO RESTORE $INTAB FROM FIRST CLASS BUFFER INTO AWA
305       3 IF RESTORE SUCCESSFUL, THEN
306       4 CALL XPGET TO RETRIEVE $PRMS
307       4 CALL XEVL TO EVALUATE RELATION
308       4 IF RELATION IS TRUE, THEN
309       5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
310       5 IF THIS IS LAST $DOSTK ENTRY, THEN
311       6 DELETE $DOSTK FROM AWA
312       6 ELSE
313       7 CALL XPREQ TO REPLACE $DOSTK LESS LAST ENTRY
314       7 ENDF
315       8 ELSE
316       9 SET RESET NUMBER TO TOP OF LOOP
317      10 ENDF
318      11 ELSE
319      12 TERMINATE WITH ERR06 FOR AWA OVERFLOW
320      12 ENDF
321      13 ELSE
322      14 TERMINATE WITH ERR03 FOR ENDDO WITH OUT DO
323      14 ENDF
324      1 END ENDDO
FORTAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

THE ENDIF utility processor returns to the FDS manager (via XPXIT) so that normal processor execution sequence may continue.

INPUTS

SCHEDULING PARAMETERS

LU = logical unit no. of FDS user

FLAGS = debug flags -- bit 11 on will cause debug print

OUTPUTS

RMPARMS - return parameters for FDS manager via XPXIT

(1) = 0 -> continue normal processor execution sequence

Routines used

RMPAR

XREXT

XPXIT

XUDBG

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF
**FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS**

- THE IF UTILITY PROCESSOR COMPARES TWO INPUT QUANTITIES ACCORDING TO A SPECIFIED RELATION. WHEN THE COMPARISON IS FALSE, THE SEQUENCE RESET NUMBER IS SET (VIA XPSET) TO THE PROCESSOR FOLLOWING THE CORRESPONDING ELSE (OR ENDIF). IF THE CONDITION IS TRUE THE NORMAL PROCESSOR EXECUTION SEQUENCE IS FOLLOWED.

**INPUT**

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

**INTERFACE TABLE PARAMETERS**

- OPRAND1 - FIRST REAL VALUE TO COMPARE
- RELATN - RELATION IDENTIFIER (>,>=,=,<=,<,<=,<,<=)
- OPRAND2 - SECOND REAL VALUE TO COMPARE

**OUTPUT**

- RPAM - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPSET
  - (1) = 0 => CONTINUE WITH NORMAL PROCESSOR EXECUTION SEQUENCE
  - (2) = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE NO. GIVEN IN RPAMS(2)
  - (3) = 8 => ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE

**INCLUDED ROUTINES**

- RESENCED ROUTINES
- RMPAR
- XPSET
- XPIIT
- XSYL
- XSRC
- XNEXT
- XCHANGE

**OTHER ROUTINES**

- CDU
- CD2
1 BEGIN IF
2 CALL IMPAR TO GET INPUT (SHEDuling) PARAMETERS
3 CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
4 VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSLATE IT TO A CODE
5 ERREXIT IF RELATIONAL OPERATOR INVALID :ERROR5:
6 CALL XIEVL TO EVALUATE THE RELATIONAL EXPRESSION
7 IF THE EXPRESSION IS FALSE, THEN
8 CALL XISCH TO LOCATE THE ELSE OR ENL': CORRESPONDING TO THIS IF
9 AND SET SEQUENCE RESET NUMBER
10 ELSE
11 CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE
12 ENDIF
13 EXIT IF
14 ERREXIT: CALL XIMSG 'INVALID RELATIONAL OPERATOR - MUST BE #,>,>,>=,<=,OR =>
15 END IF
**FORTRAN CALLING PROCEDURE**

**NAME:** XIEVL (OPRD1, RELTN, OPRD2)

**DESCRIPTION:** XIEVL compares two real single precision values under a specified relation and returns a function value of logical true or false.

**INPUT**
- OPRD1 - First real value to compare
- RELTN - 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
- OPRD2 - Second real value to compare

**OUTPUT**
- XIEVL - Function value of logical true or false

**INTERNAL VARIABLES**
- TTABNAME - Bit mask representing truth table values derived as follows:

<table>
<thead>
<tr>
<th>OPRD1 - OPRD2</th>
<th>CODE RELATION + O</th>
<th>RELATION + 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OPRD1 - OPRD2</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Therefore for OPRD1 - OPRD2 positive (bits 0-2) 1, 2, 3, 4, 5
For OPRD1 - OPRD2 zero (bits 3-5) 0, 0, 0
And for OPRD1 - OPRD2 negative (bits 6-8) 1, 0, 0
(Bits 9-15 are not used) if RELTN is 0, 1, 2 or the complement if RELTN is 3, 4, or 5

**REferenced Routines**
- XEXIT
- NO checks for valid relation code; or overflow/underflow are made
BEGIN X:EVL
CASE (I+:; I:0; I:-) DIFFERENCE OF OPRND1 AND OPRND2
  I+: SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)
  I:0: SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
  I:-: SET FIELD OFFSET TO SIX (BITS 6-8 OF TTABLE)
END CASE
IF RELATH > 2 (BOTTOM OF TRUTH TABLE), THEN
  COMPLEMENT TTABLE
  DECREMENT RELATH BY 3
ENDIF
ADD RELATH TO FIELD OFFSET (INDEXES TO CORRECT BIT FOR RELATH AND DIFFERENCE)
SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
END X:EVL
FUNCTION CALLING PROCEDURE

CALL ZISCH (SRCNF, PPARMS)

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

INPUTS

CALLING SEQUENCE

SRCNF - FLAG INDICATING ORIGINATION OF THIS CALL
0 => CALLED BY IF TO FIND MATCHING ELSE OR ENDIF
1 => CALLED BY ELSE TO FIND MATCHING ENDIF

COMMON

DEBUG - FLAG INDICATING WHETHER ONLINE DEBUG TO BE OUTPUT
0 => NO DEBUG
1 => DEBUG

PPARMS - SCHEDULED PARAMETERS FROM THE FBS MANAGER
1 = LOGICAL UNIT NO. OF THE FBS USER
5 = SEQSP, INDEX INTO SEQUENCE TABLE (&SEQTAB) OF THE CURRENT
COMMAND

OUTPUTS

CALLING SEQUENCE

PPARMS - PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XPIT?
1 = CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
3 = EXECUTE SEQUENCE NO. GIVEN BY PPARMS(2) NEXT
6 = ABNORMALLY TERMINATE PROCESSOR EXECUTION SEQUENCE
(2) = SEQUENCE NO. TO BE EXECUTED NEXT IF PPARMS(1) = 3

ROUTINES USED

EXEC
MPAR
XPPAR
XUDOS
XRCPB
XMSG

BEGIN XISCH
RETRIEVE &SETTAB FROM THE AMA USING XVPAW
STARTSEARCH UNTIL ALL COMMANDS IN &SETTAB
EXIT IF CURRENT COMMAND IS FOUND
SET NUMBER OF IF NESTS TO 1
STARTSEARCH FROM NEXT COMMAND IN &SETTAB UNTIL ALL FOLLOWING COMMANDS
IF COMMAND IS FOR ENDF PROCESSOR, THEN
DECREMENT NUMBER OF IF NESTS BY 1
ELSE
IF COMMAND IS FOR IF PROCESSOR, THEN
INCREMENT NUMBER OF IF NESTS BY 1
ELSE
IF CALLED BY IF PROCESSOR, AND COMMAND IS FOR ELSE PROCESSOR, THEN
ERROR IF THIS IS THE END OF &SETTAB :ERROR1:
IF NUMBER OF IF NESTS IS 1, THEN
DECREMENT NUMBER OF IF NESTS TO 0
ENDIF
ENDIF
EXIT IF NUMBER OF IF NESTS IS 0
SET SEQUENCE RESET NUMBER (RPARMS(2)) TO BE SEQUENCE NUMBER OF THE NEXT COMMAND IN THE TABLE
ENDLOOP
ERROR :ERROR1:
ENDSEARCH
ENDLOCOP
ERROR :ERROR4:
ENDSEARCH
EXIT XISCH
:ERROR1: CALL XIMSG = "IF CANNOT "E EXECUTED WITHOUT MATCHING ENDF"
:ERROR4: CALL X.MSG = "SYSTEM ERROR - NO &SETTAB"
END XISCH
<table>
<thead>
<tr>
<th>Symbol Definition Table</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO</td>
<td>124</td>
</tr>
<tr>
<td>ELSE</td>
<td>230</td>
</tr>
<tr>
<td>ENDDO</td>
<td>298</td>
</tr>
<tr>
<td>ENDF</td>
<td>361</td>
</tr>
<tr>
<td>ERRO1</td>
<td>591</td>
</tr>
<tr>
<td>ERRO2</td>
<td>186</td>
</tr>
<tr>
<td>ERRO3</td>
<td>428</td>
</tr>
<tr>
<td>ERRO4</td>
<td>187</td>
</tr>
<tr>
<td>ERR04</td>
<td>592</td>
</tr>
<tr>
<td>IF</td>
<td>415</td>
</tr>
<tr>
<td>OTHER</td>
<td>175</td>
</tr>
<tr>
<td>UNTIL</td>
<td>173</td>
</tr>
<tr>
<td>WHILE</td>
<td>157</td>
</tr>
<tr>
<td>XIEVL</td>
<td>490</td>
</tr>
<tr>
<td>ZISCH</td>
<td>494</td>
</tr>
<tr>
<td>TEN</td>
<td>492</td>
</tr>
<tr>
<td>FIN</td>
<td>493</td>
</tr>
</tbody>
</table>
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
<table>
<thead>
<tr>
<th>NAME</th>
<th>VERSION</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ #</th>
<th>SIZE-PRE-TEXT</th>
<th>CYCLE</th>
<th>WORD</th>
<th>PSRMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOR2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>55</td>
<td>0</td>
<td>1</td>
<td>1797</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:54:07</td>
<td>2</td>
<td>30</td>
<td>0</td>
<td>1</td>
<td>1845</td>
<td></td>
</tr>
<tr>
<td>SOR2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:59:32</td>
<td>3</td>
<td>55</td>
<td>0</td>
<td>1</td>
<td>1875</td>
<td></td>
</tr>
<tr>
<td>XUPDL</td>
<td></td>
<td>RELOCATABLE</td>
<td>15 APR 77</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAP</td>
<td></td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>03:17:22</td>
<td>4</td>
<td>84</td>
<td>0</td>
<td>1</td>
<td>1932</td>
<td></td>
</tr>
<tr>
<td>FPLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>0</td>
<td>3</td>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>FAP</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2013</td>
<td></td>
</tr>
<tr>
<td>FPLIST</td>
<td></td>
<td>ABSOLUTE</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>280</td>
<td>SET</td>
<td></td>
<td>2105</td>
<td></td>
</tr>
<tr>
<td>RAP</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:45</td>
<td>8</td>
<td>109</td>
<td>5</td>
<td>6</td>
<td>2105</td>
<td></td>
</tr>
<tr>
<td>RAP</td>
<td></td>
<td>ELT SYMB</td>
<td>10 AUG 77</td>
<td>09:10:45</td>
<td>9</td>
<td>394</td>
<td>5</td>
<td>2</td>
<td>2494</td>
<td></td>
</tr>
<tr>
<td>MESAGE</td>
<td></td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>09:19:22</td>
<td>10</td>
<td>11</td>
<td>5</td>
<td>12</td>
<td>2888</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>09:19:22</td>
<td>11</td>
<td>35</td>
<td>5</td>
<td>12</td>
<td>2893</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>10 MAR 78</td>
<td>09:19:22</td>
<td>12</td>
<td>373</td>
<td>5</td>
<td>8</td>
<td>2923</td>
<td></td>
</tr>
<tr>
<td>COMB</td>
<td></td>
<td>ELT SYMB</td>
<td>10 MAR 78</td>
<td>09:19:22</td>
<td>13</td>
<td>824</td>
<td>5</td>
<td>9</td>
<td>3296</td>
<td></td>
</tr>
<tr>
<td>COMB</td>
<td></td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>09:19:22</td>
<td>14</td>
<td>627</td>
<td>5</td>
<td>14</td>
<td>4120</td>
<td></td>
</tr>
<tr>
<td>COMB</td>
<td></td>
<td>ELT SYMB</td>
<td>22 FEB 78</td>
<td>09:19:22</td>
<td>15</td>
<td>120</td>
<td>5</td>
<td>20</td>
<td>4747</td>
<td></td>
</tr>
<tr>
<td>COMB</td>
<td></td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>09:19:22</td>
<td>16</td>
<td>200</td>
<td>5</td>
<td>20</td>
<td>4857</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>RELOCATABLE</td>
<td>03 MAR 78</td>
<td>09:19:22</td>
<td>17</td>
<td>2</td>
<td>18</td>
<td></td>
<td>5067</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>RELOCATABLE</td>
<td>03 MAR 78</td>
<td>09:19:22</td>
<td>19</td>
<td>101</td>
<td>5</td>
<td>10</td>
<td>5068</td>
<td></td>
</tr>
<tr>
<td>XP</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:29</td>
<td>20</td>
<td>101</td>
<td>5</td>
<td>10</td>
<td>5094</td>
<td></td>
</tr>
<tr>
<td>XP</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:29</td>
<td>21</td>
<td>534</td>
<td>5</td>
<td>10</td>
<td>5193</td>
<td></td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>10:54:29</td>
<td>22</td>
<td>534</td>
<td>5</td>
<td>10</td>
<td>5195</td>
<td></td>
</tr>
<tr>
<td>XM</td>
<td></td>
<td>ELT SYMB</td>
<td>16 MAY 78</td>
<td>09:12:58</td>
<td>23</td>
<td>384</td>
<td>5</td>
<td>6</td>
<td>5729</td>
<td></td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>24</td>
<td>1548</td>
<td>5</td>
<td>4</td>
<td>6113</td>
<td></td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>25</td>
<td>403</td>
<td>5</td>
<td>20</td>
<td>7661</td>
<td></td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>26</td>
<td>504</td>
<td>5</td>
<td>20</td>
<td>8064</td>
<td></td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>27</td>
<td>151</td>
<td>5</td>
<td>5</td>
<td>8532</td>
<td></td>
</tr>
<tr>
<td>ERECE</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>28</td>
<td>118</td>
<td>5</td>
<td>13</td>
<td>8723</td>
<td></td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>29</td>
<td>1201</td>
<td>5</td>
<td>21</td>
<td>8841</td>
<td></td>
</tr>
<tr>
<td>XU</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>30</td>
<td>600</td>
<td>5</td>
<td>29</td>
<td>10042</td>
<td></td>
</tr>
<tr>
<td>XU</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>31</td>
<td>326</td>
<td>5</td>
<td>2</td>
<td>10562</td>
<td></td>
</tr>
<tr>
<td>XU</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>32</td>
<td>1094</td>
<td>5</td>
<td>10</td>
<td>10968</td>
<td></td>
</tr>
</tbody>
</table>

**NEXT AVAILABLE LOCATION**

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

**DBXOT F.POLIST**