NOTICE

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

Executive Logic Flow - Program
Design Language

Mission Planning and Analysis Division
December 1979

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

FLIGHT DESIGN SYSTEM-1
SYSTEM DESIGN DOCUMENT

EXECUTIVE LOGIC FLOW - PROGRAM DESIGN LANGUAGE

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

JSC Task Monitor: Software Development Branch

Approved: Elric N. McHenry, Chief
Software Development Branch

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

Mission Planning and Analysis Division
National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas
December 1979
PREFACE

The Flight Design System-1 (FDS-1) is a pilot project to evaluate current concepts and to determine the hardware/software capability that will be required for the operational era to support Shuttle flight planning. This software system is being implemented on a Hewlett-Packard 21MX computer with a Daconics documentation system and will provide terminal-based interactive flight planning capability.

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

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

*Combined as one volume with title: Volume III FDS-1 Processor Library
<table>
<thead>
<tr>
<th>Section</th>
<th>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)

INTEGER

* CARTPG
* CLASNO
* COMPR
* FLAGS
* PRENAME(8)
* REBUF(64)
* SEREND
* SUBSTA
* TKMLNG
* TOKENS(32)
* XOR
* XE

DIMENSION

* INTNAM(3)

EQUIVALENCE

* XE(1), LU
* XE(3), DUAL
* XE(5), MASSTA
* XE(7), SEGNAME(2)
* XE(11), SEREND
* XE(13), INTNAM(1)
* XE(19), REBUF(1)
* XE(86), TKMLNG
* XE(139), EXEND
* XE(143), XE(145)

CARTPG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

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

COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER

COMPR - NUMBER OF TOKENS IN BUFFER

INTNAM - TOKENS REPRESENTING USER'S RESPONSE

COMTE - POINTERS TO TOKENS CURRENTLY BEING PROCESSED FROM COMBUF

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

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

BITS 0-10 UNUSED

PROFESSION ON-LINE DEBUG

MANAGER ON-LINE DEBUG

EXECUTIVE ON-LINE DEBUG

PRODUCE A DUMP ON ALL TERMINATIONS

MANAGER REQUEST TRANSACTION TRACE FLAG

INTNAM - NAME OF INTERFACE TABLE INPUT TO INTERFACE TABLE EDITOR
XD CONTENTS (EXECUTION CONTROL LEVEL)

INTEGER
DIMENSION

!* ASCENT  CURLINO
!* SERLEN  SERTAB
!* SERNO  

DIMENSION  LIB(150)

EQUIVALENCE

!* (XB(1), NPROC2)  *(XB(2), LIB(1))
!* (XB(236), ASCENT(1))  *(XB(235), RESIND)
!* (XB(249), SERNO)  *(XB(2),(6), CURIND)
!* (XB(250), SERLEN)  *(XB(251), SERTAB(1))

!* ASCENT - SEQUENCE TABLE ENTRY IN ASCII TO PROMPT USER
!* IN SEMI MODE ONLY
!* CURIND - CURRENT INDEX TO EXECUTING SEQUENCE ENTRY
!* LIRD - LIBRARY DIRECTORY PROCESSOR NAME TABLE
!* NPROC2 - NUMBER OF PROCESSORS IN LIRD (SAME AS KE(143))
!* RESIND - INDEX OF RESET ENTRY WHEN RESET SEQUENCE 0 IS REQUESTED
!* SERLEN - LENGTH OF SEQUENCE TABLE
!* SERNO - NUMBER OF ENTRIES IN SEQUENCE TABLE
!* SERTAB - SEQUENCE TABLE CURRENTLY BEING EXECUTED

!*
## XBCOMM

**Contents (Interface Table Edit Level)**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>INTEGER BITNO</td>
</tr>
<tr>
<td>252</td>
<td>BITNO BIT NO. IN BIT MASK OF NEXT DIFFERENT BIT</td>
</tr>
<tr>
<td>258</td>
<td>BITMUN</td>
</tr>
<tr>
<td>259</td>
<td>* * *</td>
</tr>
<tr>
<td>260</td>
<td>* * *</td>
</tr>
<tr>
<td>261</td>
<td>* * *</td>
</tr>
<tr>
<td>262</td>
<td>* * *</td>
</tr>
<tr>
<td>263</td>
<td>* * *</td>
</tr>
<tr>
<td>264</td>
<td>* * *</td>
</tr>
<tr>
<td>265</td>
<td>* * *</td>
</tr>
<tr>
<td>266</td>
<td>* * *</td>
</tr>
<tr>
<td>267</td>
<td>* * *</td>
</tr>
<tr>
<td>268</td>
<td>* * *</td>
</tr>
<tr>
<td>269</td>
<td>* * *</td>
</tr>
<tr>
<td>270</td>
<td>* * *</td>
</tr>
<tr>
<td>271</td>
<td>* * *</td>
</tr>
<tr>
<td>272</td>
<td>* * *</td>
</tr>
<tr>
<td>273</td>
<td>* * *</td>
</tr>
<tr>
<td>274</td>
<td>* * *</td>
</tr>
<tr>
<td>275</td>
<td>* * *</td>
</tr>
<tr>
<td>276</td>
<td>* * *</td>
</tr>
<tr>
<td>277</td>
<td>* * *</td>
</tr>
<tr>
<td>278</td>
<td>* * *</td>
</tr>
<tr>
<td>279</td>
<td>* * *</td>
</tr>
<tr>
<td>280</td>
<td>* * *</td>
</tr>
<tr>
<td>281</td>
<td>* * *</td>
</tr>
<tr>
<td>282</td>
<td>* * *</td>
</tr>
<tr>
<td>283</td>
<td>* * *</td>
</tr>
<tr>
<td>284</td>
<td>* * *</td>
</tr>
<tr>
<td>285</td>
<td>* * *</td>
</tr>
<tr>
<td>286</td>
<td>* * *</td>
</tr>
<tr>
<td>287</td>
<td>* * *</td>
</tr>
<tr>
<td>288</td>
<td>* * *</td>
</tr>
<tr>
<td>289</td>
<td>* * *</td>
</tr>
<tr>
<td>290</td>
<td>* * *</td>
</tr>
<tr>
<td>291</td>
<td>* * *</td>
</tr>
<tr>
<td>292</td>
<td>* * *</td>
</tr>
<tr>
<td>293</td>
<td>* * *</td>
</tr>
<tr>
<td>294</td>
<td>* * *</td>
</tr>
<tr>
<td>295</td>
<td>* * *</td>
</tr>
<tr>
<td>296</td>
<td>* * *</td>
</tr>
<tr>
<td>297</td>
<td>* * *</td>
</tr>
<tr>
<td>298</td>
<td>* * *</td>
</tr>
<tr>
<td>299</td>
<td>* * *</td>
</tr>
<tr>
<td>300</td>
<td>* * *</td>
</tr>
<tr>
<td>301</td>
<td>* * *</td>
</tr>
<tr>
<td>302</td>
<td>* * *</td>
</tr>
<tr>
<td>303</td>
<td>* * *</td>
</tr>
<tr>
<td>304</td>
<td>* * *</td>
</tr>
<tr>
<td>305</td>
<td>* * *</td>
</tr>
<tr>
<td>306</td>
<td>* * *</td>
</tr>
<tr>
<td>307</td>
<td>* * *</td>
</tr>
<tr>
<td>308</td>
<td>* * *</td>
</tr>
<tr>
<td>309</td>
<td>* * *</td>
</tr>
</tbody>
</table>
JSUB F - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
LENF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
LISTU - LU TO WHICH PRINT SHOULD GO (USED WHEN
LST (DIRECTIVE CALLS XILSD OR XICHRT)
LITDSP - DISPL. TO LITERAL DATA FOR THIS ARGUMENT
LITDWN - INDEX IN WKBUFF TO END OF LITERAL DATA
LITLEN - LENGTH OF LITERAL DATA AREA OF WKBUFF
LITPR - INDEX IN WKBUFF TO START OF LITERAL DATA
LITSZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
LSTFLG - FLAG USED TO DETERMINE ORIGIN OF A
CALL TO XILSD OR XICHRT.

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

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

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

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

ORGANIZED AS:

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

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

**ATTENTION FUNCTION**

**XA01** MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION
**XA02** USER INITIATED INTERRUPT ENTER REQUEST - KILL(S), STATUS(S), RETURN( )
**XA03** ERROR, LU IS NOT SIGNED ON TO FDS
**XA04** FDS ATTENTION FUNCTION TERMINATING
**XA05** FDS MANAGER SIGNALED TO TERMINATE BOTTOM ASSOC TASK
**XA06** FDS EXECUTIVE ACTIVE - NO ACTION TAKEN
**XA07** FDS PROCESSOR 'NAME' SCHEDULED TO ABORT
**XA08** MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN

**BATCH JOB CREATION**

**XC01** LU 'NN' SIGNED ON TO FDS
**XC02** LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED
**XC03** 'LU' IS AN INVALID LU
**XC04** FDS CURRENTLY AT MAX USER'S. CANNOT SIGN ON
**XC05** 'LU' IS ALREADY SIGNED ON TO FDS
**XC06** CANNOT FIND 'NAME' ID - SIGN ON TERMINATED
**XC07** ENTER VALID USER ID (A-Z)

**MANAGER**

**XM01** INVALID REQUEST II FROM 'NAME'
**XM02** FDS SIGN OFF FOR LU 'LU'
**XM03** INVALID BACK CHAIN FOR CURRENT PROCESSOR
**XM04** 'H' TRACKS NOT AVAILABLE FOR DWA
**XM05** NO DWA SPACE FOR 'NAME'
**XM06** NO AWA FOR DWA DIRECTORY
**XM07** I/O ERROR FOR DWA, STATUS = 'NNNNNN'
**XM08** TOO SEARCH ERROR, PHASE 3 COMPRESS

**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 NM
**XP10** PROCESSOR INITIALIZE ERROR
**XP11** AWA ACCESS FAILURE FOR
**XP12** PARAMETER I/O INCONSISTENCY
**XP13** INVALID PARAMETER REQUEST
**XP14** INVALID ORD FILE ATTRIBUTE(S)
**XP15** RETRIEVAL OF TOO MUCH DATA REQUESTED

**SYSTEM SERVICES**

**XX01** 'ID-NAME' NAMED PROGRAM MADE A PAM REQUEST RESERVED FOR MANAGER
**XX02** 'ID-NAME' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER
**XX03** '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 . 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 FSTWD & 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         SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
11         ENDIF
12     ELSE
13       CLEAR LEVEL INCREMENT
14       IF COLUMN 1 NOT = -
15         THEN
16           IF FIRST CHARACTER = : - INDICATES :LABEL:
17             THEN
18               SET LINE SKIP IN IMAGE
19               GENERATE DEFINITION TABLE ENTRY FOR LABEL
20             ELSE
21               CALL FSTVND TO GET FIRST WORD OF PDL
22               LOOKUP FIRST WORD IN KEY WORD TABLE
23               KEY WORD TABLE CONTAINS
24               1 BEGIN - BEGIN SECTION INDICATOR
25               2 IF - SECTION INDICATOR
26               3 DO - SECTION INDICATOR
27               4 DOFOR - SECTION INDICATOR
28               5 DOUNTI - SECTION INDICATOR
29               6 DOWNIL - SECTION INDICATOR
30               7 CASE - SECTION INDICATOR
31               8 START - SECTION INDICATOR
32               9 STARTS - SECTION INDICATOR
33              10 ELSE - SECTION SEPARATOR
34              11 THEN - SECTION SEPARATOR
35              12 EXIT - SECTION SEPARATOR
36              13 EXITF - SECTION SEPARATOR
37              14 OR - SECTION SEPARATOR
38              15 DELSE - SECTION SEPARATOR
39              16 ENDOLOD - SECTION SEPARATOR
40              17 END - END OR END LOOP?
41              18 ENDF - SECTION TERMINATOR
42              19 ENHDO - SECTION TERMINATOR
43              20 ENDCAS - SECTION TERMINATOR
44              21 ENDOSEA - SECTION TERMINATOR
45
46  IF KEY WORD LOCATED
47    THEN
48      CASE LOCATION (:BEGIN, :SECIND, :SECIND, :SECIND, :SECIND, :SECIND, :SECIND,
51      BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
52      SET LEVEL INCREMENT = 1
53      :SECIND: SET LEVEL INCREMENT = 1
85 8 :SEP: SET LEVEL INCREMENT = 1
86 8 DECOREMENT LEVEL
90 8 PDLIST
91 8 :ENDT: CALL FSTWDR TO GET NEXT WORD OF PDL
92 8 DECOREMENT LEVEL
93 8 IF WORD = LOOP
94 8 THEN
95 9 SET LEVEL INCREMENT = 1
96 8 ENDIF
97 8 PDLIST
98 7 END CASE
99 6 ENDIF
100 5 ENDIF
101 5 COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
102 4 ELSE
103 5 SET INDENTATION FACTOR = 1
104 4 ENDIF
105 4 CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
106 4 INPUT RECORD AND ID FIELD
107 4 OUTPUT IMAGE
108 4 APPLY LEVEL INCREMENT
109 4 CLEAR PAGE EJECT
110 3 ENDIF
111 2 ENDDO
112 2 CALL SORT1 TO ORDER DEFINITION TABLE
113 2 OUTPUT DEFINITION TABLE
114 1 END PDLIST
EXTRACT THE FIRST TOKEN FROM A PDW RECORD

116 1 CD1
117 1 CD1
118 1 CD1
119 1 CD1
120 1 CD1
121 1 CD1
122 1 CD1
123 1 CD1
124 1 CD1
125 1 CD1
126 1 CD1
127 1 CD1
128 1 CD1
129 1 CD1
130 1 CD1
131 1 CD1
132 1 CD1
133 1 CD1
134 1 CD1
135 1 CD1
136 1 CD1
137 1 CD1
138 1 CD1
139 1 CD1
140 1 CD1
141 1 CD1
142 1 CD1
143 1 CD1
144 1 CD1
145 1 CD1

INPUT
SINGLE CHARACTER PER WORD RECORD AND LENGTH

OUTPUT
FIRST (NEXT) TOKEN IN SIX CHARACTER WORD. BLANKS AND : ARE
DELIMITERS. THE DISPLACEMENT OF THE NEXT CHARACTER IN THE RECORD IS
ALSO OUTPUT

BEGIN FSTWRD
BLANK OUTPUT WORD
LOCATE FIRST NON-BLANK CHARACTER
DO UNTIL SIX CHARACTERS STORED OR END-OF-RECORD
IF CHARACTER IS NON-BLANK AND NON-
THEN
STORE CHARACTER
ELSE
EXIT DO
ENDIF
ENDDO
RETURN LOCATION
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>1</td>
<td>NAME</td>
</tr>
<tr>
<td>2</td>
<td><strong>DEFINITION</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>DIRECT</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>7</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>8</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>9</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>10</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>11</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>12</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>13</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>14</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>15</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>16</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>17</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>18</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>19</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>20</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>21</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>22</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>23</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>24</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>25</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>26</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>27</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>28</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>29</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>30</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>31</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>32</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>33</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>34</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>35</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>36</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>37</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>38</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>39</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>40</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>41</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>42</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>43</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>44</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>45</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>46</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>47</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>48</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>49</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>50</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>51</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>52</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>53</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>54</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>55</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>56</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>57</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>58</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>59</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>60</td>
<td><strong>EXECUTION SEGMENT INTERFACE ROUTINE</strong></td>
</tr>
<tr>
<td>ENTRY</td>
<td>PROCEDURE</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>61</td>
<td>XIMPT</td>
</tr>
<tr>
<td>62</td>
<td>XINTER</td>
</tr>
<tr>
<td>63</td>
<td>XIPAR</td>
</tr>
<tr>
<td>64</td>
<td>XISPT</td>
</tr>
<tr>
<td>65</td>
<td>XIPROM</td>
</tr>
<tr>
<td>66</td>
<td>XISUB</td>
</tr>
<tr>
<td>67</td>
<td>EXLibrary Maintenance Program</td>
</tr>
<tr>
<td>68</td>
<td>XLCDB</td>
</tr>
<tr>
<td>69</td>
<td>XLDSE</td>
</tr>
<tr>
<td>70</td>
<td>XLDDEL</td>
</tr>
<tr>
<td>71</td>
<td>XLFLFL</td>
</tr>
<tr>
<td>72</td>
<td>XLLIN</td>
</tr>
<tr>
<td>73</td>
<td>XLMINT</td>
</tr>
<tr>
<td>74</td>
<td>XLMINT</td>
</tr>
<tr>
<td>75</td>
<td>XLMK</td>
</tr>
<tr>
<td>76</td>
<td>XLPFS</td>
</tr>
<tr>
<td>77</td>
<td>XLPFNL</td>
</tr>
<tr>
<td>78</td>
<td>XLPFH</td>
</tr>
<tr>
<td>79</td>
<td>XLPFHR</td>
</tr>
<tr>
<td>80</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>81</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>82</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>83</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>84</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>85</td>
<td>XLPFPH</td>
</tr>
<tr>
<td>86</td>
<td>ER Manager</td>
</tr>
<tr>
<td>87</td>
<td>EMAS</td>
</tr>
<tr>
<td>88</td>
<td>EMAN</td>
</tr>
<tr>
<td>89</td>
<td>ENAPX</td>
</tr>
<tr>
<td>90</td>
<td>ENAPX</td>
</tr>
<tr>
<td>91</td>
<td>ENAPX</td>
</tr>
<tr>
<td>92</td>
<td>ERMAN</td>
</tr>
<tr>
<td>93</td>
<td>ERMSP</td>
</tr>
<tr>
<td>94</td>
<td>EMQR</td>
</tr>
<tr>
<td>95</td>
<td>ERPR</td>
</tr>
<tr>
<td>96</td>
<td>ERTK</td>
</tr>
<tr>
<td>97</td>
<td>ERTK</td>
</tr>
<tr>
<td>98</td>
<td>ERTK</td>
</tr>
<tr>
<td>99</td>
<td>ERTK</td>
</tr>
<tr>
<td>100</td>
<td>ERPR</td>
</tr>
<tr>
<td>101</td>
<td>ERPR</td>
</tr>
<tr>
<td>102</td>
<td>ERPR</td>
</tr>
<tr>
<td>103</td>
<td>ERPR</td>
</tr>
<tr>
<td>104</td>
<td>ERPR</td>
</tr>
<tr>
<td>105</td>
<td>ERPR</td>
</tr>
<tr>
<td>106</td>
<td>ERPR</td>
</tr>
<tr>
<td>107</td>
<td>ERPR</td>
</tr>
<tr>
<td>108</td>
<td>ERPR</td>
</tr>
<tr>
<td>109</td>
<td>EPAR</td>
</tr>
<tr>
<td>110</td>
<td>EPGT</td>
</tr>
<tr>
<td>111</td>
<td>EPPF</td>
</tr>
<tr>
<td>112</td>
<td>EPPTI</td>
</tr>
<tr>
<td>113</td>
<td>EPPPTI</td>
</tr>
<tr>
<td>114</td>
<td>EPPDM</td>
</tr>
<tr>
<td>115</td>
<td>EPRDS</td>
</tr>
<tr>
<td>116</td>
<td>EPRFL</td>
</tr>
<tr>
<td>117</td>
<td>EPRIT</td>
</tr>
<tr>
<td>118</td>
<td>EPRIT</td>
</tr>
<tr>
<td>119</td>
<td>ER Executive Services</td>
</tr>
<tr>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>121</td>
<td>1</td>
</tr>
<tr>
<td>122</td>
<td>1</td>
</tr>
<tr>
<td>123</td>
<td>1</td>
</tr>
<tr>
<td>124</td>
<td>1</td>
</tr>
<tr>
<td>125</td>
<td>1</td>
</tr>
<tr>
<td>126</td>
<td>1</td>
</tr>
<tr>
<td>127</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>129</td>
<td>1</td>
</tr>
<tr>
<td>130</td>
<td>1</td>
</tr>
<tr>
<td>131</td>
<td>1</td>
</tr>
<tr>
<td>132</td>
<td>1</td>
</tr>
<tr>
<td>133</td>
<td>1</td>
</tr>
<tr>
<td>134</td>
<td>1</td>
</tr>
<tr>
<td>135</td>
<td>1</td>
</tr>
<tr>
<td>136</td>
<td>1</td>
</tr>
<tr>
<td>137</td>
<td>1</td>
</tr>
<tr>
<td>138</td>
<td>1</td>
</tr>
<tr>
<td>139</td>
<td>1</td>
</tr>
<tr>
<td>140</td>
<td>1</td>
</tr>
<tr>
<td>141</td>
<td>1</td>
</tr>
<tr>
<td>142</td>
<td>1</td>
</tr>
<tr>
<td>143</td>
<td>1</td>
</tr>
<tr>
<td>144</td>
<td>1</td>
</tr>
<tr>
<td>145</td>
<td>1</td>
</tr>
<tr>
<td>146</td>
<td>1</td>
</tr>
<tr>
<td>147</td>
<td>1</td>
</tr>
<tr>
<td>148</td>
<td>1</td>
</tr>
<tr>
<td>149</td>
<td>1</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>151</td>
<td>1</td>
</tr>
<tr>
<td>152</td>
<td>1</td>
</tr>
<tr>
<td>153</td>
<td>1</td>
</tr>
<tr>
<td>154</td>
<td>1</td>
</tr>
<tr>
<td>155</td>
<td>1</td>
</tr>
<tr>
<td>156</td>
<td>1</td>
</tr>
<tr>
<td>157</td>
<td>1</td>
</tr>
<tr>
<td>158</td>
<td>1</td>
</tr>
<tr>
<td>159</td>
<td>1</td>
</tr>
<tr>
<td>160</td>
<td>1</td>
</tr>
<tr>
<td>161</td>
<td>1</td>
</tr>
<tr>
<td>162</td>
<td>1</td>
</tr>
<tr>
<td>163</td>
<td>1</td>
</tr>
<tr>
<td>164</td>
<td>1</td>
</tr>
<tr>
<td>165</td>
<td>1</td>
</tr>
<tr>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>167</td>
<td>1</td>
</tr>
<tr>
<td>168</td>
<td>1</td>
</tr>
<tr>
<td>169</td>
<td>1</td>
</tr>
<tr>
<td>170</td>
<td>1</td>
</tr>
<tr>
<td>171</td>
<td>1</td>
</tr>
<tr>
<td>172</td>
<td>1</td>
</tr>
<tr>
<td>173</td>
<td>1</td>
</tr>
</tbody>
</table>
179 1  *  X5SEM  SENI - AUTOMATIC NODE  
180 1  *  X5STO  STORE SEQUENCE TABLE IN RSETAB  
181 1  *  X5TMP  TEMPORARY EXECUTION OF ONE ENTRY WITH RINTAB  
182 1  *  XZ  UTILITY PROCESSORS  
183 1  *  ASSGN  ASSIGN PROCESSOR  
184 1  *  DBDSP  DATA BOX DISPLAY PROCESSOR  
185 1  *  DEFIP  DEFINE PROCESSOR  
186 1  *  DO  CONDITIONAL LOOP IN SEQUENCE TABLE  
187 1  *  ELSE  EXECUTION POINT FOR FALSE IF CONDITION  
188 1  *  ENDEF  TERMINATES AN IF STRUCTURE  
189 1  *  ENDE00  TERMINATES A DO LOOP STRUCTURE  
190 1  *  ENSC  END SCAN PROCESSOR  
191 1  *  IF  CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES  
192 1  *  SCAN  SCAN PROCESSOR  
193 1  *  XICRP  CHARACTER OBJECT STORE FOR ASSEG  
194 1  *  XZDFT  FIND ANY TOKEN IN A SYMBOLIC STRING  
195 1  *  XZDIN  DATA BOX DISPLAY INPUT PROCESSOR  
196 1  *  XZDNC  DATA BOX DISPLAY CONSTRAINT MASKER  
197 1  *  XZDOT  DATA BOX DISPLAY OUTPUT ROUTINE  
198 1  *  XZDP1  DATA BOX DISPLAY PASS 1 PROCESSOR  
199 1  *  XZDP2  DATA BOX DISPLAY PASS 2 PROCESSOR  
200 1  *  XZEV  PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS  
201 1  *  XZFL  FIND PROCESSOR CLASS NUMBER  
202 1  *  XZFTC  FUNCTIONAL OPERATIONS FOR ASSEG  
203 1  *  XZHF  FREE OBJECT STORE FOR ASSEG  
204 1  *  XZFDK  FIXED OBJECT STORE FOR ASSEG  
205 1  *  XZIIP  REMOVE DUPL, BLANKS & BLANK FILL  
206 1  *  XZLLS  SYMBOLIC STRING SYNTAX ERROR LISTER  
207 1  *  XZMSG  FPS PROCESSOR MESSAGE ROUTINE  
208 1  *  XZMRT  MATHEMATICAL OPERATIONS FOR ASSEG  
209 1  *  XZPES  DATA CONVERSION AND STORAGE FOR ASSEG  
210 1  *  XZPS1  PASS 1 SUBROUTINE FOR ASSEG PROCESSOR  
211 1  *  XZPS2  PASS 2 SUBROUTINE FOR ASSEG PROCESSOR  
212 1  *  XZRET  DATA RETRIEVAL FOR ASSEG  
213 1  *  XZSCN  SEARCHES SEQUENCE TABLE FOR IF STRUCTURES  
214 1  *  XZSYM  SYMBOL TABLE INTERFACE FOR ASSEG  
215 1  *  XZSTT  SYMBOL TABLE MAINTENANCE  
216 1  *  XZSTT  SYMBOL TABLE MAINTENANCE  
217 1  *
SAVE EGT ADDRESS(IN BREG ON ENTRY)
CALL ENLU(BREG) GET L U IN ASCII & BINARY
STARTSEARCH UNTIL LAST STATUS TABLE ENTRY
EXITIF STBU EN L U
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STNG)
IF MANAGER IS DORMANT, THEN
WRITE "***RAO- MANAGER HAS TERMINATED;
REPLY TO CONTINUE TERMINATION:
READ(LU) ** WAIT FOR REPLY **
LOCK ON THE FDS TABLE RESOURCE
CALL SLBNR DISABLE
IF STBEX(E~ECUTIVE ADDRESS .NE. 0, THEN
IF STBEAT(CURRENT) .NE. ,STBE, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XMCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STNG(MANAGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARENT ONE(P1)
CALL SLBNR DISABLE
CALL MESS 'OFF,BOTTOM'
CALL SLBNR DISABLE
SET BOTTOM TO NEXT-AT
ENDO
ENDIF
ENDIF
CALL SLBNR MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBER
ENDIF
DECREMENT NUMBER ACTIVE(STBAC)
GET EGT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBER)
CLEAR STBER
CLEAR MANAGER'S ID, STNG, & STBU
ENABLE... (VIA A JMP TO EXEC(DISPATCHER))
RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS
CLEAR LOCK ON FDS TABLE
ELSE ** MANAGER IS STILL ALIVE **
WRITE "***RAO- USER INITIATED INTERRUPT'
WRITE 'ENTER REQUEST- KILL(S), STATUS(S),OR RETURN(BLANK)'
READ (LU) REQUEST
IF REQUEST IS KILL OR S, THEN
PERFORM XAKILL
ELSE IF REQUEST IS STATUS OR S, THEN
PERFORM XASTAF
ENDIF
ENDIF
ELSE
WRITE '***XAO- ERROR LU IS NOT SIGNED-ON TO FDS'
ENDIF
WRITE '***XAO4 FDS ATTENTION FUNCTION TERMINATING'
END
PRODUCE A DFS STATUS REPORT

1.FGIN XSTAT
2.CALL SLIBX - DISABLE
3.GET CURRENT-TIME FROM STIME
4.GET MANAGER'S ADDRESS FROM STMBG
5.MOVE NAME, STATUS, PARTITION, & PRIORITY
6.GET EXECUTIVE'S ADDRESS FROM STBEX
7.MOVE NAME, STATUS, PARTITION, & PRIORITY
8.GET CURRENT AT FROM STBAT
9.MOVE NAME, STATUS, PARTITION, & PRIORITY
10.PEFORM XABTM(CURRENT) FIND BOTTOM AT
11.SET BOTTOM TO CURRENT
12.GO WHILE FATHER-ID NE ZERO
13.SECOND FATHER-ID FROM CURRENT
14.CALCULATE NEXT
15.IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
16.MOVE NAME, STATUS, PARTITION, & PRIORITY
17.ENDIF
18.IF NEXT IS THE MANAGER, THEN
19.ENDIF
20.SET CURRENT AS TOP
21.ENDIF
22.SET CURRENT TO NEXT
23.ENDDO
24.IF TOP OR ZERO, THEN
25.SET TOP TO CURRENT
26.MOVE TOP TO MOVE NAME, PARTITION, & PRIORITY
27.SET STATUS TO 'IN USE' OCTAL 17
28.ENDIF
29.CALL SLIBX - ENABLE
30.WRITE FIRST SET OF HEADERS
31.WRITE TOP AS REPORT DATA
32.WRITE REPORT LINE
33.SET MANAGER AS REPORT DATA
34.WRITE REPORT LINE
35.SET EXECUTIVE AS REPORT DATA
36.WRITE REPORT LINE
37.SET CURRENT AS REPORT DATA
38.WRITE REPORT LINE
39.WRITE INTERMEDIATE HEADERS
40.DO UNTIL MAX ENTRIES OR NO MORE DATA
41.WRITE REPORT LINE
42.SET NEXT REPORT DATA
43.ENDDO
44.END XSTAT
45.SAMPLE REPORT: DFS STATUS FOR LU 10 HH:MM:SS 360
46.FD NAME PRIOR PARTY STATUS
47.TOP AT- PROC 922 4 GENERAL WAIT
48.MANAGER- XMGNN 40 3 GENERAL WAIT
49.EXECUTIVE- XEXHN 60 3 GENERAL WAIT
50.CURRENT AT- PROC 11311 4 GENERAL WAIT
51.BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
52.PROC 32767 6 DISC ALLOCATE SUSPEND
53.PROC 11311 4 GENERAL WAIT
54.PROC 2060 3 GENERAL WAIT
55.PROCB 845 6 GENERAL WAIT
BEGIN XAKILL
  1. TERMINATE CURRENT FDS FUNCTION
  2. IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
  3. ON NEXT RETURN VIA A PAN
  4. IF THE EXEC IS ACTIVE-DO NOTHING
  5. IF A PROCESSOR IS ACTIVE- USE RTE MESS TO OFF THE PROCESSOR
  6. SET MANAGER'S ID ADDRESS(STMG)
  7. IF STATUS OF MANAGER IS NOT WAIT, THEN
  8. SET TERMINATE FLAG IN STB-ENTRY
  9. WRITE "***XA05 FDS MANAGER SIGNAL TO TERMINATE SEQUENCE"
  10. ELSE
  11. IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
  12. WRITE "***XA06 FDS EXECUTIVE ACTIVE; NO ACTION TAKEN"
  13. ELSE
  14. PERFORM XA07M(CURRENT) FIND BOTTOM AT
  15. IF BOTTOM AT IS D.RTR OR SHP THEN
  16. WRITE "**XA08 MANAGER IS WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
  17. EXIT XAKIL
  18. ELSE
  19. IF RETURNED BOTTOM IS MANAGER THEN
  20. IF MANAGER IS NOT WAITING ON A PROGRAM THEN
  21. WRITE "**XA09 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
  22. EXIT XAKIL
  23. ELSE
  24. SET RETURN PARAMETER TO PROCESSOR ABENDED
  25. INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
  26. CALL BLIST TO REACTIVATE MANAGER
  27. ENDIF
  28. ENDIF
  29. WRITE "**XA10 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT."
  30. IF RETURNED BOTTOM WAS NOT MANAGER THEN
  31. SET NAME IN 'OFF' COMMAND
  32. CALL MESS TO 'OFF' THE PROCESSOR
  33. ENDIF
  34. ENDIF
  35. ENDIF
  36. ENDIF
  37. END
  38. END XAKILL
179 1 BEGIN XADYN
180 2 DO MILL TUPPER FIND BOTTOM AT
181 3 DO MILL TUPPER CHECK SMALL OCTAL
182 4 END SET TUPPER AS CURRENT
183 5 END SET TUPPER AS CURRENT
184 6 END SET TUPPER AS CURRENT
185 7 END SET TUPPER AS CURRENT
186 8 END SET TUPPER AS CURRENT

3-11
BEGIN XCNFG

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

REPRODUCIBILITY OF THE ORIgINAL PAGE IS POOR
1 BEGIN XCON
2 * SIGN ON A USER TO FDS
3 LO \U NTIL VALID USER ID (P3)
4 IF ID NOT A - 2, THEN
5 WRITE 'XCOG ENTER VALID ID (A - Z)'
6 HEAD RESPONSE
7 ENDIF
8 ENDIF
9 IF USER ID IS BEING USED, THEN
10 WRITE '**XCOG LU 'LU' IS CURRENTLY USING ID 'ID' SIGN ON REJECTED'
11 EXIT:XCGA
12 EDFIF
13 IF FDS RESOURCE NOT DEFINED, THEN
14 CALL RNAG (GLOBAL ALLOCATE, LOCAL SET)
15 ELSE
16 CALL RNAG (LOCAL SET)
17 EDFIF
18 IF NUMBER SIGNED ON (STRAC) .EQ. MAXIMUM USERS (STMN) THEN
19 ISSUE MESSAGE '**XCOG FDS CURRENTLY AT MAX USERS'
20 ELSE
21 DO FOR STMN (NUMBER OF FDS ENTRIES)
22 IF ENTRY'S LU (ESTBLU) .EQ. REQUESTING LU(P1) THEN
23 ISSUE MESSAGE '**XCOG 'LU' IS ALREADY SIGNED ON TO FDS'
24 EXIT:XCGA
25 ELSE
26 IF THIS ENTRY IS AVAILABLE, THEN
27 SET AS CURRENT-ENTRY-ADDRESS
28 EDFIF
29 ELSE
30 ENDIF
31 ENDIF
32 BECOME PRIVILEGED & DISABLED
33 CALL SLCR
34 STARTSEARCH WHILE NUMBER-FOUND < MAX NUMBER-NEEDED
35 SEARCH ID-SEGMENTS USING KEYID(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 XNGR NOT FOUND AND THIS ID .EQ. XNGR, THEN
41 SET ID ADDRESS OF XNGR
42 INCREMENT NUMBER-FOUND
43 ELSE
44 IF XATH NOT FOUND AND THIS ID .EQ. XATH, THEN
45 SET ID ADDRESS OF XATH
46 INCREMENT NUMBER-FOUND
47 ELSE
48 IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
49 SET ID ADDRESS OF FIRST-BLANK
50 INCREMENT NUMBER-FOUND
51 ELSE
52 IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
53 SET ID ADDRESS OF SECOND-BLANK
54 INCREMENT COUNT
55 ENDFIF
56 ENDFIF
57 ENDFIF
58 ENDFIF
59 EXIT: THERE ARE NO MORE IDS
CALL BLBRX    ENABLE
ISSUE MESSAGES "**XCO? CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP

BUILD ENTRY IN YSTD

* SET LU IN Q STBLU

SET LU IN ASCII INTO STBLA

SET USER'S ID INTO STBID

SET ADDRESS OF FIRST-BLANK INTO STBAR

SET ADDRESS OF SECOND-BLANK INTO STBAR

INCREMENT ACTIVE COUNT(STBAR)

BUILD XPGMN & XEXMN

MOVE PRIORITY THRU DISC ADDRESS FROM XMDR TO FIRST-BLANK

TURN ON TH BIT

SET NAME TO XPGMN

MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK

TURN ON TH BIT

SET NAME TO XEXMN

LINK ATTENTION FUNCTION TO THE USER

IF LU .NE. 1, THEN

DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE

CALCULATE ERT OVERLAY

SAVE ERT VALUE IN STBAR

SET ID ADDRESS OF XATTN INTO ERT

ENDIF

SET INPUT PARNs INTO ID OF XMGMN

SCHEDULE XMGMN VIA BLIST

CALL BLIST

CALL BLBRX    ENABLE

IF FDS HAS A FATHER, THEN

CALL MESS 'OFF,FATHER'

ENDIF

ISSUE MESSAGES "**XCO1 LU "HR" SIGNED ON TO FDS"

ENDXSEARCH

:XCTA

CALL NWOR (LOCAL CLEAR)

ENDIF

CALL EXEC TERMINATE

END XCON
FORTAN CALLING PROCEDURE

CALL XECLS (XDCLD)

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

LIST

INPUT

X COMMON - MASTSA (BITS 10-13 CONTAIN A 0 INDEX INTO A LIST OF
DIRECTIVES)

INTERNAL VARIABLES

LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

NOTES

USES .ENTR, XDLST, XERTM

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

BEGIN XDCLD
EXTRACT DIRECTIVE INDEX FROM MASTSA
CASE (LIST-) INDEX
:LIST: CALL XDLST
ENDCASE
CALL XERTM TO RETURN FROM SEGMENT
END XDCLD
40 1 #00  FORTRAN CALLING PROCEDURE
41 1 #00  CALL XELS (XCLF)
42 1 #00  
43 1 #01  
44 1 #01  XCLF DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
45 1 #01  REQUESTED AND CALLS THE APPROPRIATE HANDLER FUNCTION
46 1 #01  
47 1 #01  TOC
48 1 #01  SAVE
49 1 #01  RECALL
50 1 #01  DELETE
51 1 #01  RENAME
52 1 #01  COPY
53 1 #01  CLEAR
54 1 #01  OFF
55 1 #01  
56 1 #02  INPUT
57 1 #02  XE COMMON - MASSTA (BITS 10-13 CONTAIN A 1-8 INDEX INTO A LIST OF
58 1 #02  DIRECTIVES)
59 1 #02  
60 1 #02  
61 1 #04  INTERNAL VARIABLES
62 1 #04  LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
63 1 #04  
64 1 #04  
65 1 #05  NOTES
66 1 #05  USES XTOC, XSAVE, XRECA, XDELE, XOFF, XDREC, XDREN, XDSAY,
67 1 #05  XTOC, XRECA
68 1 #05  
69 1 #05  XCLF IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
70 1 #05  CONTAINING THE REFERENCED DIRECTIVES
71 1 #05  
72 1 #05  
73 1 #05  
74 1 #05  
75 1 #05  
76 1 #05  
77 1 BEGIN XCLF
78 2 XTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
79 2 CASE (XTOC, XSAVE, XRECA, XDELE, XDREN, XDSAY, XTOC, XRECA) INDEX
80 3  :TOC: CALL XTOC
81 3  :SAVE: CALL XSAVE
82 3  :RECA: CALL XDREC
83 3  :DELE: CALL XDELE
84 3  :REN: CALL XDREN
85 3  :COPY: CALL XDREC
86 3  :CLEA: CALL XCLF
87 3  :OFF: CALL XOFF
88 2 END CASE
89 2 CALL XRECA TO RETURN FROM SEGMENT
90 1 END XCLF
1  *DO  FORTRAN CALLING PROCEDURE
2  *DO  CALL XELEDS (XECU)
3  *DO
4  **********
5  *DO
6  XECU DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
7  *DO  REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
8  *DO  STORE
9  *DO  RESTORE
10  *DO  UNLOAD
11  *DO  LOAD
12  *DO  BATCH
13  *DO
14  **********
15  *DO  INPUT
16  *DO  XE COMMON - MASSTA (BITs 10-13 CONTAIN A 9-13 INDEX INTO A LIST OF
17  *DO  DIRECTIVES)
18  *DO
19  **********
20  *DO  INTERNAL VARIABLES
21  *DO  LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
22  *DO
23  **********
24  *DO  NOTES
25  *DO  USES .ENTR, XDSTO, XDRES, XDUNL, XDLOA, XDBAT, XERN
26  *DO  XDCLU IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
27  *DO  CONTAINING THE REFERENCED DIRECTIVES
28  *DO
29  **********
30  *DO
31  *DO
32  *DO
33  *DO
34  *DO
35  *DO
36  *DO
37  *DO
38  *DO
39  *DO
40  *DO
41  *DO
42  *DO
43  *DO
44  *DO
45  *DO
46  *DO
47  *DO
48  *DO
49  *DO
50  *DO
51  *DO
52  *DO
53  *DO
54  *DO
55  *DO
56  *DO
57  *DO
58  *DO
59  *DO
60  *DO
61  *DO
62  *DO
63  *DO
64  *DO
65  *DO
66  *DO
67  *DO
68  *DO
69  *DO
70  *DO
71  *DO
72  *DO
73  *DO
74  *DO
75  *DO
76  *DO
77  *DO
78  *DO
79  *DO
80  *DO
81  *DO
82  *DO
83  *DO
84  *DO
85  *DO
86  *DO
87  *DO
88  *DO
89  *DO
90  *DO
91  *DO
92  *DO
93  *DO
94  *DO
95  *DO
96  *DO
97  *DO
98  *DO
99  *DO
100 *DO
101 *DO
102 *DO
103 *DO
104 *DO
105 *DO
106 *DO
107 *DO
108 *DO
109 *DO
110 *DO
111 *DO
112 *DO
113 *DO
114 *DO
115 *DO
116 *DO
117 *DO
118 *DO
119 *DO
120 *DO
121 *DO
122 *DO
123 *DO
124 *DO
125 *DO
126 *DO
127 1 BEGIN XDCLU
128 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7
129 3 CASE (:STOR:, :REST:, :UNLO:, :LOAD:, :BATC:) INDEX
130 3 :STOR: CALL XDSTO
131 3 :REST: CALL XDRES
132 3 :UNLO: CALL XDUNL
133 3 :LOAD: CALL XDLOA
134 3 :BATC: CALL XDBAT
135 2 ENDCASE
136 1 END XDCLU
FORTRAN CALLING PROCEDURE

CALL XDCLE

**FUNCTION**

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

**INPUT**

XCOMMON - CARTAG, FLAG, LU

**OUTPUT**

XCOMMON - REBUFF, REOPTR, COMBUF-SCRATCH

**LOCAL VARIABLES**

AVAH - AWA HEADER AND TOC RECEIVED FROM MANAGER

**NOTES**

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 XCODE
189 2 RETRIEVE TOC
190 2 BUILD REQUEST TO CLEAR AW
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 2 BUILD REQUEST TO REALLOCATE TABLE
195 2 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
196 2 BUILD REQUEST TO STORE TABLE INTO AW FROM SAM
197 2 ENDO
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 ENDO
203 2 BUILD REQUEST TO TERMINATE LIST
204 2 DO UNTIL END OF DDRE CHAIN (CHAIN 3)
205 2 IF CHAIN POINTS BEYOND TOC BUFFER
206 2 THEN
207 4 OUTPUT XD3 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
208 4 EXIT PURGE LOOP
209 2 ENDF
210 2 PURGE FILE
211 2 ENDO
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 ENDO
216 1 EXIT XCODE
217 2 :ERR23:
218 2 DO FROM FAILING REQUEST TO END OF LIST
219 3 IF REQUEST TO STORE
220 4 THEN
221 4 READ SAM TO FREE BUFFER AND CLASS NUMBER
222 4 ENDF
223 2 ENDO
224 2 DO UNTIL END OF DATABASE FILE CHAIN (CHAIN 8)
225 3 IF FILE IS UTD (TYPE 1)
226 3 THEN
227 4 CALL PURGE TO DELETE FILE
228 4 ENDF
229 2 ENDO
230 2 PURGE ALL UTD 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 ENDO
237 1 EXIT XCODE WITH CLEAR FAILURE MESSAGE
238 1 END XCODE
FORTRAN CALLING PROCEDURE

CALL XDCOP

BEGIN

XDCOP PROCESSES A USER REQUEST TO COPY A SEQUENCE TABLE.

INTERFACE TABLE, DATA ELEMENT, DDE, UDD, OR FOR. ONLY
THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S AWA CAN BE COPIED.

BEGIN

COMMON XE - CARTAG, COMBUF, QGAL, REQPTR, TOKENS

COMMON XE - COMPRTR, REBUF

COMMON XS - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE)
(2) DTYPE: TYPE CODE FOR DATA BASE FILES
(3) I: INDEX
(4) IDCB: EXEC BUFFER
(148) IERR: FILE MANAGER ERROR RETURN
(149) IMSG: ERROR MESSAGE NUMBERS
(150) FNNAME: NEW FMGR FILE NAME
(153) MNMPTR: POINTER TO NEW NAME IN COMBUF
(154) ONAME: OLD FMGR FILE NAME
(157) OMMPTR: POINTER TO OLD NAME IN COMBUF
(158) TOCET: TOC ENTRY FOR DATA BASE RENAME
(166) EOF: EOF RETURN FROM FMGR

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

COMMON XG - (124A) FNTYPE: FMGR FILE TYPE
(124B) FNAME: NEW FMGR FILE NAME
(124C) PANS: SECOND BYTE OF REQPTR
(124D) PARS: SECOND BYTE OF REQPTR
(124E) PRN: PREFIX FOR FILE NAME
(124F) SCRIT: FILE SECURITY CODE (0 OR 88)
(124G) SIZM: FILE SIZE IN BLOCKS
(125) IDCBY: EXEC BUFFER

END
BEGIN XDCOP
EXE TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
SAVE POINTER TO CURRENT NAME
IF NEXT TOKEN IS HYPHEN, THEN
DECODE CLASS NAME
EXIT TO :CLASS: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,B)
ELSE
SET CLASS TO DATA ELEMENT
ENDIF
EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
SAVE POINTER TO NEW NAME
EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT END-OF-MESSAGE
IF CLASS IS DATA BASE OR ORDER, THEN
EXIT IF NEW NAME IS MORE THAN 4 CHARACTERS :NAMEERR:
IF CLASS IS DATA BASE, THEN
EXIT TO :NAMEERR: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
ENDIF
BUILD AVAL MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
BUILD AVAL MANAGER REQUEST TO VERIFY NEW NAME
CALL XREN
EXIT TO :TOCERR: IF RETURN INDICATES ERROR ON FIRST REQUEST
EXIT TO :TOCERR: IF NO ERROR RETURNED ON SECOND REQUEST
CALL EXEC TO GET TOC ENTRY
IF CLASS IS DATA BASE THEN
IF TYPE IS POB, THEN
CALL XDBA TO ADD NEW POB TO LOG FILE (XPDB)
EXIT TO :FILERR: IF FMGR ERROR RETURNED
EXIT TO :TOCERR: IF NEW NAME IS DUPLICATE
EXIT TO :MAXERR: IF POB MAX IS EXCEEDED
SET FILE PREFIX TO RIGHT BRACKET
ELSE
EXIT TO :INVLD: IF FILE IS NOT
SET FILE PREFIX TO *
ENDIF
SET FMGR FILE TYPE TO 1
SET SECURITY CODE TO 88
ELSE
SET FILE PREFIX TO /
SET SECURITY CODE = 0
SET FMGR FILE TYPE FROM TOC ENTRY
IF FILE IS TYPE 2, THEN
STORE RECORD LENGTH FROM TOC ENTRY
ENDIF
ENDIF
CALL XRFNM TO FORMAT NEW FILE NAME
CALL CREATE FOR NEW FILE
EXIT TO :FILERR: IF FMGR ERROR RETURNED
CALL XRFNM TO FORMAT OLD FILE NAME
CALL OPEN FOR OLD FILE
EXIT TO :FILERR: IF FMGR ERROR RETURNED
EXIT TO :TYPERR: IF TYPE IS NOT SAME AS IN TOC
DO UNTIL END-OF-FILE IS READ ON OLD FILE
CALL READ FROM RECO RECORD FROM OLD FILE
EXIT IF ERROR DETECTED TO :FILERR:
CALL WRITE TO WRITE RECORD TO NEW FILE
EXIT IF ERROR DETECTED TO :FILERR:
END DO
EXIT TO :TOCERR: IF RETURN INDICATES ERROR ON SECOND REQUEST
CALL close FOR OLD FILE
CALL close FOR NEW FILE
BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY
EXIT TO :T OCERR: IF ERROR IS INDICATED
ELSE
BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
BUILD MANAGER REQUEST FOR DATA RETRIEVAL
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
CALL EXEC TO GET TOC ENTRY
BUILD MANAGER REQUEST TO ALLOCATE NEW TABLE
BUILD MANAGER REQUEST TO STORE TABLE
IF CLASS IS INTERFACE TABLE, THEN
CALL EXEC TO WRITE NEW TABLE NAME TO SAM
BUILD MANAGER REQUEST TO STORE NEW NAME IN TABLE
ENDIF
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
ENDIF
EXIT TO :TOCERR: IF ERROR IS INDICATED
EXIT XDCOP

:SYNTAX: CALL XRM  - "SYNTAX ERROR ...
:CLASER: CALL XRM  - "INVALID CLASS DESIGNATOR ...
:NAMERR: CALL XRM  - "NEW NAME IS INVALID ...
:MAXERR: CALL XRM  - "AUTHORIZED LIMIT ...
:INVLIB: CALL XRM  - "NOB CANNOT BE ...
:FILERR: CALL XRM  - "FILE ACCESS ERROR ...
:
:
:END:
IF PBO HAS BEEN LOGGED IN XPDQ, THEN
CALL XDQOD TO DELETE PBO FROM XPDQ
ENDIF
IF A NEW FILE HAS BEEN BUILT, THEN
PURGE NEW FILE
CLOSE OLD FILE
ENDIF
END XDCOP
394 1.00  FORTRAN CALLING PROCEDURES
395 1.00
396 1.00
397 1.00  CALL XDDBA(NAME, IERR, ISIZE)
398 1.00  CALL XDDBY(NAME, IERR, ISIZE)
399 1.00  CALL XDBD(NAME, IERR, IERR, IERR)
400 1.00  ********
401 1.00
402 1.00  XDDBA, XDDBY, AND XDBD PROVIDE THE CAPABILITY TO ADD AN MDB OR
403 1.00  PDB TO MDB/PDB DIRECTORY (XPDB). TO VERIFY THAT AN MDB OR PDB IS
404 1.00  LOGGED IN XPDB AND RETRIEVE THE SIZE OF THAT MDB/PDB, AND TO
405 1.00  DELETE AN MDB OR PDB FROM XPDB, RESPECTIVELY
406 1.00  ******
407 1.00
408 1.00  INPUT
409 1.00  NAME - 2 WORD INTEGER ARRAY CONTAINING THE UNQUALIFIED ASCII
410 1.00  MDB/PDB NAME
411 1.00  ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
412 1.00  XDDBA ONLY)
413 1.00  ******
414 1.00
415 1.00  OUTPUT
416 1.00  IERR - INTEGER WORD CONTAINING RETURN CODE
417 1.00  =0 FAGE ERROR CODE RETURNED FOR XPDB ACCESS
418 1.00  =0 NO ERRORS
419 1.00  =1 DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE
420 1.00  =2 PDB/PDB LIMIT EXCEEDED
421 1.00  =3 ISIZE - INTEGER WORD CONTAINING THE MDB/PDB SIZE IN BLOCKS (FOR
422 1.00  XDDBY ONLY)
423 1.00  ******
424 1.00
425 1.00  COMMON USED
426 1.00  XE - EQU
427 1.00  CARING
428 1.00  ******
429 1.00
430 1.00  NOTES
431 1.00  XRLK, XRLK, .ENBR, OPEN, CLOSE, READF, AND WRITF ARE USED
432 1.00  WHEN WORKING WITH MDB'S, QUAL SHOULD BE SET TO 77 OCTAL
433 1.00  ******
445 1 BEGIN XDDBA
446 2: STORE RETURN ADDRESS
447 2: CALL .ENTRY TO SET UP CALLING ARGUMENTS
448 2: CALL XLCK FOR EXCLUSIVE USE OF XDDBA
449 2: CALL OPEN FOR EXCLUSIVE USE OF XDDBA
450 1: EXIT TO :FILERR: IF ERROR RETURNED
451 2: COMPUTE RECORD NUMBER FOR USER'S DIRECTORY (QUAL=77)\(2+1\)
452 2: CALL READ FOR RECORD COMPUTED
453 1: EXIT TO :FILERR: IF ERROR RETURNED
454 2: DETERMINE PART OF RECORD TO BE USED
455 2: IF REQUEST IS FOR ADD, THEN
456 2: EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
457 2: START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
458 2: EXIT TO :NAMEERR: IF ENTRY NAME MATCHES PARAMETER NAME
459 2: ENDLOOP
460 2: ENDSERCH
461 2: STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
462 2: INCREMENT # OF CURRENT ENTRIES
463 2: ELSE
464 2: START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
465 2: EXIT IF ENTRY NAME MATCHES PARAMETER NAME
466 2: ENDLOOP
467 2: EXIT TO :NAMEERR:
468 2: ENDSERCH
469 2: IF REQUEST IS FOR VERIFY, THEN
470 2: STORE WORD 3 OF ENTRY IN ISIZE
471 2: ELSE
472 2: REPLACE ENTRY WITH LAST ENTRY
473 2: STORE ENDBS IN LAST ENTRY
474 2: DECREMENT # OF CURRENT ENTRIES
475 2: ENDIF
476 2: ENDF
477 2: CALL WRIT TO WRITE RECORD TO XDDBA
478 1: EXIT TO :RETURN:
479 2: :NAMEERR: SET IERR = 1 AND EXIT TO :RETURN:
480 2: :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
481 2: :FILERR: SET IERR = FMDA ERROR CODE
482 2: :RETURN: CALL CLOSE FOR XDDBA
483 2: CALL XLCK TO RETURN RESOURCE #
484 1 END XDDBA
BEGIN XDLE

DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED

ERERE: IF COMMA IS NOT NEXT LEXICAL ELEMENT :ERROR:

IF CLASS DESIGNATOR IS SPECIFIED, THEN

SET REQUESTED CLASS APPROPRIATELY (B, S, I, D, OR F)

ELSE

SET REQUESTED CLASS TO BE (0)

ENDIF

IF DATA BASE TO BE DELETED, THEN

BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY

ENDIF

IF ELEMENT IS NOT A MASTER DATA BASE, THEN

IF ELEMENT IS A PERSONAL DATA BASE, THEN

CALL XDODD TO DELETE THIS PDD FROM XPDD

ENDIF

IF ERROR IS RETURNED, THEN

CALL XRMG - "FILE ACCESS ERROR 0... XPDD"

SET ERROR FLAG

ENDIF

ENDIF

BUILD AND ISSUE AWA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED

IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN

CALL XRMG - "XXXXX NOT FOUND"

SET ERROR FLAG

ENDIF

IF CLASS IS DATA BASE (C), OR

CLASS IS DDE (F), THEN

IF ERROR FLAG IS NOT SET, THEN

CALL XRMG TO CONSTRUCT FILE NAME

ISSUE RTE PURGE FOR THE FILE

IF RETURN CODE FROM PURGE, THEN

CALL XRMG - "FILER ERROR XXXXXXX"

ENDIF

ENDIF

ELSE

CALL XRMG - "... IS A HDB. NOT DELETED."

ENDIF

ENDO

END TO :RETURN:

:ERROR: CALL XRMG - "SYNTAX ERROR"

:RETURN:

1 END XDLE
```
647 1 BEGIN XDLIS
648 2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
649 3 TURN ERFGL OFF
650 4 :RTN1:
651 5 DO UNTIL EOS IS SENSED IN COMBUS
652 6 ERREXIT IF TOKEN IS NOT "NAME" TO :ERR1:
653 7 SAVE INDEX TO NAME FIELD
654 8 INCREMENT TO NEXT TOKEN
655 9 IF TOKEN IS A HYPHEN THEN
656 10 ERREXIT IF NEXT TOKEN IS NOT "NAME" TO :ERR1:
657 11 INCREMENT TO NEXT TOKEN
658 12 DECODE CLASS NAME (I, S, D, F)
659 13 ERREXIT IF CLASS SPECIFIED IS NOT VALID TO :ERR1:
660 14 SET CLASS TO CLASS SPECIFIED
661 15 ELSE
662 16 SET CLASS TO DATA ELEMENT
663 17 ENDIF
664 18 IF XDLIS CALLED FROM STORE THEN
665 19 ERREXIT IF PREFIX IS DOUBLE EXCLAMATION TO :ERR2:
666 20 ENDIF
667 21 ERREXIT IF NAME/CLASS ENTRY NOT FOUND IN TOC TO :ERR2:
668 22 CALL ARSET TO TURN STORE/RESTORE BIT ON
669 23 INCREMENT TOTAL SIZE BY SIZE OF THIS ELEMENT
670 24 ENDDO
671 25 :RTN2:
672 26 IF ERROR FLAG IS ON THEN
673 27 CALL XCOM TO REPROMPT USER TO CONTINUE
674 28 ERREXIT IF RESPONSE IS X TO :ERR3:
675 29 ENDDO
676 30 ENDDO
677 31 EXIT XDLIS
678 32 :ERR1:
679 33 SET ERROR FLAG ON
680 34 CALL XRMSG TO DISPLAY SYNTAX ERROR
681 35 GO TO :RTM2:
682 36 :ERR2:
683 37 IF ERROR FLAG IS OFF THEN
684 38 TURN ERROR FLAG ON
685 39 CALL XRMSG TO DISPLAY NOT STORED/RESTORED MESSAGE
686 40 ENDF
687 41 CALL XECX TO DISPLAY ELEMENT NAME
688 42 GO TO :RTM1:
689 43 :ERR3:
690 44 SET ABFLG TO ABORT STORE/RESTORE OPERATION
691 45 END XDLIS
```
1 BEGIN XDLST
2 IF DEVICE ID FIELD SPECIFIED, THEN
3 SET LU FOR LISTING AS INDICATED ON INRECTIVE
4 ELSE
5 SET LU FOR LISTING TO BE TERMINAL LU
6 ENDIF
7 DO UNTIL END-OF-STATEMENT IS REACHED
8 ERREXIT IF COMMA IS NOT SPECIFIED :ERROR9:
9 ERREXIT IF A NAME DOES NOT FOLLOW THE COMMA :ERROR9:
10 RETAIN NAME FOR XREX CALL
11 IF A CLASS DESIGNATOR IS SPECIFIED, THEN
12 SET CLASS (I, S, OR D) FOR XREX CALL
13 ELSE
14 USE DATA (O) CLASS IN XREX CALL
15 ENDIF
16 CALL XREX TO RETRIEVE THIS TABLE OR DATA ELEMENT
17 ERREXIT IF NOT FOUND :ERROR9:
18 ERREXIT IF AWA SPACE NOT AVAILABLE FOR TABLE IN DNA :ERR11:
19 CALL EXEC TO PERFORM CLASS READ OF DATA OR TABLE INTO
20 BOTTOM OF WORKING BUFFER
21 IF INTERFACE TABLE TO BE LISTED, THEN
22 CALL XMOV TO MOVE CHARACTERISTICS TO TOP OF WORKING BUFFER
23 READ SHORT PROMPTS FOR THIS PROCESSOR INTO WORKING BUFFER
24 CALL XEXIT TO INITIALIZE LITERAL ENTRIES
25 INITIALIZE INTERFACE TABLE EDITOR COMMON TO USE ITS LIST RTN.
26 INITIALIZE 'LSTFLG' TO INDICATE ENTRIE TABLE TO BE LISTED
27 CALL XILST TO LIST THE INTERFACE TABLE
28 ELSE
29 IF SEQUENCE TABLE TO BE LISTED, THEN
30 CALL XMOV TO MOVE TABLE TO TOP OF WORKING BUFFER
31 CALL XILST TO LIST SEQUENCE TABLE
32 ELSE
33 INITIALIZE INTERFACE TABLE EDITOR COMMON FOR USE OF ITS LIST RTN.
34 SET 'LSTFLG' TO INDICATE ONLY 1 DATA ELEMENT BEING LISTED
35 INITIALIZE PRINT BUFFER WITH NAME OF ELEMENT
36 CALL XILSD TO LIST THE DATA
37 ENDIF
38 ENDIF
39 INCREMENT TO NEXT TOKEN IN OPERAND LIST OF THE DIRECTIVE IMAGE
40 :ERROR10: CALL XRMSG -- '... NOT FOUND'
41 :ERR11: CALL XRMSG -- '... CANNOT BE MOVE FROM DNA TO AWA'
42 ENDO
43 EXIT TO :RETURN:
44 :ERROR9: CALL XRMSG -- 'SYNTAX ERROR'
45 :RETURN:
46 END XDLST
783 1 CD************
784 1 CD0        FORTRAN CALLING PROCEEDURE
785 1 CD0        CALL XDOFF
786 1 CD0        
787 1 CD0        
788 1 CD0        
789 1 CD0        
790 1 CD1        XDOFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
791 1 CD1        
792 1 CD1        DELETES ALL DRDE AND UTDB FILES LOGGED IN THE ANA,
793 1 CD1        PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
794 1 CD1        RETURNS NORMAL PATHS TO THE FDS MANAGER AND TERMINATES
795 1 CD1        
796 1 CD1        
797 1 CD1        
798 1 CD2        INPUT
799 1 CD2        
800 1 CD2        COMMON X E - L,U,FLAGS,QUAL,REBUF
801 1 CD2        
802 1 CD2        COMMON X B - ORG = ORIGIN ADDRESS OF ANA. USED TO CALCULATE
803 1 CD2        INDICES INTO 'ANA' FROM ADDRESS POINTERS
804 1 CD2        OF TOC ENTRIES
805 1 CD2        ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND
806 1 CD2        TOC RETRIEVED VIA XREQ
807 1 CD2        
808 1 CD2        
809 1 CD2        
810 1 CD4        INTERNAL VARIABLES
811 1 CD4        
812 1 CD4        COMMON X S - POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC
813 1 CD4        
814 1 CD4        ENTRY. MOST SIGNIFICANT (BIT 15) BIT
815 1 CD4        SET TO INDICATE END-OF-CHAIN
816 1 CD4        INDEX = VALUE COMPUTED FROM POINTER TO BE
817 1 CD4        FORTRAN INDEX INTO 'ANA' FOR NEXT TOC
818 1 CD4        ENTRY
819 1 CD4        
820 1 CD5        NOTES
821 1 CD5        
822 1 CD5        ROUTINE USED - EXEC, PURGE, XDSTA, XPIT, XREQ, XREXT, XR16,
823 1 CD5        XRM0, XRM5, XRM5F, XRPCX, XRCST, XRUPK, XCMT,
824 1 CD5        XDBUG, XYABN
FORTRAN CALLING SEQUENCE:

CALL XDRDE (DATBUF, DBDCB)

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

INPUTS FROM CALLING SEQUENCE:

DATBUF - (INTEGER, 1400 WORDS) BUFFER USED TO READ IN RECORDS
OF DATA FROM DATA BASE FILE
DBDCB - (INTEGER, 144 WORDS) OPEN DATA BASE FILE DBO

INPUTS FROM XB COMMON:

XB(154) ALLFLG, XB(177) ENDBLK, XB(182) TOTSZ,
XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

DATBUF, DBDCB

OUTPUTS IN XB COMMON

XB(151) ABFLG, XB(152) EREFL, XB(165) TOCHSD,
XB(169) TOPLBLK, XB(170) ENDBLK, XB(171) TOCHNR,
XB(201) NOTOC

INTERNAL XB COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) EREFL - ERROR MESSAGE FLAG
XB(154) ALLFLG - RESTORE ALL UDBL FLAG
XB(155) DBFLG - DEBUG FLAG
XB(155) DEBUG - DEBUG FLAG
XB(159) FNAME - DB FILE NAME
XB(160) DATBLK - BLOCK # WHERE DATA ITEM BEGINS
XB(161) DATINDEX - WORD INDEX INTO DATBUF WHERE DATA BEGINS
XB(163) EERR - ERROR FLAG FOR FMGR CALLS
XB(165) TOCHNR - # BLOCKS TO READ
XB(166) NOBLKS - # BLOCKS TO READ
XB(167) LEN - # WORDS OF DATA TO MOVE
XB(168) DATEND - BLOCK # WHERE DATA ITEM ENDS
XB(169) TOPBLK - BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBLK - BLOCK # OF LAST BLOCK READ
XB(171) TOCHNR - INDEX INTO TCB OF CURRENT DATA ITEM
XB(175) IL - # WORDS OF DATA TO READ
XB(176) TCBTENT - DATA BASE TCB ENTRY
XB(201) NOTOC - # TCB ENTRIES IN TCBUF
XB(201) TCBUF - TCB 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-ORDE FILES TO PROCESS
5 IF ALLFLG IS ZERO OR IF STORE/RESTORE BIT IS ON THEN
6 IF DATA IS NOT CURRENTLY IN DATBUF THEN
7 CALL READ TO READ 1 BUFFER BEGINNING WITH DATBLK FOR THIS ELEMENT
8 ERREXIT IF READ ERROR TO :ERR1:
9 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
10 ELSE, DATA BEGINS IN DATBUF
11 IF DATA DOES NOT END IN DATBUF THEN
12 CALL XMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
13 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
14 CALL READY TO READ ENOUGH TO FILL DATBUF
15 ERREXIT IF READ ERROR TO :ERR1:
16 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
17 ENDIF
18 ENDIF
19 BUILD DMA REQUEST TO ALLOCATE AND STORE DATA
20 CALL EXEC TO WRITE DATA TO SAN
21 ERREXIT IF ERROR FROM EXEC TO ... 2:
22 IF DMA REQUEST BUFFER IS FULL THEN...
23 CALL XRDE TO ISSUE REQUEST
24 EXIT XRDE IF ERROR IN XRDE
25 ENDIF
26 ENDIF
27 END XRDE
28 EXIT XRDE
29 :ERR1:
30 CALL XR16 TO CONVERT ERROR CODE TO ASCII
31 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
32 GO TO :ERR3:
33 :ERR2:
34 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
35 :ERR3:
36 SET ABFLG TO SAY ABORT RESTORE
37 END XRDE
FORTRAN CALLING SEQUENCE:

CALL XRDFB(DATBUF, DBDBC)

XRDFB HANDLES THE RESTORING OF DRIE FILES FROM A MDB/UTDB FILE

INPUTS IN CALLING SEQUENCE:

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

DBDBC - (INTEGER, 144 WORDS) OPEN DATA BASE FILE BCB

INPUTS IN XR COMMON:

XU(152) ERFILE, XU(154) ALLFLG, XU(165) TOCBUS;
XU(169) TOBBLE, XB(170) ENDBUS, XB(171) TOCBUS,
XB(182) TOBBUS

OUTPUTS IN CALLING SEQUENCE:

DBDBC

OUTPUTS IN XR COMMON:

XB(151) ABFLG

INTERNAL XR COMMON USED:

XB(151) ABFLG - ABORT FLG
XB(152) ERFILE - ERROR MESSAGE FLG
XB(154) ALLFLG - RESTORE ALL UTDB FLG
XB(155) DEBUG - DEBUG FLG
XB(159) FINAN- OB FILE NAME
XB(162) DATBLK- BLOCK # WHERE DRIE BEGINS
XB(163) DATNOR- WORD INDEX INTO DATBUF WHERE DRIE BEGINS
XB(164) ERR- ERROR FLG FOR FNGR CALLS
XB(165) TOCBUS- # WORDS OF TOC IN TOCBUF (MOTOC #) + 8
XB(166) NOBLKS- # BLOCKS OF UTDB/DRIE TO READ/WRTE
XB(167) LEN - # WORDS OF DATA TO MOVE
XB(169) TOPBLK- BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBLK- BLOCK # OF LAST BLOCK READ
XB(171) TOCBUS- INDEX TO CURRENT TOC ENTRY
XB(172) DRIEFL- DRIE FILE NAME
XB(175) IL - # WORDS OF DATA TO READ/WRTE
XB(201) TOCBUS- TOC BUFFER

COMMON USED:

1 CD***************
BEGIN XDRDF
DO WHILE THERE ARE IOC ENTRIES TO PROCESS
   IF ALL FI IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
      CALL XOPEN TO CREATE THE Dede FILE NAME
   CALL CREAT TO CREATE THE Dede FILE
   IF THERE WAS A CREATE ERROR THEN
      IF ERROR FLAG IS OFF THEN
         SET ERROR FLAG ON
      CALL XRMSG TO DISPLAY MAIN MESSAGE
   ENDIF
   CALL XTRG TO CONVERT ERROR CODE TO ASCII
   CALL XREC TO DISPLAY Dede NAME AND ERROR CODE
   ENDIF
ELSE
   CALL CLOSE TO CLOSE FI
   IF XERR IF CLOSE ERROR TO :ERR1:
   CALL OPEN TO OPEN Dede FILE AS TYPE 1
   IF XERR IF OPEN ERROR TO :ERR1:
   DO UNTIL ALL BLOCKS OF Dede HAVE BEEN PROCESSED
      IF DATA FOR Dede IS IN DATUFR THEN
         CALL WRITE DATA TO Dede
      ELSE
         IF DATA FOR Dede IS IN DATUFR THEN
            CALL WRITE DATA TO Dede
         ELSE
            CALL READF TO READ NEXT BUFFER OF DATA
         ENDIF
      ENDIF
      SET INDICES INDICATING DATA IN BUFFER
   ENDIF
ENDIF
ENDDO
CALL CLOSE TO CLOSE Dede FILE
IF XERR IF CLOSE FAILED TO :ERR1:
BUILD AWA REQUEST TO ALLOCATE Dede IN AHA
IF AWA REQUEST BUFFER IS FULL THEN
CALL XOPEN TO MAKE REQUEST
EXIT XDRDF IF XERR ERROR
ENDIF
ENDIF
EXIT XDRDF
END
106       1 C**********
107       1 C      FORTRAN CALLING PROCEDURE
108       1 C
109       1 C
110       1 C      CALL XREC
111       1 C
112       1 C**********
113       1 C01  XREC PROCESSES THE RECALL DIRECTIVE. A UTOB IS CREATED AND
114       1 C01  THE CONTENTS OF THE SPECIFIED PDBG ARE COPIED TO IT.
115       1 C01
116       1 C01
117       1 C**********
118       1 C02  INPUT
119       1 C02
120       1 C02  COMMON XE - CARTAG, COMBUF, COMPTR, Flags, LV, TOKENS
121       1 C02
122       1 C02  FILES - )XIXX (PDG FILE SPECIFIED)
123       1 C02
124       1 C02
125       1 C**********
126       1 C03  OUTPUT
127       1 C03
128       1 C03  COMMON XE - REGBUF, REOPTR
129       1 C03
130       1 C03  FILES - )XIXX (UTOB FILE SPECIFIED)
131       1 C03
132       1 C03
133       1 C**********
134       1 C04  INTERNAL VARIABLES
135       1 C04
136       1 C04
137       1 C04  DCPDGB - DCB FOR THE PDG FILE; ALLOCATED IN X0 COMMON;
138       1 C04  CONTAINS 1552 WORD BUFFER USED TO READ THE PDG
139       1 C04  AND TO WRITE THE UTOB
140       1 C04  DCPDUB - DCB FOR THE UTOB FILE; ALLOCATED IS X0 COMMON
141       1 C04
142       1 C**********
1144 1 BEGIN XREC
1145 2 SET STATUS FLAG INDICATING PDD & UTD0 FILES NOT OPEN
1146 2 ERRO3 IF "-" IS NOT NEXT TOKEN :ERROR5:
1147 2 INCREMENT TO NEXT TOKEN
1148 2 ERRO3 IF NEXT TOKEN IS NOT 'NAME' :ERROR5:
1149 2 ERRO3 IF THIS NAME IS A CHARACTERS :ERROR5:
1150 2 ERRO3 IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR5:
1151 3 RETURN THIS NAME AS PDD
1152 3 RETURN CURRENT USER IN QUALIFIER
1153 3 INCREMENT TO NEXT TOKEN
1154 3 IF NEXT TOKEN IS "-", THEN
1155 3 INCREMENT TO NEXT TOKEN
1156 3 ERRO3 IF NEXT TOKEN IS NOT 'NAME' :ERROR5:
1157 3 ERRO3 IF 'NAME' IS GREATER THAN 1 CHARACTER :ERROR5:
1158 3 ERRO3 IF QUALIFIER < 'A' OR > 'Z' :ERROR5:
1159 3 SAVE IN AT (ZERO FIL.), RIGHT-JUSTIFIED FORMAT AS QUALIFIER
1160 3 INCREMENT TO NEXT TOKEN
1161 3 ENDF
1162 3 ERRO3 IF NEXT TOKEN IS NOT A 'NAME' :ERROR5:
1163 3 INCREMENT TO NEXT TOKEN
1164 3 ERRO3 IF NEXT TOKEN IS NOT A 'NAME' :ERROR5:
1165 3 ERRO3 IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR5:
1166 3 RETURN THIS NAME AS UTD0
1167 3 INCREMENT TO NEXT TOKEN
1168 3 ERRO3 IF NEXT TOKEN IS NOT EOS :ERROR5:
1169 3 BUILD AWA REQUEST TO VERIFY EXISTENCE OF UTD0
1170 3 CALL AWA TO PROCESS AWA REQUEST
1171 3 ERRO3 IF UTD0 DOES ALREADY EXIST :ERROR5:
1172 3 CALL AWA TO VERIFY PDD AND RETRIEVE SIZE
1173 3 ERRO3 IF PDD DOES NOT EXIST :ERROR5:
1174 3 ERRO3 IF PMR ERROR RETURNED :ERROR5:
1175 3 CALL XBUF TO CULK PDD FILE NAME
1176 3 CALL OPEN TO OPEN PDD FILE
1177 3 ERRO3 IF OPEN FAILED :ERROR5:
1178 3 SET STATUS FLAG INDICATING PDD FILE OPEN
1179 3 CALL XBUF TO PROCESS AWA REQUEST
1180 3 ERRO3 IF AWA REQUEST FAILS :ERROR5:
1181 3 SET STATUS FLAG TO INDICATE UTD0 FILE ALLOCATED IN AWA
1182 3 RESTORE CURRENT USER'S ID (QUALIFIER)
1183 3 CALL XBUF TO BUILD UTD0 FILE NAME
1184 3 CALL CREATE TO CREATE TYPE 1 UTD0 FILE
1185 3 ERRO3 IF CREATE FAILED :ERROR5:
1186 3 SET STATUS FLAG INDICATING UTD0 FILE NOW OPEN
1187 3 DO FOR EACH BUFFER OF DATA IN PDD FILE
1188 3 CALL READ TO READ 1 BUFFER FROM PDD FILE
1189 3 ERRO3 IF READ ERROR :ERROR5:
1190 3 CALL WRITE TO WRITE 1 BUFFER TO UTD0 FILE
1191 3 ERRO3 IF WRITE ERROR :ERROR5:
1192 3 ENDDO
1193 3 CALL CLOSE FOR PDD
1194 3 CALL CLOSE FOR UTD0
1195 1 EXIT XREC
1196 2 :ERROR5: ISSUE MESSAGE - "SYNTAX ERROR. EXTRANEOUS DATA"
1197 2 :ERROR5: ISSUE MESSAGE - "ILLEGAL UTD0 NAME (NOT FOUND OR TOO LONG)"
1198 2 :ERROR5: ISSUE
1200 2 :ERROR? ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERROR9 ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERROR6 ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERROR8 ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
1204 2 :ERROR9 ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERROR1 ISSUE MESSAGE - "AVA OVERFLOW - XXX NOT LOGGED"
1206 2 :ERROR2 ISSUE MESSAGE - "XXX ALREADY EXISTS"
1207 2 :ERROR4 ISSUE MESSAGE - "FILE ACCESS ERROR $-- XPD" /!
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 AWA, THEN 
1213 3 CALL XER TO DELETE UTDB FROM AWA
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

XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
XREC
*******
CD0 FORTRAN CALLING PROCEDURE
CD1 CALL XDREN
CD2 ********
CD1 XDREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE,
CD1 INTERFACE TABLE, DATA ELEMENT, DDE, UDOE, OR PDB. ONLY
CD1 THOSE TABLES OR ELEMENTS LOGGED IN THE USERS AWK ARE RENAMED.
CD1 ********
CD2 CO INPUT
CD2 COMMON XE - CARTAG, COMBUF, COMPTR, QUAL, RERPT, TOKENS
CD2 ********
CD5 COMMON XE - REBUF
CD5 ********
CD8 INTERNAL VARIABLES
CD8 COMMON XS - (2) DTYPE: TYPE CODE FOR DATA BASE FILES
CD8 (3) I: INDEX
CD8 (148) IERR: FILE MANAGER ERROR RETURN
CD8 (149) JMSG: ERROR MESSAGE NUMBERS
CD8 (150) MFNAME: NEW FGR FILE NAME
CD8 (153) MNPT: POINT TO NEW NAME IN COMBUF
CD8 (154) UFNAME: OLD FGR FILE NAME
CD8 (157) OMPT: POINT TO OLD NAME IN COMBUF
CD8 (158) TOCENTRY: TOC ENTRY FOR DATA BASE RENAME
CD8 (166) TSTCLS: DATA CLASS CODE (STORED IN LEFT BYTE)
CD8 COMMON XB - (201) IDC8: EXEC BUFFER
CD8 *******
CD8 EXEC - Routines Used -
CD8 EXEC JANG KCVT
CD8 MANP XDOBA
CD8 XDOBD XREG
CD8 XRMOV XRMSG
CD8 XRDFN XRFSR
CD8 *******
BEGIN XGREM
EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
SAVE POINTER TO CURRENT NAME
INCREMENT TO NEXT TOKEN
IF TOKEN IS A HYPHEN, THEN
EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
DECODE CLASS NAME
EXIT TO :CLSER: IF CLASS SPECIFIED IS NOT VALID (S,F,D,B)
ELSE
SET CLASS TO BE
ENDIF
EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
SAVE POINTER TO NEW NAME
IF CLASS IS DATA BASE OR DRDE, THEN
EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS
ENDIF
IF CLASS IS DATA BASE, THEN
EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
ENDIF
IF CLASS IS INTERFACE TABLE, THEN
CALL EXEC TO WRITE/READ NEW NAME
BUILD AWA MANAGER REQUEST TO CHANGE NAME IN TOC
BUILD AWA MANAGER REQUEST TO STORE NEW NAME IN TABLE
CALL XREN TO ISSUE REQUESTS
EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR
ELSE
IF CLASS IS DATA BASE, THEN
BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY
EXIT TO :NAMERR: IF RETURN INDICATES ERROR
EXIT TO :INVALID: IF DATA BASE IS AN NDB
ENDIF
BUILD AND ISSUE AWA MANAGER REQUEST FOR NAME CHANGE
EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR
IF CLASS IS DATA BASE OR DRDE, THEN
CALL XRPFM TO FORMAT FILE NAME
CALL FILE MANAGER TO CHANGE DISC FILE NAME
EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR
IF FILE IS A PDB, THEN
CALL XADD TO DELETE OLD PDB FROM XPDG
EXIT TO :NAMAG: IF ERROR RETURNED
CALL XPDG TO ADD NEW PDB NAME TO XPDG
EXIT TO :TELUSER: IF ERROR RETURNED
ENDIF
ENDIF
EXIT XGREM
:SYNTAX: CALL XRMSG TO DISPLAY SYNTAX ERROR AND EXIT
:TOOLNG: CALL XRMGS ("NEW NAME IS TOO LONG") AND EXIT
:CALGET: CALL EXEC TO GET BUFFER CONTAINING NEW TABLE NAME
:NAMERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT
:CLSER: CALL XRMSG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT
:INVALID: CALL XRMSG ("AN NDB CANNOT BE RENAMED") AND EXIT
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1333</td>
<td>2 :NAMEAGM:</td>
</tr>
<tr>
<td>1334</td>
<td>2 CALL FILE MANAGER TO CHANGE NAME BACK</td>
</tr>
<tr>
<td>1335</td>
<td>2 :UNDO:</td>
</tr>
<tr>
<td>1336</td>
<td>2 BUILD AND ISSUE AN AMA MANAGER REQUEST TO CHANGE NAME BACK</td>
</tr>
<tr>
<td>1337</td>
<td>2 CALL XRMSG (&quot;FILE MANAGER ERROR #: RENAME UNSUCCESSFUL&quot;) AND EXIT</td>
</tr>
<tr>
<td>1338</td>
<td>2 :CLEAN:</td>
</tr>
<tr>
<td>1339</td>
<td>2 CALL XRMSG (&quot;PDB NOT LOGGED IN XPDB; SYSTEM ERROR #: ...&quot;) AND EXIT</td>
</tr>
<tr>
<td>1340</td>
<td>1 END XDREM</td>
</tr>
</tbody>
</table>

**NOTE:** The original page is poor.
BEGIN XDREQ

CALL XDREP 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 XRNSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED' ENDIF

CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON

IF CLASS OF ELEMENT IS DBDE 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 REGBUF BEYOND FAILING REQUEST, THEN

MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER

ENDIF

ELSE

SET RSPTR TO 1 INDICATING NO REQUESTS PRESENT

ENDIF

EXIT XDREP

END XDREP
REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
1476 1 BEGIN XRES
1477 2 SET ADBFL TO ZERO
1478 3 ERREXIT IF NEXT TOKEN IS NOT A COMM :ERR9:
1479 4 ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERR9:
1480 5 BETAIN THIS NAME AS DATA BASE TO BE RESTORED
1481 6 INCREMENT TO NEXT TOKEN
1482 7 IF TOKEN IS NOT A COMM, THEN
1483 8 ERREXIT IF TOKEN IS NOT EOS :ERR04:
1484 9 ENDIF
1485 10 BUILD AWA REQUEST FOR TOC ENTRY RETRIEVE
1486 11 CALL XRES TO PROCESS AWA REQUEST
1487 12 ERREXIT IF AWA REQUEST FAILED :ERR10:
1488 13 IF DATA BASE FOUND IS A PDB :ERR3:
1489 14 IF DATA BASE IS A UTBD, THEN
1490 15 CALL XRQFM TO CONSTRUCT QUALIFIED FILE NAME
1491 16 ENDIF
1492 17 CALL OPEN TO OPEN SPECIFIED FILE
1493 18 ERREXIT IF OPEN FAILED :ERR0:
1494 19 CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1495 20 ERREXIT IF READ FAILED :ERROR:
1496 21 INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1497 22 IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1498 23 CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1499 24 ERREXIT IF READ FAILED :ERROR:
1500 1 ENDIF
1501 2 UPDATE TOTBL TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 3 CLEAR ERROR MESSAGE FLAG (ERFLG)
1503 4 IF TOKEN IS EOS (I.E. NO LIST OF ELEMENTS), THEN
1504 5 SET ALFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 6 ELSE
1506 7 SET ALFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 8 CALL XRPLS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 9 EXIT XRES IF ALFLG SET BY XRPLS
1509 10 ENDIF
1510 11 CALL XRES TO RESTORE AWA RESIDENT ELEMENTS
1511 12 1 EXIT XRES IF ALFLG SET BY XRRES
1512 13 CALL XRRES TO RESTORE XRRES'
1513 14 EXIT XRES IF ALFLG SET BY XRRES
1514 15 CALL CLOSE TO CLOSE DATA BASE FILE
1515 16 ERREXIT IF CLOSE FAILED :ERROR:
1516 17 DO WHILE AWA REQUESTS REMAIN IN REBUF
1517 18 CALL XRRES TO PROCESS AWA REQUESTS
1518 19 EXIT XRES IF ALFLG SET BY XRRES
1519 20 ENDDO
1520 1 EXIT XRES
1521 2 :ERR04: CALL XRMSG - 'SYNTAX ERROR, EXTRAMESOS DATA'
1522 2 :ERR05: CALL XRMSG - 'FILE MANAGER ERROR ...
1523 2 :ERR09: CALL XRMSG - 'SYNTAX ERROR, MISSING OR ILLEGAL FIELD'
1524 2 :ERR10: CALL XRMSG - '........ NOT FOUND'
1525 2 :ERR33: CALL XRMSG - 'CANT NOT RESTORE A PDB'
1526 2 DO UNTIL ALL AWA REQUESTS IN REBUF 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 ENDF
**C**

1534 1 C**********
1535 1 CD0  FORTRAN CALLING PROCEDURE
1536 1 CD0  CALL XDSAV
1537 1 CD0  XDSAV PROCESSES THE SAVE DIRECTIVE. A PDB IS CREATED AND THE
1538 1 CD0  CONTENTS OF THE SPECIFIED UTOB ARE COPIED TO IT.
1539 1 CD0  C**********
1540 1 CD1  C**********
1541 1 CD1  INPUT
1542 1 CD2  COMMON XE - CARTR, COMBUF, COMPTR, FLAGS, LU, TOKENS
1543 1 CD2  FILES - *XXXQ (UTOB FILE SPECIFIED)
1544 1 CD2  C**********
1545 1 CD3  OUTPUT
1546 1 CD3  COMMON XE - REGBUF, REPRTR
1547 1 CD3  FILES - *XXXQ (PDB FILE SPECIFIED)
1548 1 CD3  C**********
1549 1 CD4  INTERNAL VARIABLES
1550 1 CD4  DCBPDB - DCB FOR THE PDB FILE; ALLOCATED IN XS COMMON
1551 1 CD4  DCUTDB - DCB FOR THE UTOB FILE; ALLOCATED IN XS COMMON;
1552 1 CD4  CONTAINS 1152 WORD BUFFER USED TO READ THE
1553 1 CD4  UTOB AND TO WRITE THE PDB.
1554 1 CD4  C**********
1625 2 :ERROR: ISSUE MESSAGE 'PDB FILE ACCESS ERROR ... ."
1626 2 :ERROR: ISSUE MESSAGE "FILE MANAGER ERROR ... ...... ."
1627 2 :ERROR: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PDB'S
1628 2 ALREADY REACHED"
1629 2 :ERROR: ISSUE MESSAGE "ANA OVERFLOW. ..... NOT LOGGED"
1630 2 :RETURN:
1631 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1632 2 PURGE PDB FILE
1633 2 ENDIF
1634 2 IF FLAG INDICATES PDB IS IN XPD, THEN
1635 2 CALL XPD TO DELETE PDB FROM XPD
1636 2 ENDIF
1637 2 IF Flag INDICATES UDB IS OPEN, THEN
1638 2 CLOSE UDB
1639 2 ENDIF
1640 2 IF FLAG INDICATES PDB IS IN XPD, THEN
1641 2 CAL T XPD TO DELETE PDB FROM XPD
1642 2 ENDIF
1643 2 END XDSAV

XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
XDSAV
164: 1 CBO  CALL XDSTA (LU)
164: 2 CBO
164: 3 CBO
164: 4 CBO  [FORTRAN CALLING PROCEDURE]
165: 1 CBO  OUTPUT ANA AND DMA USAGE DATA AND STATISTICS ON ANA COMPACTION
165: 2 CBO  PHASES
165: 3 CBO  INPUT
165: 4 CBO  - LOGICAL UNIT OF OUTPUT DEVICE
166: 1 CBO  LU
166: 2 CBO  XB COMMON
166: 3 CBO  ASZ - TOTAL SIZE OF ANA
166: 4 CBO  HR - AMOUNT OF FREE SPACE IN ANA
166: 5 CBO  DSR - TOTAL SIZE OF DMA (IN 64 WORD SECTORS)
166: 6 CBO  DFR - AMOUNT OF FREE SECTORS IN DMA
166: 7 CBO  HP - ARRAY OF THREE COMPACTION PHASE COUNTS
167: 1 CBO  OUTPUT
167: 2 CBO  USAGE AND STATISTICS TO LU
168: 1 CBO  NOTES
168: 2 CBO  USES EXEC, XRIG
169: 1 CBO  [********]
170: 1 +
171: 2 +
172: 3 +
173: 4 +
174: 5 +
175: 6 +
176: 7 +
177: 8 +
178: 9 +
179: 10 +
180: 11 +
181: 12 +
182: 13 +
183: 14 +
184: 15 +
185: 16 +
186: 17 +
187: 18 +
188: 19 +
189: 20 +
190: 21 +
191: 22 +
192: 23 +
193: 24 +
194: 25 +
195: 26 +
196: 27 +
197: 28 +
198: 29 +
199: 30 +
200: 31 +
201: 32 +
202: 33 +
203: 34 +
204: 35 +
205: 36 +
206: 37 +
207: 38 +
208: 39 +
209: 40 +
210: 41 +
211: 42 +
212: 43 +
213: 44 +
214: 45 +
215: 46 +
216: 47 +
217: 48 +
218: 49 +
219: 50 +
220: 51 +
221: 52 +
222: 53 +
223: 54 +
224: 55 +
225: 56 +
226: 57 +
227: 58 +
228: 59 +
229: 60 +
230: 61 +
231: 62 +
232: 63 +
233: 64 +
234: 65 +
235: 66 +
236: 67 +
237: 68 +
238: 69 +
240: 1 END XDSTA
241: 2 FORMAT AND PRINT ANA USAGE DATA
242: 3 FORMAT AND PRINT DMA USAGE DATA
243: 4 FORMAT AND PRINT COMPACTION STATISTICS
244: 5 END XDSTA
245: 6 END XDSTA
246: 7 END XDSTA
247: 8 END XDSTA
248: 9 END XDSTA
250: 2 FORMAT AND PRINT COMPACTION STATISTICS
683 1 C0************
684 1 C00 CALL XDSTO (DATBUF)
685 1 C00
686 1 C00
687 1 C00
688 1 C00
689 1 C01 XDSTO IS THE STORE DIRECTIVE HANDLER. IT VERIFIES INPUTS ON
690 1 C01 DIRECTIVE, BUILDS UTDB TOC, CREATES UTDB AND STORES UTDB TOC
691 1 C01 ENTRY IN AWA.
692 1 C01
693 1 C01
694 1 C02
695 1 C02
696 1 C02
697 1 C02
698 1 C02
699 1 C02
700 1 C04
701 1 C04
702 1 C04
703 1 C04
704 1 C04
705 1 C04
706 1 C04
707 1 C04
708 1 C04
709 1 C04
710 1 C04
711 1 C04
712 1 C04
713 1 C04
714 1 C04
715 1 C04
716 1 C05
717 1 C05
718 1 C05
719 1 C05
720 1 C05
721 1 C05
722 1 C05
723 1 C05
724 1 C05
725 1 C05
726 1 C05
727 1 C05
728 1 C05
729 1 C05
730 1 C05
731 1 C05
732 1 C05
733 1 C05
734 1 C05
735 1 C05

INTERNAL XB COMMON USED:

X0(151) ABFLE - (INTEGER, 1 WORD) ABORT FLAG
X0(152) ERFLE - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
X0(153) NSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER TO BE DISPLAYED
X0(157) TOTSZ- (INTEGER, 1 WORD) TOTAL SIZE OF UTDB FILE
X0(158) TOTWRS- (INTEGER, 1 WORD) TOTAL WORDS IN A BRDE FILE
X0(159) FILE - (INTEGER, 3 WORDS) UTDB FILE NAME (**XXX)
X0(162) DATREC- (INTEGER, 1 WORD) RECORD # WHERE DATA GOES NEXT
X0(164) UBBERR- (INTEGER, 1 WORD) UTDB FILE ERROR FLAG
X0(166) UBBNM- (INTEGER, 1 WORD) UTDB NAME (**XXX)
X0(201) MOTOC- (INTEGER, 1 WORD) NUMBER OF TOC ENTRIES
X0(201) TOCBUF- (INTEGER, 1200 WORDS) UTDB TOC BUFFER

COMMON USED:

+ (X0(4), FLGS), (X0(49), REPTR),
+ (X0(20), RERBUFF), (X0(85), EOS ),
+ (X0(99), NAME ), (X0(113), COMMA ),
+ (X0(142), ICR ), (X0(144), COMPTR ),
+ (X0(145), COMBUF )

RTE ROUTINES USED:

CLOSE, CREAT, EXEC, KCVT, PURGE, WRITF

FDS ROUTINES USED:

XDSL, XDBRT, XRCPR, XREQ, XREXT, XRMOV,
XRMST, XRSET, XRSFL, XRSFR, XRSFU, XUBPG
1737 1  BEGIN XDBTO
1738 2  BEGIN ABFLG TO ZERO (ABORT FLAG)
1739 2  :ERR IF UTDB NAME IS NOT VALID TO :ERR2:
1740 2  :ERR IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1741 3  CALL XREQ TO MAKE MANAGER REQUEST
1742 3  CALL EXEC TO GET MAN TOC
1743 3  ERR IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1744 4  ENDDO
1745 4  SET ERFLG OFF (ERROR MESSAGE FLAG)
1746 5  SET TOTSI = O (UTDB TOTAL SIZE)
1747 5  IF WHOLE AW A IS TO BE STORED THEN
1748 6  DO FOR IT, ST, DE, ORDE UTDB TOC ENTRIES
1749 7  DO FOR EACH ENTRY IN THIS CHAIN
1750 8  IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1751 9  PREFIX IS NOT AN AMPERSAND THEN
1752 5  INCREMENT TOTSI BY SIZE OF THIS ELEMENT
1753 5  ENDIF
1754 6  ENDDO
1755 5  ELSE
1756 6  CALL XOLIS TO PROCESS LIST TO BE STORED
1757 7  ERR IF ABFLG IS NOT ZERO TO :ERR5:
1758 8  ENDIF
1759 7  SET NOTOC = O (NUMBER OF UTDB TOC ENTRIES)
1760 8  DO FOR IT, ST, DE, ORDE CHAINS
1761 9  DO FOR EACH ENTRY IN THIS CHAIN
1762 10  IF STORE/RESTORE BIT ON IN TOC ENTRY
1763 11  TURN STORE/RESTORE BIT OFF
1764 12  BUILD UTDB TOC ENTRY
1765 13  INCREMENT NOTOC BY 1
1766 12  ENDIF
1767 11  ENDDO
1768 10  ERR IF THERE ARE NO UTDB TOC ENTRIES (NOTOC=O) TO :ERR2:
1769 9  COMPUTE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1770 8  CALL XP ' ' TO CREATE FILE NAME
1771 7  CALL TO CREATE UTDB FOR TOTSI
1772 6  ERR IF ERROR IN CREATE TO :ERR3:
1773 5  CALL TOWRT TO WRITE UTDB FILE
1774 4  ERR IF ABFLG IS 4 (ORDE LARGER THAN SPECIFIED) TO :ERR1:
1775 3  ERR IF ABFLG IS 3 (ORDE FILE ERROR) TO :ERR4:
1776 2  ERR IF ABFLG IS 2 (ORDE FILE ERROR) TO :ERR5:
1777 1  ERR IF WRITE TOC RECORDS AT RECORD 1
1778 2  ERR IF ERROR IN WRITE TO :ERR3:
1779 1  CALL CLOSE TO CLOSE UTDB FILE
1780 2  ERR IF ERROR IN CLOSE TO :ERR3:
1781 1  BUILD REQUEST TO ALLOCATE UTDB IN AW A
1782 2  CALL XREQ TO MAKE REQUEST
1783 1  ERR IF AVA OVERFLOW TO :ERR1:
1784 1  EXIT XDBTO
1785 2  ERR1:
1786 3  CALL XMSG TO DISPLAY MSGNO
1787 2  GO TO :ERR4:
1788 1  :ERR2;
1789 1  XDBTO
1866 1 BECOME XDOTC
1867 2 5. 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 10 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
1876 11 SET OUTPUT UNIT FOR LINE PRINTER
1877 12 ENEIF
1878 13 INCREMENT TO NEXT TOKEN
1879 14 ENEIF
1880 15 IF TOKEN IS NOT EOS
1881 16 THEN
1882 17 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1883 18 INCREMENT TO NEXT TOKEN
1884 19 IF TOKEN IS NOT A COMMA
1885 20 THEN
1886 21 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (O, F, I, S OR B)
1887 22 SET OPTION FOR INDICATED CLASS
1888 23 INCREMENT TO NEXT TOKEN
1889 24 ENEIF
1890 25 IF TOKEN IS NOT EOS
1891 26 THEN
1892 27 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1893 28 INCREMENT TO NEXT TOKEN
1894 29 EXIT TO :ERROR: IF TOKEN IS NOT A FOUR CHARACTER NAME
1895 30 INCREMENT TO NEXT TOKEN
1896 31 EXIT TO :ERROR: IF TOKEN IS NOT EOS
1897 32 CALL XREQ TO RETRIEVE DATA BASE FILE TOC ENTRY
1898 33 EXIT TO :ERROR: IF NOT LOGGED IN TOC
1899 34 GET TYPE AND SET NAME IN HEADER
1900 35 CONSTRUCT FILE NAME
1901 36 READ FIRST DATA BASE FILE TOC RECORD
1902 37 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AWA TOC
1903 38 READ SUBSEQUENT TOC RECORDS
1904 39 EXIT TO :ERROR? IF FILE ACCESS FAILS
1905 40 DO FOR EACH NON-EMPTY TOC CHAIN
1906 41 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
1907 42 IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NULL CHAIN EXISTED
1908 43 THEN
1909 44 MARK PREVIOUS ENTRY AS AN END OF CHAIN
1910 45 ENEIF
1911 46 EMDDO
1912 47 DO FOR EACH ENTRY IN TOC
1913 48 IF NOT MARKED AS AN END OF CHAIN
1914 49 THEN
1915 50 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
1916 51 ENEIF
1917 52 EMDDO
1918 53 EXCLUDE CHAIN 8 (DATA BASE FILES) FROM DISPLAY
1919 54 ENEIF
1920 55 ENDIF
1921 56 ENDIF
1922 57 IF REFERENCING AWA TOC
1923 58 THEN
1924 59 CALL XREQ TO RETRIEVE AWA TOC
IF DEBUG AND/OR TRACE FLAGS ARE SET
THEN
EXTEND OPTION TO INCLUDE SYSTEM CLASSES (0, 1, 5 AND 7)
ENDIF
ENDIF
OUTPUT TOC HEADER
DO FOR EACH CLASS INDICATED BY OPTION
OUTPUT CLASS HEADER
LOCATE CLASS CHAIN HEAD
DO UNTIL END OF CHAIN FOUND (-32768)
INDEX TO NEXT TOC ENTRY
IF CHAIN "OINTS WITHIN LIMIT OF BUFFER
THEN
FORMAT NAME & SIZE FIELDS
IF DEBUG AND/OR TRACE FLAGS SET
THEN
FORMAT I-DIM FIELD IN OCTAL
ENDIF
FORMAT TYPE FIELD IN INTEGER
IF CLASS 2 OR 3
THEN
FORMAT TYPE FIELD USING DATA TYPE TABLE
IF CLASS 2
THEN
FORMAT I-DIM & J-DIM FIELDS
ENDIF
ELSE
IF CLASS 8
THEN
FORMAT TYPE FIELD USING FILE TYPE TABLE
ENDIF
ENDIF
ELSE
PRINT 'DATA LOST' MESSAGE
EXIT PROCESSING FOR THIS CHAIN
ENDIF
PRINT ENTRY
ENDIF
ENDDO
ENDIF
ENDDO
IF PROCESSING AWAY TOC
THEN
CALL XSTA TO DISPLAY AWAY 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 UTDB FILE NAME
:ERROR7: EXIT WITH UTDB FILE ACCESS ERROR
1 END XDTOC
1976 1 CD----------
1977 1 CD
1978 1 CD
1979 1 CD
1980 1 CD
1981 1 CD
1982 1 CD
1983 1 CD
1984 1 CD
1985 1 CD
1986 1 CD
1987 1 CD
1988 1 CD
1989 1 CD
1990 1 CD
1991 1 CD
1992 1 CD----------
1993 1 CD
1994 1 CD
1995 1 CD
1996 1 CD
1997 1 CD
1998 1 CD
1999 1 CD
2000 1 CD
2001 1 CD
2002 1 CD
2003 1 CD
2004 1 CD
2005 1 CD
2006 1 CD
2007 1 CD
2008 1 CD
2009 1 CD
2010 1 CD
2011 1 CD
2012 1 CD
2013 1 CD
2014 1 CD----------
2015 1 CD
2016 1 CD
2017 1 CD
2018 1 CD
2019 1 CD
2020 1 CD
2021 1 CD
2022 1 CD
2023 1 CD
2024 1 CD
2025 1 CD
2026 1 CD
2027 1 CD
2028 1 CD
2029 1 CD
2030 1 CD
2031 1 CD----------
BEGIN XDMR

SET WRDM = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)

set WORE = 0 (NUMBER OF AWAY REQUESTS IN REBUF)

DO ALL UTD DB TOC ENTRIES UNTIL CLASE IS DRDE

BUILD REQUEST FOR DATA FROM AWAY

INCREMENT WORE BY 1

IF REQUEST BUFFER IS FULL (NORES=8) THEN

PERFORM REBDAT TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE

ENDIF

END IF

IF THERE ARE REMAINING REQUESTS (NORES>0) THEN

SET NEXT REQUEST TO END OF REQUEST LIST

PERFORM REBDAT TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE

IF THERE IS A PARTIAL DATA RECORD LEFT (WRDMO>1) THEN

PERFORM WRITE TO OUTPUT DATA TO UTD FILE

END IF

DO FOR EACH DRDE UTD TOC ENTRY

STORE DATREC IN UTD TOC ENTRY

CALL EKRFM TO CREATE FILE NAME

IF DRDE FILE IS TYPE 3 THEN

CALL OPEN TO OPEN FILE AS CORRECT TYPE

ERRERIT IF OPEN ERROR TO :ERR1:

DO UNTIL EOF IS READ

CALL READ TO READ 1 RECORD

ERRERIT IF READ ERROR TO :ERR1:

STORE RECORD LENGTH AT FRONT AND REAR OF DATA

INCREMENT WRDMO BY LENGTH + 2

IF THERE IS ENOUGH DATA TO WRITE (WRDMO>128) THEN

PERFORM WRITE TO OUTPUT DATA TO UTD FILE

ENDIF

END IF

IF THERE IS REMAINING DATA (WRDMO>1) THEN

PERFORM WRITE TO OUTPUT DATA TO UTD FILE

ENDIF

SET DATREC TO NEXT AVAILABLE RECORD FOR DATA

ELSE

CALL OPEN TO OPEN FILE AS TYPE 1

ERRERIT IF OPEN ERROR TO :ERR1:

COMPUTE TOTAL SIZE OF FILE IN WORDS

DO UNTIL ALL DATA IS COPIED TO UTD (SIZE=0)

IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN

SET LENGTH = SIZE

ENDIF

CALL READ TO READ LENGTH DATA

ERRERIT IF READ ERROR TO :ERR1:

CALL WRITF TO WRITE LENGTH DATA

ERRERIT IF WRITE ERROR TO :ERR1:

INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN

DECREMENT SIZE BY LENGTH IN WORDS WRITTEN

ENDIF

END DO

CALL CLOSE TO CLOSE DRDE FILE

ERRERIT IF CLOSE ERROR TO :ERR1:

END DO

EXIT XDMR
BEGIN READ:
CALL XREU TO MAKE REQUEST
EXIT IF THERE IS AN OVERFLOW TO :ERR4:
DO FOR NUMBER OF REQUESTS (MORE?) IN BUFFER
CALL EXEC TO GET DATA FROM SAM
SET DATREC AND WDBO INTO THIS UTDB TOC ENTRY
INCREMENT WDBO BY SIZE OF THIS ELEMENT
IF THERE IS ENOUGH DATA TO WRITE (WDBO>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
REINITIALIZE REQUEST BUFFER (MORE=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
EXIT IF THERE ARE MORE BLOCKS THAN SPECIFIED TO :ERR2:
ENDIF
CALL WRITF TO WRITE LENGTH DATA TO UTDB FILE
EXIT IF WRITE ERROR TO :ERR3:
DECREMENT DATREC BY LENGTH
INCREMENT DATREC BY LENGTH/128
IF THERE IS REMAINING DATA (WDBO>1) THEN
CALL XRMV TO MOVE REMAINING DATA UP IN BUFFER
ENDIF
END WRITE

:ERR1:
CALL XRMG WITH DRE NAME AND RC
SET ABFLG TO SAY A DRE ERROR
GO TO :ERR3:

:ERR2:
CALL XRMG TO SAY DRE FILE LARGER THAN SPECIFIED IN TOC
SET ABFLG TO SAY DRE FILE ERROR

:ERR3:
CLOSE TO CLOSE DRE FILE
SET ABFLG TO SAY UTDB FILE ERROR
EXIT XDWRT

:ERRA:
SET ABFLG TO SAY AWAY OVERFLOW
DO FOR REMAINING REQUESTS
CALL EXEC TO FREE CLASS NUMBER
END
END XDWRT
1  CO0  FORTRAN CALLING PROCEDURE
2  CO0  CALL XELDS (THREE WORD ARRAY CONTAINING 'XECAL')
3  C00
4  C00
5  C00
6  C00
7  C00
8  C00
9  C00
10  C00
11  C00
12  C00
13  C00
14  C00
15  C00
16  C00
17  C00
18  C00
19  C00
20  C00
21  C00
22  C00
23  C00
24  C00
25  C00
26  C00
27  C00
28  C00
29  C00
30  2  CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SREPDT:, :INTEDT:) SUBSTA (-1 TO 3)
31  3  :GLOBAL: CALL XEIN TO INITIALIZE GLOBAL COMMON
32  3  :DIRECT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES
33  3  :EXECUT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL
34  3  :SREPDT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING
35  3  :INTEDT: CALL XEIN TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING
36  2  ENDCASE
37  1  ENDCAL
30 1 CDO  FORTRAN CALLING PROCEDURE
40 1 CDO
41 1 CDO  CALL XEIND
42 1 CDO
43 1 C*******
44 1 CDO  INITIALIZE XE COMMON FOR DIRECTIVE LEVEL
45 1 CDO
46 1 C*******
47 1 C** INPUT
48 1 CDO  COMMON XE - CARTEL, FLAGS
49 1 CDO  XRPM - FDS DIRECTIVE PROMPT FILE
50 1 CDO
51 1 CDO
52 1 C*******
53 1 CDO  OUTPUT
54 1 CDO  COMMON XE - DIRECT, NUMDIR
55 1 CDO
56 1 C*******
57 1 CDO  NOTES
58 1 CDO  USES CLOSE, EXEC, OPEN, PRM, READF, XREXT, XRIG, XMSG, XVA8N
59 1 CDO
60 1 C*******
61 1 *
62 1 *
63 1 *
64 1 *
65 1 BEGIN XEIND
66 2 READ FDS DIRECTIVE PROMPT FILE
67 2 STORE DIRECTIVES IN DYNAMIC COMMON
68 1 END XEIND
1 BEGIN XEINE

2 INITIALIZE COMMON TO ZEROS

3 SET FILE (ART-DCS NUMBER

4 SET STATES TO DIRECTIVE LEVEL

5 SET TKMLNG

6 INITIALIZE TOKENS

7 READ LIBRARY DIRECTORY FIRST RECORD

8 EXIT TO :ERR01: IF READ FAILED

9 STORE NUMBER OF PROCESSORS IN NDPROC

10 READ PROCESSOR NAME RECORD

11 EXIT TO :ERR01: IF READ FAILED

12 CALL XER0 TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AMA

13 EXIT TO :ERR01: 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 THEN

17 NOTE EXISTANCE OF ID ENDIF

18 ENDDO

19 IF ANY PROCESSORS NOT MARKED THEN

20 LIST PROCESSORS MISSING ID SEGMENTS

21 EXIT XEINE WITH INITIALIZATION FAILURE

22 ENDF

23 CALL OPEN TO OPEN PDB/MDB DIRECTORY ( XPDB)

24 EXIT TO :ERR01: IF OPEN FAILED

25 CALL READ TO READ 1ST RECORD OF XPDB (LIST OF MDB'S)

26 EXIT TO :ERR01: IF READ FAILED

27 PERFORM DLOG TO LOG MDB'S IN AMA

28 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER

29 CALL READ TO READ THAT RECORD OF XPDB

30 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)

31 CALL CLOSE TO CLOSE XPDB

32 PERFORM DLOG TO LOG PDB'S IN AMA

33 IF THERE ARE REQUESTS IN THE AMA REQUEST BUFFER, THEN

34 CALL XER0 TO PROCESS THE AMA REQUESTS

35 ENDF

36 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL

37 EXIT XEINE

38 BEGIN DLOG

39 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED

40 BUILD AN ENTRY IN AMA REQUEST BUFFER TO ALLOCATE THIS DATA BASE

41 IF 5 AMA REQUESTS HAVE BEEN BUILT, THEN

42 CALL XER0 TO PROCESS AMA REQUESTS

43 ENDF

44 EXIT TO :ERR01: IF A REQUEST FAILED

45 ENDDO

46 END DLOG

47 :ERR01: LIBRARY INITIALIZATION ERROR TERMINATION

48 END XEINE
FORTRAN CALLING PROCEDURE

CALL XEINI

*********

INITIALIZE XE AND XB COMMON FOR INTERFACE TABLE EDITING

**********

INPUT

COMMON XE -

MASTA = MASTER STATE

TOKENS = LEXICAL TOKEN VALUES

NOPRO = NUMBER OF ENTRIES IN LIBRARY DIRECTORY

COMTR = INDEX OF NEXT TOKEN IN COMBUF

COMBUF = COMMUNICATIONS BUFFER

PRCNAM = PROCESSOR NAME FROM EXECUTION CONTROLLER

LIBRARY -

INTMN = OLD INTERFACE TABLE TO BE EDITED

<XXXX = DEFAULT INTERFACE TABLE

>XXXX = PROMPT TABLE

**********

OUTPUT

COMMON XE -

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

REQBUF = REQUEST BUFFER FOR AWA INPUTS

PRCNAM = PROCESSOR NAME ON INTE DIRECTIVE

COMMON XB -

WKBLNG = LENGTH OF WKBUF (CONSTRUCT)

WKBUF = WORKING BUFFER CONTAINING IN/E. TABLE WITH

SHORT PROMPTS

DIRECT = SUPPORTED INTE DIRECTIVES

HNUMDIR = NO. OF ENTRIES IN DIRECT

HMENTAB = HEAD OF INTE. TABLE NAME

HNUMARG = NO. OF ARGUMENTS

WMARG = NO. OF WORDS IN SPACED AND HEADER OF WKBUF

LITTLE = NO. OF WORDS IN LITERAL AREA

LITPRX = INDEX TO START OF LITERAL AREA

ISIZES = ARRAY MAPPING ARG. TYPE TO EFFECTIVE LENGTH

**********

NOTES

<XXXX = DEFAULT INTERFACE TABLE FOR THIS PROCESSOR

>XXXX = PROMPT TABLE FOR THIS PROCESSOR

USES ROUTINES

XRMOV = EXEC

XRMSG = CLOSE

XREST = OPEN

XREQ = READF

XUPK = XRADB

XRCP = XRCPN

BEGIN XEINI

IF CALLED AS A RESULT INTE DIRECTIVE, THEN
POSITION TO 1ST TOKEN AFTER 'INTE'
ERROR IF TOKEN IS NOT ',', :ERR1:
INCREMENT TO NEXT TOKEN
ERROR IF TOKEN IS NOT A NAME :ERR2:
SET PRCNAM TO THIS NAME
INCREMENT TO NEXT TOKEN
SET INTNAM TO 0
SET NEWTAB TO 'EXIT'
ERROR IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN
ERROR 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
    ENDIF
LAT LOCAL MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED
MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE
ERROR IF REQUEST IS UNSUCCESSFUL, :ERR11:
START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, DC
EXIT IF PROCESSOR PRCNAM IS FOUND
ELSE
ENDIF
ENDIF
ENDIF
ELSE
INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER
NEWTAB IS SET TO 'EXIT'
MAKE MANAGER REQUEST FOR LIBRARY INTERFACE FILE
ERROR IF REQUEST IS UNSUCCESSFUL :ERR10:
ERROR IF OPEN FAILED :ERR12:
READ DEFAULT INTERFACE TABLE
ERROR IF THE READ FAILED :ERR16:
READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM
OF WKBUF
ERROR IF READ FAILED :ERR14:
ELSE
MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE
ERROR IF ERROR OR COULD NOT FIND :ERR:
ENDIF
ERROR IF VERS .ME. VERSION NO. OF TABLE TO BE EDITED :ERR10:
CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"
OPEN THE PROMPT TABLE
ERROR IF OPEN FAILED :ERR15:
READ 2ND RECORD OF THIS PROMPT TABLE
ERREXIT IF READ FAILED :ERN17:
MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUF TO AREA FOLLOWING
THE PROMPT TABLE
THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
LITERAL DISPL. THAT MATCH EACH LITERAL DATA ENTRY
THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
EXIT XEINI

:ERR2: "SYNTAX ERROR ... "
:ERR4: "INTNAM" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNAM DOES NOT MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
304 1 CD0       FORTRAN CALLING PROCEDURE
305 1 CD0
306 1 CD0
307 1 CD0       CALL XEINS
308 1 CD0
309 1 CD0       ****************
310 1 CD1
311 1 CD1       INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING
312 1 CD1
313 1 CD1       ****************
314 1 CD2
315 1 CD2       INPUT
316 1 CD2
317 1 CD2       COMMON XE - CARTAG, COMBUF, COMPTR, LU, NOPROC, TOKENS
318 1 CD2
319 1 CD2       ****************
320 1 CD3
321 1 CD3       OUTPUT
322 1 CD3
323 1 CD3       COMMON XB - REQBUF, REQPTR, SUBSTA,
324 1 CD3
325 1 CD3       COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,
326 1 CD3
327 1 CD3       OLDTAB, PRMTAB, WKBLNG, WKBUF, XLIBD
328 1 CD3       ****************
329 1 CD5
330 1 CD5
331 1 CD5       NOTES
332 1 CD5
333 1 CD5       USES FILES - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE
334 1 CD5
335 1 CD5       USES ROUTINES
336 1 CD5
337 1 CD5       EXEC
338 1 CD5       IAND
339 1 CD5       OPEN
340 1 CD5       READF
341 1 CD5       XCRPR
342 1 CD5       XREQ
343 1 CD5       XREX
344 1 CD5       XRMOV
345 1 CD5       XRMGR
346 1 CD5       XDDBG
347 1 CD5       ****************
1  BEGIN XEMS
2       BUILD AW A REQUEST TO RETRIEVE "XLIDB";
3              ERREXIT IF TOKEN IS NOT COMMA :ERR02:
4              INCREMENT TO NEXT TOKEN
5              IF TOKEN IS NAME, THEN
6              BUILD AWS REQUEST TO RETRIEVE OLDTAB
7              RETAIN THIS NAME AS OLDTAB
8              INCREMENT TO NEXT TOKEN
9 ELSE
10              SET OLDTAB TO ZERO
11            ENDIF
12            ERREXIT IF TOKEN IS NOT COMMA :ERR02:
13            INCREMENT TO NEXT TOKEN
14            ERREXIT IF TOKEN IS NOT NAME :ERR02:
15            RETAIN THIS NAME AS NEWTAB
16            IF NEWTAB NOT EQUAL TO OLDTAB, THEN
17               BUILD AWS REQUEST TO VERIFY EXISTENCE OF NEWTAB
18            ENDIF
19            CALL XRER TO PROCESS AWS REQUEST(S)
20            IF NEWTAB NOT EQUAL TO OLDTAB, THEN
21               ERREXIT IF NO ERROR FROM XRER :ERR12:
22            ENDIF
23            IF OLDTAB NOT ZERO, THEN
24               ERREXIT IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED :ERR04:
25               SET PROMPT MODE AS UPDATE ...
26               SET NO. ENTRIES AS OLDTAB SIZE / 7
27               READ OLDTAB INTO WORKING BUFFER
28               ERREXIT IF FIRST REQUEST (RETRIEVE "XLIDB") FAILED :ERR01:
29               READ "XLIDB" INTO COMMON
30 ELSE
31               SET PROMPT MODE AS CREATE
32               SET NO. ENTRIES TO ZERO
33            ENDIF
34            OPEN, READ AND CLOSE FILE XSPRM
35            SET COUNT AND SEREDIT DIRECTIVES INTO XB COMMON
36            SET SUBSTATE FLAG TO SEQ. EDIT. (=2)
37            EXIT XEMS
38  2     :ERR01: CALL XRMSG - 'INITIALIZATION ERROR ....'
39  2     :ERR02: CALL XRMSG - 'SYNTAX ERROR'
40  2     :ERR04: DEFAULT MESSAGE TO '...NOT FOUND'
41            IF ERROR WAS NO AWS SPACE THEN
42               SET MSG TO '...NO AWS SPACE'
43            ENDIF
44            CALL XRMSG TO DISPLAY MESSAGE
45  2     :ERR12: CALL XRMSG - '... ALREADY EXISTS'
46  1     END XEMS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XB - LITPTR, NUMARS, WKLNG, WKBuffers

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

OUTPUT

COMMON XB - LITPTR, LITDYN, NARC, W'DUF

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

USES ROUTINES

XEXIT XRMOV XINXKB XRMSG XRSET
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 E-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 ENDIF
14 DO UNTIL ALL BITS OF BIT MASK PROCESSED
15 DETERMINE NUMBER OF CONTIGUOUS BITS ON (OR OFF)
16 AND MOVE CORRESPONDING NUMBER OF DATA WORDS
17 (OR 0'S) INTO NEW LITERAL ENTRY AREA
18 ENDDO
19 ORELSE
20 INCREMENT TO NEXT ARGUMENT
21 ENDLOOP
22 ERROR IF NOT CALLED BY LIBRARY MAINTENANCE :ERR5:
23 SKIP TO NEXT LITERAL ENTRY
24 ENDS
25 ENDDO
26 1 EXIT TO :RETURN:
27 :ERR5: CALL XRM6G - 'SYSTEM INITIALIZATION ERR 5 '}

1 ENDDO XEINT
FORTRAN CALLING PROCEDURE

CALL XEINX

INITIALIZE XE AND XB COMMON FOR EXECUTION CONTROLLER

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

COMMON XB - SEQUENCE TABLE, LIBRARY DIRECTORY NAME TABLE

COMMON XE - COMPTR, MASSTA, RERBUF, RERPTR, SERRND, SERNAME, SERPTR

COMMON XB - LIBD, NOPRC2, SELNG, WKLNG, WKBUF

USES EXEC, PTMN, XRNG, XREAT, XRMV, XMMSG, XUDEG, XVABN
1 BEGIN XEINX
2 IF INITIALIZATION FROM DIRECTIVE
3 THEN
4 IF DIRECTIVE IS NAME
5 THEN
6 EXIT TO :ERROR2: IF NEXT TOKEN IS NOT EOS
7 ELSE
8 IF DIRECTIVE IS AUTO
9 THEN
10 IF TOKEN IS A 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
16ENDIF
17ENDIF
18EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A COMMA
19INCREMENT TO NEXT TOKEN
20EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A NAME
21STORE NAME IN SEQNUM
22CALL REG TO RETRIEVE SEQUENCE TABLE
23EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
24INCREMENT TO NEXT TOKEN
25SET SERSTA TO FIRST SEQUENCE NUMBER
26SET SEERND TO LAST SEQUENCE NUMBER
27IF TOKEN NOT EOS
28THEN
29EXIT TO :ERROR2: IF TOKEN NOT A COMMA
30INCREMENT TO NEXT TOKEN
31IF TOKEN IS AN INTEGER
32THEN
33STORE STARTING RANGE NUMBER
34SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
35EXIT TO :ERROR1: IF NOT FOUND
36INCREMENT TO NEXT TOKEN
37ENDIF
38IF TOKEN NOT ETC.
39THEN
40EXIT TO :ERROR2: IF TOKEN NOT A COMMA
41INCREMENT TO NEXT TOKEN
42EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
43ERROR TO :ERRORS: IF TOKEN IS REPEATED
44SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
45EXIT TO :ERROR1: IF NOT FOUND
46INCREMENT TO NEXT TOKEN
47EXIT TO :ERROR2: IF TOKEN NO. :OS
48ENDIF
49ENDIF
50ENDIF
51SET SERPRT TO SERSTA
52ENDIF
53ENDIF
54INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
55EXIT TO :ERROR1: IF INITIALIZATION FAILS
56EXIT KEINX
57:ERROR1: INITIALIZATION FAILURE TERMINATION
58:ERROR2: SET SUBSTA 10 DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
59XENIX
60XENIX
61XENIX
62XENIX
63XENIX
64XENIX
65XENIX
66XENIX
67XENIX
68XENIX
69XENIX
70XENIX
71XENIX
72XENIX
73XENIX
74XENIX
551 2 :ERROR3: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552 2 :ERROR4: SET SUBSTA TO DIRECTIVE LEVEL
553 2 IF ERROR WAS NO AWA SPACE THEN
554 2 EXIT WITH NO AWA SPACE ERROR
555 2 ELSE
556 2 EXIT WITH SEQUENCE TABLE NOT FOUND ERROR
557 2 ENDIF
558 2 :ERROR5: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
559 2 :ERROR13: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH NUMBER NOT FOUND
560 1 EXIT XEIXX
562 1 CD**********
563 1 CD
564 1 CD FORTRAN CALLING PROCEDURE:
565 1 CD
566 1 CD CALL XELDS (SEGNM) TO CALL SEGMENT FROM MAIN
567 1 CD CALL XERTH TO RETURN TO MAIN PROGRAM
568 1 CD
569 1 CD**********
570 1 CD 1 YELDS allows a main program to "call" a segment and
571 1 CD the entry point XERTH passes control back to the main program
572 1 CD
573 1 CD**********
574 1 CD
575 1 CD 1 CD2 inputs in calling sequence:
576 1 CD 1 CD2 SEGNM - (INTEGER, 3 WORDS) ARRAY CONTAINING NAME OF THE
577 1 CD 1 CD2 SEGMENT TO BE LOADED
578 1 CD
579 1 CD
580 1 CD
581 1 CD**********
582 1 CD5 subroutines and functions called:
583 1 CD
584 1 CD
585 1 CD
586 1 CD5 notes:
587 1 CD5 1) XELDS CONTAINS 2 ENTRY POINTS: XELDS AND XERHT
588 1 CD5 2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
589 1 CD5 3) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XERTH"
590 1 CD5 FOLLOWED BY A CALL TO MAIN WHICH IS NOT EXECUTED
591 1 CD5 4) WHEN LOADER IS RUN, THE FOURTH PARAMETER MUST BE 1
592 1 CD5 INDICATING TO LOAD MAIN PLUS SEGMENTS
593 1 CD
594 1 CD
595 1 CD****
596 1 *
597 1 *
598 1 *
599 1 BEGIN XELDS
600 2 :XELDS:
601 2 CALL .ENTR TO RETRIEVE PARAMETERS AND RETURN ADDRESS
602 2 CALL ADDR OF THE SEGMENT NAME INTO EXEC PARAMETER LIST
603 2 CALL EXEC TO LOAD AND PASS CONTROL TO SEGMENT
604 2 :XERTH:
605 2 RETURN TO MAIN PROGRAM
606 1 END XELDS
FORTAN CALLING PROCEDURE

CALL XESC

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

*********

INPUT

X COMON - CARTNG

ANA - DOSTK, BSCNTB (SEE DO AND SCAN PROCESSORS

*********

OUTPUT

X COMON - REBUF, RERPR

X B COMON - SCRATCH

*********

EXTERNAL ROUTINES

CLOSE, EXEC, PURGE, XREQ, XRG, XMOV, XRUNG

*********

BEGIN XESC

1 BUILD REQUESTS TO DELETE DOSTK AND RETRIEVE BSCNTB THEN DELETE IT
2 CALL XREQ TO ATTEMPT REQUESTS
3 ISSUE MESSAGE XE18 FOR SUCCESSFUL REQUESTS
4 IF XREQ COMPLETED REQUESTS, I.E., BSCNTB EXISTED, THEN
5 DO FOR EACH SCAN CONTROL ENTRY IN BSCNTB
6 CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN
7 CALL PURGE TO PURGE THE FILE
8 IF PURGE RETURNED AN ERROR, THEN
9 CALL XRSRS TO DISPLAY WARNING MESSAGE
10 ENDIF
11 BUILD REQUEST TO DELETE DATA BOX FROM TOC
12 ENDDO
13 CALL XREQ TO DELETE DATA BOXES FROM ANA TOC
14 ENDIF
15 END XESC
1 CD9 FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
1 CD1 ONE PROGRAM PER SIGNED-ON USER
1 CD3
1 C********
1 CD2 INPUT
1 CD2 SCHEDULING PARAMETERS - LU, CLASHO, GUAL, FLAGS (SEE XE COMMON)
1 CD2
1 C********
1 CD3 OUTPUT
1 CD5 COMMON XE - COMBUF, COMPTR, MASSTA, SUBSTA, PLUS XEINE
1 CD5 INITIALIZATION
1 CD5 COMMON XB - INITIALIZATIONS FROM XEIND, XEINI, XEINS, XEINX
1 CD3
1 C********
1 CD5 NOTES
1 CD5 USES RMPAR, XDCLD, XDCLF, XELDS, XINTE, XRCPR, XRMIG, XSEGE,
1 CD5 XTCOM, XXCMT
1 CD5
1 CD5 THE CALLS TO XELDS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
1 CD5 XEAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
1 CD5
1 CD5 THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CONTROL OCCURS
1 CD5 BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XXCMT FROM
1 CD5 CALLING XSEGE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
1 CD5 MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
1 CD5 WHEN MASSTA IS SET TO THE DIRECTIVE LEVEL.
1 CD5
1 CD5 C********
BEGIN XEXEC

RETIRE SCHEDULING PARAMETERS AND SET LU, CLASMO, QUAL & FLAGS

CALL XEINE TO INITIALIZE GLOBAL COMMON

DO FOREVER -- TERMINATES INSIDE HANDLER FOR XOFF

CALL ATM FOR INPUT OF DIRECTIVE

IF ERROR OR NOT A VALID DIRECTIVE NAME

THEM

ELSE

IF NAME IS INTE

THEM

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 SERE

THEM

SET STATES TO SERE 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

THEM

SET STATES TO APPROPRIATE EXECUTION CONTROL MODE

DO UNTIL MASSTA IS AT DIRECTIVE LEVEL

CALL XEINT TO INITIALIZE DYNAMIC COMMON

EXIT TO :RESET; IF ERROR

CALL XCNMT TO PERFORM EXECUTIONS

IF SUBSTA IS SET TO SERE LEVEL

THEM

CALL XEINS TO REINITIALIZE DYNAMIC COMMON

EXIT TO :RESET; IF ERROR

CALL XSERE 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

IF EXECUTION MODE WAS SEMI OR AUTO

THEN

CALL XECON TO PURRED ANY RESIDUAL SCAN CONTROL DATA AND FILES

ENDIF

ELSE

CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?

ENDIF

ENDIF

:RESET:

IF SUBSTA IS NOT DIRECTIVE LEVEL

THEN

CALL XEIND TO REINITIALIZE DYNAMIC COMMON

ENDIF

ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 CD0 FORTRAN CALLING PROCEDURE
2 CD0 CALL XINTE
3 CD0
4 CD0
5 CD0
6 CD0
7 CD0
8 CD0 OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR
9 CD0
10 CD0
11 CD0
12 CD0
13 CD0
14 CD0 COMMON XE -
15 CD0
16 CD0
17 CD0
18 CD0 COMMON XB -
19 CD0 DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE
20 CD0 TABLE EDITOR Routines
21 CD0
22 CD0
23 CD0
24 CD0 USES Routines
25 CD0
26 CD0 XINIX
27 CD0
28 CD0
29 CD0
30 CD0
31 CD0
32 CD0
33 CD0
34 CD0
35 CD0
36 CD0
37 CD0
38 CD0
Fortran Calling Procedure

CALL XIMIX

Main Program for Interface Table Editor

INPUT

COMMON XB -

COMBUF = TERMINAL COMMUNICATIONS OUTPUT BUFFER
COMFIN = INDEX TO NEXT TOKEN IN COMBUF
BACKLS = TOKEN FOR "/"
TOKENS = IDENTIFYING VALUES OF TOKENS IN COMBUF

COMMON XB -

LITLEN = LENGTH OF LITERAL AREA OF INTERFACE TABLE IN WBUF
MARG = INDEX TO START OF SHORT PROMPT AREA OF WBUF
UNITAB = NAME OF INTERFACE TABLE TO BE CREATED BY THIS EDIT. INPUT = 0 IF XITE IS CALLED BY DFS LIBRARY MAINTENANCE PROGRAM.
NUMARG = NUMBER OF ARGUMENTS IN THIS INTERFACE TABLE

WORKBUF = WORKING BUFFER CONTAINING INTERFACE TABLE AND SHORT PROMPTS WITH LITERAL IN EXPANDED FORM.

OUTPUT

COMMON XB -

MASTER = MASTER STATUS SET TO '2' LEVEL IF ERROR OR IF '2' INPUT

COMMON XB -

PRMTMO = CURRENT PROMPT MODE. SET TO 5 (MODIFY MODE) IF 'Y' INPUT

WMBUF = WORKING BUFFER CONTAINING INTERFACE TABLE

WITH LITERALS IN COMPACTED FORM

AMA - XXXXX = NEW INTERFACE TABLE AND LITERAL AREA

(NOT DONE IF CALLED BY LIBRARY MAINTENANCE)

Note:

Uses Routines

EXEC

XREG

XILIT

XIPRM

XINPT
1 CDS
2 XIMOV
3 XBMSEG
4 XTCON
5 *****************
6 *                   *
7 * THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'IMT?' DIRECTIVE
8 * OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
9 * PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
10 * TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
11 * OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
12 * NOTE: ALL INITIALIZATION, INCLUDING MKBUF (OLD INTERFACE TABLE),
13 * HAS BEEN PERFORMED BY XEINI.
14 *
15 1 BEGIN XIMIX
16 2 IF NP (NO. OF PARAMETERS) > 0, THEN
17 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
18 9 PRMTNO = 1 => CREATE A MODE
19 3 PRMTNO = 3 => CREATE A MODE
20 3 PRMTNO = 4 => CREATE CONTINUE MODE
21 9 PRMTNO = 5 => MODIFY MODE
22 DO UNTIL 'EXIT' OR 'S' IS ENTERED
23 4 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTND, SIZE, TYPE, AND STATUS
24 4 OF NEXT ARGUMENT
25 4 CALL XITCOM TO PROMPT USER AND RETURN PARSED INPUT
26 4 IF 'I' WAS NOT ENTERED, THEN
27 5 IF '\' WAS ENTERED, THEN
28 6 SET PRMTND TO 5
29 7 ELSE
30 6 IF NOTHING WAS ENTERED (I.E. TOKEN IS COS), THEN
31 7 INCREMENT TO NEXT ARGUMENT
32 9 ELSE
33 7 CALL XINPT TO PROCESS THE USER'S INPUT
34 ENDIF
35 ENDIF
36 ENDDO
37 IF 'I' WAS ENTERED, THEN
38 SET RETURN CODE INDICATING I (I.E. MASSTA = 0)
39 ELSE
40 COMPRESS THE LITERAL LIST AREA
41 ENDIF
42 ENDF
43 STORE INTERFACE TABLE AS NEWNAME
44 IF STORE INTO AWA FAILED, THEN
45 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
46 ELSE
47 SET GOOD RETURN CODE
48 ENDIF
49 1 END XIMIX
10 CONSTRUCT PROMPT TO BE ISSUED
11 BEGIN XIPM
12 DO UNTIL A PROMPT IS CONSTRUCTED
13 IF PRTMD = 5, THEN
14 CONSTRUCT A ":" PROMPT
15 ELSE
16 IF PRTMD = 4 (CONTINUE MODE), OR
17 PRTMD = 6 (CONTINUE HERE MODE), THEN
18 IF ARGNO IS A SCALAR, THEN
19 CONSTRUCT PROMPT AS \"ARG\" OR \"ARG\";
20 ELSE
21 IF PRTMD NOT = 6, THEN
22 IF ISUB IS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LASTE)
23 ELSE
24 SET PRTMD TO 4
25 ELSE
26 IF THERE ARE NO EMPTY SLOTS BEYOND LASTE, THEN
27 IF PRTMD TO MOOSAV (EXIT THE CONTINUE MODE)
28 ELSE
29 IF ARGNO IS DOPPLY SUBSCRIPTED PARAMETER, THEN
30 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
31 CONSTRUCT PROMPT AS \"ARG=\"(I,J)\"
32 ELSE
33 CONSTRUCT PROMPT AS \"ARG=(ISUB)\"
34 ENDIF
35 ELSE
36 IF ARGNO = 6P, THEN
37 SET PRTMD TO 5
38 ELSE IF ARGNO TO NEXT PARAMETER
39 SET IFLAG TO 1, 2, OR 3 INDICATING 1, 2, OR 0
40 SET LAST ENTERED INDICATOR (LASTE) TO 0
41 IF PRTMD = 3, THEN
42 IF SOME DATA VALUE(S) OR PARAM NAME EXISTS FOR ARGNO, THEN
43 CALL XILSD TO LIST DATA FOR THIS ARGUMENT
44 ENDIF
45 CONSTRUCT PROMPT AS \"ARG\" OR \"ARG\";
46 ELSE IF ARGNO MARKED INCOMPLETE, THEN
47 IF A PARTIAL LITERAL LIST EXISTS, OR
48 THIS ARGUMENT IS A SCALAR, THEN
49 COMPUTE ISUB AS FIRST EMPTY ELEMENT
50 ELSE IF DOPPLY SUBSCRIPTED PARAMETER, THEN
51 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
52 CONSTRUCT PROMPT \"ARG=\"(I,J)\"
53 ELSE
54 CONSTRUCT PROMPT \"ARG=(ISUB)\"
55 ENDIF
56 ELSE IF ARGNO MARKED INCOMPLETE, THEN
57 COMPUTE ISUB AS FIRST EMPTY ELEMENT
58 ELSE IF DOPPLY SUBSCRIPTED PARAMETER, THEN
59 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
60 CONSTRUCT PROMPT \"ARG=\"(I,J)\"
61 ELSE
62 CONSTRUCT PROMPT \"ARG=(ISUB)\"
63 ENDIF
64 ENDIF
65 ENDIF
66 ENDIF
67 ENDIF
68 ENDIF
69 ENDIF
70 ENDIF
71 ENDIF
72 ENDIF
73 ENDIF
74 ENDIF
75 ENDIF
76 ENDIF
77 ENDIF
78 ENDIF
79 ENDIF
80 ENDIF
81 ENDIF
82 ENDIF
83 ENDIF
84 ENDIF
85 ENDIF
86 ENDIF
87 ENDIF
88 ENDIF
89 ENDIF
90 ENDIF
91 ENDIF
92 ENDIF
93 ENDIF
94 ENDIF
95 ENDIF
96 ENDIF
97 ENDIF
98 ENDIF
99 ENDIF
100 ENDIF
101 ENDIF
102 ENDIF
103 ENDIF
104 ENDIF
105 ENDIF
106 ENDIF
107 ENDIF
108 ENDIF
109 ENDIF
110 ENDIF
111 ENDIF
112 ENDIF
113 ENDIF
114 ENDIF
115 ENDIF
116 ENDIF
117 ENDIF
118 ENDIF
119 ENDIF
120 ENDIF
121 ENDIF
122 ENDIF
123 ENDIF
124 ENDIF
125 ENDIF
126 ENDIF
127 ENDIF
128 ENDIF
129 ENDIF
130 ENDIF
131 ENDIF
132 ENDIF
133 ENDIF
134 ENDIF
135 ENDIF
136 ENDIF
137 ENDIF
138 ENDIF
139 ENDIF
140 ENDIF
141 ENDIF
142 ENDIF
143 ENDIF
144 ENDIF
145 ENDIF
146 ENDIF
147 ENDIF
148 ENDIF
149 ENDIF
150 ENDIF
FOR CALLING PROCEDURE

CALL XIPMT

**FORM**

**INPUT**

COMMON XE - COMBUF, COMTRA, TOKENS

**OUTPUT**

COMMON XB - ARGNO, PRMTMD

* XIPMT PROCEDURE IS THE PROMPT DIRECTIVE

BEGIN XIPMT

IF TOKEN IS NOT COMMA :ERROR2:

POSITION TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT NAME :ERROR2:

IF NAME IS 'M', THEN

IF NAME IS 'A', THEN

SET PRMTMD TO 3

ELSE

ERREXIT :ERROR2:

ENDIF

IF ARGNO TO 0

EXIT TO :RETURN:

:ERROR2: CALL XMSG FOR 'INVALID SYNTAX'

:RETURN:

END XIPMT
337 1 * XILST PROCESSES THE LIST DIRECTIVE
338 2 BEGIN XILST
339 3 IF TOKEN IS ' ' THEN
340 4 POSITION TO NEXT TOKEN
341 5 ERREXIT IF TOKEN IS NOT NAME ; ERRO2:
342 6 ERREXIT IF NAME IS NOT 'C', 'V', OR 'A' ; ERRO2:
343 7 SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)
344 8 POSITION TO NEXT TOKEN
345 9 ELSE
346 10 SET MODEFG TO 2
347 11 ENDF
348 12 IF TOKEN IS EOS, THEN
349 13 WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
350 14 AND STATUS
351 15 DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
352 16 IF MODEFG = 1 OR MODEFG = 3, THEN
353 17 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
354 18 ENDF
355 19 IF MODEFG = 2 OR MODEFG = 3, THEN
356 20 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
357 21 ENDF
358 22 ENDDO
359 23 ELSE
360 24 DO UNTIL EOS IS REACHED
361 25 ERREXIT IF TOKEN IS NOT COMMA ; ERRO2:
362 26 ERREXIT IF NEXT TOKEN IS NOT NAME ; ERRO2:
363 27 SET ARGNO TO 1
364 28 START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
365 29 IF NAME = ARGNO'S NAME IN PROPPT TABLE
366 30 IF MODEFG = 1 OR MODEFG = 3, THEN
367 31 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
368 32 ENDF
369 33 IF MODEFG = 2 OR MODEFG = 3, THEN
370 34 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
371 35 ENDF
372 36 ENDLOOP
373 37 PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
374 38 ENDSEARCH
375 39 INCREMENT TO NEXT TOKEN
376 40 ENDDO
377 41 ENDF
378 42 EXIT TO :RETURN:
379 43 :ERRO2: CALL XRMGR TO WRITE 'INVALID SYNTAX'
380 44 :RETURN:
381 45 END XILST
FORTRAN CALLING PROCEDURE

CALL XISUB

COMMON IE - IEXT, IREL, TOKENS
COMMON XE - ISUB, IREL, LENEFF

OUTPUT

USES ROUTINES

XRMG

CPP

* XISUB IS CALLED TO CALCULATE AN EFFECTIVE SUBSCRIPT (ISUB) FROM
* THE INPUT SUBSCRIPT

BEGIN XISUB

INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:
IF IDIM FOR THIS ARGUMENT > 0, THEN
ERREXIT IF SPECIFIED INTEGER VALUE > IDIM :ERR16:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT COMMA :ERR15:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:
RETURN ISUB AS (J-1)*IDIM+1
ELSE
SET ISUB TO INTEGER VALUE
ENDIF
ERREXIT IF ISUB > AIXE :ERR16:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:
EXIT TO :RETURN:

:ERR14: CALL XRMG - 'INVALID SUBSCRIPT SYNTAX'

:ERR15: CALL XRMG - 'DOUBLE SUBSCRIPTED - MUST SPECIFY BOTH'

:ERR16: CALL XRMG - 'INVALID SUBSCRIPT VALUE'

RETURN:
END XISUB
1 CD0 FORTRAN CALLING PROCEDURE

1 CD0 CALL XIDAT

1 CD0 C**********

1 CD0 C LITERAL DATA PROCESSOR

1 CD0 C**********

1 CD0 INPUT

1 CD0 C CD0 COMMON XB - COMBUF, COMPTR, TOKENS

1 CD0 C CD0 COMMON XB - CFLAG, COMPLG, DFLAG, IARG, IARG2, ISIZE, ISIZE, SSTYPE, LITDF, LITDSP,

1 CD0 C CD0 LITOWN, LITSIZE, MODSAY, MXBTH, MBTH,

1 CD0 C CD0 PRNTND, SFLAG, WKBNG, WKBUF

1 CD0 C**********

1 CD0 OUTPUT

1 CD0 C CD0 COMMON XB - IRETC, ISUB, LITDSP, LITOWN, MXBTH,

1 CD0 C CD0 PRNTND, WKBUF

1 CD0 C**********

1 CD0 INTERNAL VARIABLES

1 CD0 C CD0 COMMON XS -

1 CD0 C CD0 IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS

1 CD0 C CD0 LITERAL DATUM

1 CD0 C CD0 NUMCL = NUMBER OF CONSECUTIVE ELEMENTS TO BE

1 CD0 C CD0 MARKED COMPLETED AS A RESULT OF THIS

1 CD0 C CD0 LITERAL DATUM

1 CD0 C CD0 STACK = PUSH-DOWN LIST (MAX. OF 4 ENTRIES)

1 CD0 C CD0 DESCRIBING NESTED REPEAT GROUPS.

1 CD0 C CD0 EACH ENTRY IS 3 WORDS:

1 CD0 C CD0 WORD 1 = INDEX TO 1ST TOKEN (IN COMBUF)

1 CD0 C CD0 AFTER REPEAT SYMBOL

1 CD0 C CD0 WORD 2 = REPEAT COUNT

1 CD0 C CD0 WORD 3 = FLAG INDICATING WHETHER

1 CD0 C CD0 REPEAT GROUP IS PARENTHETICALLY

1 CD0 C CD0 GROUPED

1 CD0 C CD0 STKPTR = INDEX TO NEXT ENTRY TO BE BUILT IN 'STACK'

1 CD0 C**********

1 CD0 NOTES

1 CD0 C CD0 USES ROUTINES

1 CD0 C CD0 XISUB

1 CD0 C CD0 XRBIT

1 CD0 C CD0 XRMOV

1 CD0 C CD0 XRMSG
1 CD5 XNWKB
2 CD5 XRESET
3 CD5 XDIS
4 C*********
5 /* XIAD PROCESSES THE INPUT LITERAL LIST
6 1 BEGIN XIAD
7 DO UNTIL EOS IS REACHED, OR
8 UNTIL AN ERROR OCCURS
9 IF THIS IS A DATA ELEMENT, THEN
10 (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
11 IF ARG. TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
12 ERREXIT IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
13 SET LENGTH TO BE MOVED (LENMOV) TO EFFECTIVE LENGTH FOR DATA
14 OF ARG'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
15 ELSE
16 IF ARG. TYPE IS FREE, THEN
17 SET LENGTH TO BE MOVED (LENMOV) TO EFFECTIVE LENGTH
18 FOR DATA INPUT
19 ELSE THIS MUST BE CHARACTER DATA BEING INPUT
20 ERREXIT IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ER10:
21 INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
22 COMPUTE NO. WORDS IN INPUT CHARACTER STRING
23 ERREXIT IF NO. WORDS (LENMOV) > 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 - LENMOV + 1)
30 ERREXIT 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 BUFFERS TO LITERAL AREA
35 SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
36 ARGUMENT, TO LENMOV
37 IF LENMOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
38 MOVE LENMOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
39 ENDIF
40 INCREMENT TO NEXT TOKEN
41 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENMOV
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 XIAD IF ERROR (IRETC < 0)
49 ELSE
50 IF THIS IS AN ",", THEN
51 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
52 CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
53 ELSE
54 MARK ONE ELEMENT OF THIS ARGUMENT AT ISUB AS INCOMPLETE
55 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
56 LENGTH OF ONE ELEMENT (LENEFF)
57 ENDIF
58 TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
59 INCREMENT TO NEXT TOKEN
60 */
615 1 EXIT TO :RETURN:
616 2 :ERROR: CALL XRMSSG "INVALID SYNTAX"
617 2 :ERROR: CALL XRMSSG "ONLY DATA VALID TO RIGHT OF ="
618 2 :ERROR: CALL XRMSSG "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
619 2 :RETURN:
620 1 END XIDAT
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CD0</td>
<td>FORTRAN CALLING PROCEDURE</td>
</tr>
<tr>
<td>2</td>
<td>CD0</td>
<td>CALL XINPT</td>
</tr>
<tr>
<td>3</td>
<td>CD0</td>
<td>INTERFACE TABLE EDITOR'S INPUT PROCESSOR</td>
</tr>
<tr>
<td>4</td>
<td>CD0</td>
<td>INPUT</td>
</tr>
<tr>
<td>5</td>
<td>CD0</td>
<td>COMMON XE - COMBUF, COMTR, TOKENS</td>
</tr>
<tr>
<td>6</td>
<td>CD0</td>
<td>COMMON XB - DIRECT, IOFLG, ISUB, MARE, MDIR, NUMRNG, PRNTMD, WKBUF</td>
</tr>
<tr>
<td>7</td>
<td>CD0</td>
<td>COMMON XB - ARGNO, IRET</td>
</tr>
<tr>
<td>8</td>
<td>CD0</td>
<td>NOTES</td>
</tr>
<tr>
<td>9</td>
<td>CD5</td>
<td>USES ROUTINES</td>
</tr>
<tr>
<td>10</td>
<td>CD5</td>
<td>XIDAF</td>
</tr>
<tr>
<td>11</td>
<td>CD5</td>
<td>XIEFX</td>
</tr>
<tr>
<td>12</td>
<td>CD5</td>
<td>XILSF</td>
</tr>
<tr>
<td>13</td>
<td>CD5</td>
<td>XIPAM</td>
</tr>
<tr>
<td>14</td>
<td>CD5</td>
<td>XIMF</td>
</tr>
<tr>
<td>15</td>
<td>CD5</td>
<td>XRCPR</td>
</tr>
<tr>
<td>16</td>
<td>CD5</td>
<td>XRMV</td>
</tr>
<tr>
<td>17</td>
<td>CD5</td>
<td>XRMSC</td>
</tr>
<tr>
<td>18</td>
<td>CD5</td>
<td>XRMWB</td>
</tr>
<tr>
<td>19</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>20</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>21</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>22</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>23</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>24</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>25</td>
<td>CD0</td>
<td>C*******</td>
</tr>
<tr>
<td>26</td>
<td>CD0</td>
<td>C*******</td>
</tr>
</tbody>
</table>
1 * XINPT PROCESSES THE USER'S INPUT TEXT
2 BEGIN XINPT
3 IF PRMTD = 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 '"', THEN
10 SET IFLAG TO 10
11 POSITION TO NEXT TOKEN
12 ELSE
13 SET IFLAG TO 1
14 ENDIF
15 ELSE
16 IF TOKEN IS 'b', THEN
17 SET IFLAG TO 0
18 ENDIF
19 ENDIF
20 IF IFLAG NOT SET, THEN
21 CASE NAME (:EXIT:, :PROMPT:, :LIST:)
22 ERREXIT IF ANOTHER TOKEN FOLLOW :ERROR2:
23 :EXIT: SET IRET0 SO THAT PROMPTING LOOP TERMINATES
24 :PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE
25 :LIST: CALL XLST TO PROCESS LIST DIRECTIVE
26 ENDCASE
27 ENDIF
28 START SEARCH UNTIL BP ENTRIES
29 EXIT IF NAME FOUND IN PROMPT TABLE
30 SET AENO TO ENTRY NO.
31 SET ISUB TO 1
32 ORELSE
33 INCREMENT TO NEXT PROMPT TABLE ENTRY
34 ENDCASE
35 ERREXIT :ERROR10:
36 ENDOSEARCH
37 ERREXIT IF IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:
38 ENDIF
39 IF NEXT TOKEN IS A NAME, THEN
40 CALL XIPAR TO PROCESS A PARAMETER FIELD
41 ELSE
42 ERREXIT IF IFLAG IS NOT I ("=") :ERROR8:
43 CALL XDAT TO PROCESS DATA LIST
44 ENDIF
45 1 EXIT XINPT
46 1 EXIT TO :RETURN:
47 2 :ERROR2: CALL XMSG "INVALID SYNTAX"
48 2 :ERROR8: CALL XMSG "MUST USE PARAMETER NAME TO RIGHT OF & OR =&"
49 2 :RETURN:
50 1 END XINPT
714 1 COO   FORTRAN CALLING PROCEDEUR
715 1 COO   CALL XIPAR
716 1 COO   CALL XIPAR
717 1 COO   CALL XIPAR
718 1 COO   CALL XIPAR
719 1 COO   CALL XIPAR
720 1 COO   CALL XIPAR
721 1 COO   CALL XIPAR
722 1 COO   CALL XIPAR
723 1 COO   CALL XIPAR
724 1 COO   CALL XIPAR
725 1 COO   CALL XIPAR
726 1 COO   CALL XIPAR
727 1 COO   CALL XIPAR
728 1 COO   CALL XIPAR
729 1 COO   CALL XIPAR
730 1 COO   CALL XIPAR
731 1 COO   CALL XIPAR
732 1 COO   CALL XIPAR
733 1 COO   CALL XIPAR
734 1 COO   CALL XIPAR
735 1 COO   CALL XIPAR
736 1 COO   CALL XIPAR
737 1 COO   CALL XIPAR
738 1 COO   CALL XIPAR
739 1 COO   CALL XIPAR
740 1 COO   CALL XIPAR
741 1 COO   CALL XIPAR
742 1 COO   CALL XIPAR
743 1 COO   CALL XIPAR
744 1 COO   CALL XIPAR
745 1 COO   CALL XIPAR
746 1 COO   CALL XIPAR
747 1 COO   CALL XIPAR
748 1 COO   CALL XIPAR
749 1 COO   CALL XIPAR
1 * XIPAR PROCSES A USER SPECIFIED PARAMETER FIELD
2 BEGIN XIPAR
3 IF A NAME IS SPECIFIED, THEN
4 INCANT TO NEXT TOKEN
5 IF TOKEN IS '(', THEN
6 PROCESS I AND J SUBSCRIPTS
7 EXIT IF INVALID SUBSCRIPTING :ERR14:
8 IF DOUBLY SUBSCRIBED, 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 ENDIF
15 ELSE
16 SET LITOSP IN ARGNO'S SPECS TO 0
17 ENDIF
18 ERREXIT IF ORDE HAS MORE THAN A CHARACTERS :ERR18:
19 ERREXIT IF EXTRANOUS FIELD INPUT :ERR02:
20 TURN OF B-FLAG (SAME 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 ENDIF
26 ELSE
27 $B INPUT
28 "REXIT IF NOT AN AMPERSAND ($) INPUT :ERR02:
29 CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS
30 SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
31 ENDIF
32 EXIT TO :RETURN:
33 ERRO2: CALL XMESS - "INVALID SYNTAX"
34 ERRO14: CALL XMESS - "INVALID SUBSCRIPT SYNTAX"
35 ERRO18: CALL XMESS - "INVALID ORDE NAME"
36 RETURN:
37 END XIPAR
FORTRAN CALLING PROCEDURE

CALL XILSD

*********

LIST DATA OR PARAMETER VALUES FOR ONE ARGUMENT

INPUT

 COMMON XE - ..J

 COMMON XB - I:K.., IDLE, IARG, IDIN,

 COMMON XS - J:K.., IDLE, IARG, IDIN,

 COMMON XT - L:K.., IDLE, IARG, IDIN,

 COMMON XS - BUFFER = LINE TO BE OUTPUT

BUFFTR = INDEX INTO BUFFER FOR NEXT ASCII DATA

KETRM = RETURN INDICATOR FOR INTERNAL Routines

*********

INTERNAL VARIABLES

*********

NOTES

 *********

USES ROUTINES

*********

EXEC

*********

XREP1

*********

XREP2

*********

XREP3

*********

XREP4

*********

XREP5

*********

XREP6

*********

XREP7

*********

XREP8

*********

XREP9

*********

XREP10

*********
9 * XILSD WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
34 * IS RETURNED AS A PROMPT.
35 BEGIN XILSD
36 SET ARGUMENT NAME INTO BUFFER
37 USE TOFLAG TO DETERMINE WHICH OF 'B', 'N', OR '2'
38 WILL GO INTO THE PRINT BUFFER
39 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
40 IF A PARAMETER NAME IS SPECIFIED, THEN
41 PUT PARAMETER NAME INTO BUFFER
42 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
43 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
44 PUT SUBSCRIPT INTO BUFFER
45 ELSE
46 IF LITDSP OF ARGUMENT IS > 0, THEN
47 COMPUTE AND CONVERT THIS SUBSCRIPT
48 PUT SUBSCRIPT INTO BUFFER
49 ENDFI
50 ENDFI
51 WRITE OUT THE PRINT BUFFER BUILT
52 ENDFI
53 ELSE
54 LOCATE LITERAL LIST AND MASK
55 IF SYMBOLIC STRING, THEN
56 CALL FILES TO PRINT SYMBOLIC STRING
57 ELSE
58 DO UNTIL ALL ELEMENTS PROCESSED
59 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
60 UNTIL ALL ELEMENT'S PROCESSED
61 COMPUTE AND CONVERT THE SUBSCRIPT
62 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
63 PUT "A" INTO BUFFER
64 ELSE
65 CONVERT THE DATA USING XR06, XRE14, OR XR16
66 PUT DATA AND "A" INTO BUFFER
67 ENDFI
68 ENDDO
69 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
70 REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
71 ENDFI
72 WRITE OUT THE PRINT BUFFER BUILT
73 ENDDO
74 ENDFI
75 ENDFI
76 ENDFI
77 ENDFI
FORTRAN CALLING PROTOCOLD

CALL XILSS

XILSS IS CALLED BY XILSS TO LIST SYMBOLIC STRING DATA

INPUT

COMMON XE - LU
COMMON XB - DEBUG, LSLTY, WKRUS
COMMON XS - BUFFER = PRINT LINE BUFFER ALREADY INITIALIZED WITH NAME
BUFFTR = INDEX INTO BUFFER OF NEXT POSITIX)
DATPTR = INDEX INTO WKBUS OF SYMBOLIC STRING DATA

OUTPUT

COMMON XS - BUFFER, BUFFTR, DATPTR

END
### INTERNAL VARIABLES

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>TOKEN</th>
<th>WORD 1</th>
<th>WORD 2</th>
<th>WORD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>906</td>
<td>CD4</td>
<td>INTEGER</td>
<td>3</td>
<td>-1=&gt;CALL X816</td>
</tr>
<tr>
<td>907</td>
<td>CD4</td>
<td>REAL</td>
<td>7</td>
<td>-2=&gt;CALL X814</td>
</tr>
<tr>
<td>908</td>
<td>CD4</td>
<td>DOUBLE</td>
<td>9</td>
<td>-3=&gt;CALL X8018</td>
</tr>
<tr>
<td>909</td>
<td>CD4</td>
<td>NAME</td>
<td>3</td>
<td>-4=&gt;USE 3 WORDS</td>
</tr>
<tr>
<td>910</td>
<td>CD4</td>
<td>--------</td>
<td>0</td>
<td>-5=&gt;ERROR (INVALID)</td>
</tr>
<tr>
<td>911</td>
<td>CD4</td>
<td>--------</td>
<td>0</td>
<td>-6=&gt;ERROR (INVALID)</td>
</tr>
<tr>
<td>912</td>
<td>CD4</td>
<td>--------</td>
<td>0</td>
<td>-7=&gt;ERROR (INVALID)</td>
</tr>
<tr>
<td>913</td>
<td>CD4</td>
<td>CHAR. STR.</td>
<td>-8=&gt;USE VALUE</td>
<td>5=&gt;USE SIZE - #- -9=&gt;USE SIZE+2 WORDS</td>
</tr>
<tr>
<td>914</td>
<td>CD4</td>
<td>--------</td>
<td>0</td>
<td>BEGINNING</td>
</tr>
<tr>
<td>915</td>
<td>CD4</td>
<td>TOKEN</td>
<td>0</td>
<td>O=&gt;ERROR (INVALID)</td>
</tr>
<tr>
<td>916</td>
<td>CD4</td>
<td>--------</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>917</td>
<td>CD4</td>
<td>+</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>918</td>
<td>CD4</td>
<td>-</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>919</td>
<td>CD4</td>
<td>/</td>
<td>1</td>
<td>/</td>
</tr>
<tr>
<td>920</td>
<td>CD4</td>
<td>1</td>
<td>1</td>
<td>&gt;</td>
</tr>
<tr>
<td>921</td>
<td>CD4</td>
<td>&gt;</td>
<td>1</td>
<td>&gt;</td>
</tr>
<tr>
<td>922</td>
<td>CD4</td>
<td>#</td>
<td>1</td>
<td>#</td>
</tr>
<tr>
<td>923</td>
<td>CD4</td>
<td>&amp;</td>
<td>1</td>
<td>&amp;</td>
</tr>
<tr>
<td>924</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>925</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>926</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>927</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>928</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>929</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>930</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>931</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>932</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>933</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>934</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>935</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>936</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>937</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>938</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>939</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>940</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>941</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>942</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>943</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>944</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>945</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>946</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>947</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>948</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>949</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>951</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>952</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>953</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>954</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>956</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>957</td>
<td>CD4</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
1 C**nnes
2 CD5
3 CD5
4 CD5
5 CD5
6 CD5
7 CD5
8 CD5
9 CD5
10 CD5
11 CD5
12 CD5
13 CD5
14 CD5
15 CD5
16 CD5
17 CD5
18 CD5
19 CD5
20 CD5
21 CD5
22 CD5
23 CD5
24 CD5
25 CD5
26 CD5
27 CD5
28 CD5
29 CD5
30 CD5
31 CD5
32 CD5
33 CD5
34 CD5
35 CD5
36 CD5
37 CD5
38 CD5
39 CD5
40 CD5
41 CD5
42 CD5
43 CD5
44 CD5
45 CD5
46 CD5
47 CD5
48 CD5
49 CD5
50 CD5
51 CD5
52 CD5
53 CD5
54 CD5
55 CD5
56 CD5
57 CD5
58 CD5
59 CD5
60 CD5
61 CD5
62 CD5
63 CD5
64 CD5
65 CD5
66 CD5
67 CD5
68 CD5
69 CD5
70 CD5
71 CD5
72 CD5
73 BEGINXLSS
74 MOVE A * CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
75 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
76 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
77 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
78 EXIT TO ERROR 2 IF FIELD = 0
79 IF SIZE < 0, THEN
80 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
81 ENDIF
82 IF TOKSIZ < 0, THEN
83 SET TOKSIZ TO SIZE + 2
84 ENDIF
85 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
86 CALL EXEC TO WRITE PREVIOUS NUMBER OF WORDS TO BLANKS
87 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
88 CLEAR PRINT BUFFER TO BLANKS
89 ENDIF
90 IF FIELD > 0, THEN
91 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
92 ELSE
93 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
94 :ONE: CALL XR16 WITH VALUE IN NEXT WORD OF SYMB. STRING
95 AND PUT RESULTS INTO PRINT BUFFER
96 :TWO: CALL XR14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
97 AND PUT RESULTS INTO PRINT BUFFER
98 :THREE: CALL XR18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
99 AND PUT RESULTS INTO PRINT BUFFER
100 :FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
101 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
102 :SIX: CALL XR16 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
103 INTO PRINT BUFFER FOLLOWED BY AN "R"
104 :EXIT: PUT A * CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
105 INDEX BY 1
106 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
107 EXIT XLSS
108 ENDCASE
109 INCREMENT PRINT BUFFER INDEX BY SIZE
1015 1 CDO  FORTRAN CALLING PROCEEDURE
1016 1 CDO
1017 1 CDO
1018 1 CDO  CALL XICHR
1019 1 CDO
1020 1 C********
1021 1 CDO
1022 1 CDO  PRINT THE CHARACTERISTICS OF AN ARGUMENT
1023 1 CDO
1024 1 C********
1025 1 CDO
1026 1 CDO  INPUT
1027 1 CDO
1028 1 CDO  COMMON XE - LU
1029 1 CDO
1030 1 CDO  COMMON XB - ARGNO, IFLAG, ISIZE, ITYPE,
1031 1 CDO              LENEFF, NARG
1032 1 CDO
1033 1 C********
1034 1 CDO
1035 1 CDO  NOTES
1036 1 CDO
1037 1 CDO  USES ROUTINES
1038 1 CDO
1039 1 CDO  EXEC
1040 1 CDO  XRPCK
1041 1 CDO  XRMOB
1042 1 CDO  XRMOB
1043 1 CDO  XRMOB
1044 1 CDO
1045 1 C********
1046 1 C** WRITE ARGUMENT CHARACTERISTICS
1047 1 C** BEGIN XICHR
1048 1 C** BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPTS, I/O TYPE AND
1049 1 C** DATA TYPE
1050 1 C** WRITE OUT THE PRINT BUFFER
1051 1 C** END XICHR
FORTRAN CALLING PROCEDURE
CALL XIEXIT

**EXTRACT VARIOUS FIELDS OF AN ARGUMENT CHARACTERISTICS**
AND PUT VALUES INTO COMMON

**INPUT**
COMMON XB - ARGNO, ISISZES, WKBUF

**OUTPUT**
COMMON XB - CFLAG, DFLAG, IARG, JARG,
ICLASS, IDIM, IFLAG, ISISZ,
ISUB, ITYPE, LENEFF, LITDSP,
LITSZ, MDIBTM, NMOBTM, SFLAG

**NOTES**
USES ROUTINES
IAMD XIEXIT

**EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS ARGUMENT**
BEGIN XIEXIT
USING THE ARGUMENT NO. (ARGNO),LOCATE THIS ARGUMENT'S CHARACTERISTICS
IN THE WORKING BUFFER
EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
END XIEXIT
FORTRAN CALLING PROCEDURE

CALL XILIT

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

INPUTS

COMMON X8 - ARGNO, DFLAG, IARG4, ISIZE,
LENF, LITOSP, LITDN, LITPTR,
LITSZ, MARG, NOXBTM, NOBITM
NUMARG, SFLAG, WBLNG, WBUF

OUTPUTS

COMMON X8 - LITDN, LITLEN, LITPTR, WBUF

INTERNAL VARIABLES

COMMON X8 - LITUP = INDEX INTO WBUF OF AREA FOR NEXT LITERAL
ENTRY TO BE MOVED INTO
LITUP = INDEX INTO WBUF OF LITERAL ENTRY TO BE
COMPRESSED AND MOVED

NOTES

USES ROUTINES

XIEXT
XBIT
XMOV
XRNXB
XRSET
1145 1 * PACK LITERAL AREA INTO FORMAT FOR STORAGE OF INTERFACE TABLE
1146 1 BEGIN XILIT
1147 2 DO UNTIL ALL LITERAL AREAS PROCESSED
1148 3    DO UNTIL ALL ARGUMENTS SEARCHED
1149 4      IF THIS LITERAL ENTRY BELONGS TO THIS ARGUMENT, THEN
1150 5      IF ALL ELEMENTS OF THIS ARGUMENT ARE COMPLETE, THEN
1151 6      MOVE ALL DATA FOR LITERAL ENTRY UP IN WORKING BUFFER
1152 7      ELSE
1153 8      DO UNTIL ALL BITS OF BIT MASK PROCESSED
1154 9      IF THE BIT IS ON, THEN
1155 10     MOVE CORRECT NUMBER OF WORDS (LEN_EFF) OF LITERAL
1156 11     UP IN THE WORKING BUFFER
1157 12     ENDEF
1158 13 ENDDO
1159 14 ENDIF
1160 15 CALCULATE NEW DISPLACEMENT AND SET IN LITOSP
1161 16 ENDEF
1162 17 ENDDO
1163 18 ENDDO
1164 1 END XILIT
CD**********
CD
CD RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:
CD :RU,XLMAN,LU,O,ISECU,FLAGS
CD**********
CD
CD OFFLINE MAINTENANCE PROGRAM, XLMAN CREATES, DELETES AND
CD MODIFIES FDS FILES.
CD**********
CD
CD INPUTS FROM RUN SEQUENCE
CD
CD LU - LOGICAL UNIT NUMBER WHERE USER DESIRES
CD HIS INPUTS/OUTPUTS
CD ISECU - SECURITY CODE OF FDS LIBRARY FILES
CD FLAGS - DEBUG FLAG: 0 - OFF
CD 4 - ON
CD
CD**********
CD
CD INTERNAL VARIABLES:
CD
CD PROMT - (INTEGER, 31 WORDS) PROMPT FOR OPTION
CD TO BE EXECUTED
CD TEMP - (INTEGER, 1 WORD) TEMPORARY USED FOR RESPONSE
CD TOKEMS - (INTEGER, 29 WORDS) THE 29 TOKEMS TO BE
CD INITIALIZED INTO COMMON
CD
CD**********
CD
CD RTE FUNCTIONS AND SUBROUTINES USED:
CD
CD EXEC,RMPAR
CD
CD FDS FUNCTIONS AND ROUTINES USED:
CD
CD XELDS, XLDIF, XIDEL, XIMLT, XIMOD, XLMG,
CD XLPAM, XLPAMD, XTCOM, XTMOD, XTMG
CD
CD XE COMMON USED:
CD
CD EQUIVALENCE (XE(1), LU ),
CD + (XE(2), ICLASS), (XE(3), ISECU ),
CD + (XE(4), FLAGS ), (XE(5), MASST ),
CD + (XE(6), SUBSTA ), (XE(7), NUMER ),
CD + (XE(8), VALFLG ), (XE(9), TOKEMS ),
CD + (XE(142), ICR ), (XE(143), COMBUF )
CD**********
BEGIN XLMAN
CALL RMPLR TO GET INPUT PARAMETERS
INITIALIZE COMMON TO ZERO
SET CLASS NUMBER TO ZERO
CALL EXEC TO GET A CLASS NUMBER
EXIT XLMAN IF SECURITY CODE IS NOT VALID
CALL XMOV TO INITIALIZE TOKENS IN COMMON
DO FOREVER
PROMPT:
INITIALIZE MASTER AND SUBSTATE FLAGS
CALL XCMC TO PROMPT FOR OPTION
IF XCMC RETURN CODE IS NOT ZERO OR
FIRST TOKEN IS NOT AN INTEGER OR
INTEGER > 7 THEN
CALL XRMSE TO WRITE INVALID RESPONSE
GO TO :PROMPT:
ENDIF
CASE INTEGER (XLPRM:,XLPRM:,XLPRM:,XLPRM:,XLPRM:,XLPRM,:
:XLPRM:,XLPRM:,XLPRM:,XLPRM:,XLPRM:,EXIT):
:XLPRM:
SET NUMBER TO INTEGER
CALL XELS TO LOAD XLPRM TO CREATE SYSTEM PROMPT FILE
:XLPRO:
SET VALFLG TO SAY ORGINAL XLPRO REQUEST
CALL XELS TO LOAD XLPRO TO ADD A PROCESSOR
DO UNTIL VALFLG SAYS EXIT (X)
CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
CALL XELS TO LOAD XLPRO TO COMPLETE PROCESSING
ENDDO
:XLDEL:
CALL XELS TO LOAD XLDEL TO DELETE A PROCESSOR
:XLMOD:
SET VALFLG TO SAY ORGINAL XLMOD REQUEST
CALL XELS TO LOAD XLMOD TO MODIFY A PROCESSOR
DO UNTIL VALFLG SAYS EXIT (X)
CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
CALL XELS TO LOAD XLMOD TO COMPLETE PROCESSING
ENDDO
:XLMSG:
CALL XELS TO LOAD XLMSG TO ADD A MESSAGE
:XLDBF:
CALL XELS TO LOAD XLDBF TO HANDLE DATA BASE FILES
ENDCASE
ENDDO
EXIT:
CALL EXEC TO RELEASE CLASS NUMBER
END XLMAN
FORTRAN CALLING PROCEDURE:
CALL XELS ('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 - XLPRM 2 - XLPRM 3 - XLPRM

INTERNAL VARIABLES:
COM3 - (INTEGER, 1 WORD) FIRST TOKEN IN COMBUF
COM4 - (INTEGER, 1 WORD) FIRST DATA IN COMBUF
MSG - (INTEGER, 1 WORD) CONTAINS APPROPRIATE MESSAGE NUMBER
WITH WHICH TO CALL XMSG
NAME - (INTEGER, 9 WORDS) 5 ELEMENT ARRAY, EACH ELEMENT IS A SYSTEM PROMPT FILE NAME
MODIR - (INTEGER, 1 WORD) NUMBER OF DIRECTIVES
NOTOK - (INTEGER, 1 WORD) NUMBER OF TOKENS
PROMS - (INTGERS) PROM2,PROM3,PROM4 ARE ALL USER PROMPT ARRAYS

RTE FUNCTIONS AND SUBROUTINES USED:
KCVT,CLOSE,CREATE,PURGE,WAITF

FRS FUNCTIONS AND ROUTINES USED:
XRMOV,XRMSG,XTCOM

COMMON USED:
ERUVALENCE
+(X(3),ISECU),(X(7),NUMBR),
+(X(145),XCR),(X(145),COMBUF),
+(X(145),NOTOK),(X(147),COMS),
+(X(148),COM4),(X(1),IBUF),
+(X(128),MODIR)
160 1 BEGIN XLPRM
161 2 :PRM1:
162 3 CALL XICOM TO PROMPT USER FOR # DIRECTIVES
163 4 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
164 5 CALL CREAT TO CREATE PROMPT FILE
165 6 IF FILE ALREADY EXISTS THEN
166 7 EXIT XLPRM IF RESPONSE IS CANCEL
167 8 CALL PURGE TO PURGE PROMPT FILE
168 9 ERREXIT IF PURGE ERROR TO :FILEERR:
170 10 GO TO :PRM1:
171 11 ELSE (CREATE NEW FILE)
172 12 ERREXIT IF CREATE ERROR TO :FILEERR:
173 13 CALL XICOM TO PROMPT USER FOR LIST OF DIRECTIVES
174 14 ERREXIT IF LIST IS INCONSISTENT WITH # OF DIRECTIVES TO :ERR1:
175 15 DO FOR EACH DIRECTIVE
176 16 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
177 17 CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
178 18 ENDDO
179 19 CALL WRITF TO WRITE LIST OF DIRECTIVES TO FILE
180 20 ERREXIT IF WRITF ERROR TO :FILEERR:
181 21 DO FOR EACH DIRECTIVE
182 22 CALL XICOM TO PROMPT USER FOR DEFINITION
183 23 DO UNTIL E0S IS REACHED IN RESPONSE
184 24 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185 25 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186 26 MOVE RESPONSE INTO BUFFER
187 27 SET CONTROL CHARACTERS IN BUFFER
188 28 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
189 29 ENDDO
190 30 SET REMAINDER OF BUFFER TO NULL
191 31 CALL WRITF TO WRITE DEFINITION
192 32 ERREXIT IF WRITF ERROR TO :FILEERR:
193 33 ENDIF
194 34 CALL CLOSE TO CLOSE FILE
195 35 ERREXIT IF CLOSE ERROR TO :FILEERR:
196 36 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
197 37 2 ENDIF
198 3 1 EXIT XLPRM
199 2 :ERR1: (ERROR IN RESPONSE)
200 3 CALL XMSG TO DISPLAY ERROR IN RESPONSE
201 4 RETURN TO XPRFAST FOR ANOTHER RESPONSE
202 2 :FILEERR: (FILE ACCESS ERROR)
203 3 CALL XMSG TO DISPLAY ERROR
204 4 CALL CLOSE TO CLOSE FILE
205 5 CALL PURGE TO PURGE FILE
206 6 1 END XLPRM
FORTRAN CALLING SEQUENCE:

CALL KEYS ('XLPRO')

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

INPUT FROM COMMON:

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

INTERNAL VARIABLES:

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

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

TO INTERF ACE TABLE FILE

DIRECT - (INTEGER, 6 WORDS) 3 NAME ARRAY CONTAINING VALID DIRECTIVES FOR THE INTERFACE TABLE EDITOR

NAME - (INTEGER, 3 WORDS) COMBINATION OF PROCESSOR NAME AND VERSION NUMBER

PRMAH - (INTEGER, 3 WORDS) PROCESSOR NAME

PROMS - (INTEGER) PROMPTS FOR USER TO BE PROMPTED WITH

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

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

RTE AND FRGR Routines USER

EXEC, FAND, KEVT, CREAT, OPEN,
READ, WRITF, CLOSE, PURGE

FDS ROUTINES USED:

XLIF, XLIM, XLPIL, XLPN,
XEPP, XEROM, XRMSE, XRPCC, XRSET, XRUPK, XTCON

COMMON USED:

(EQ3), (ISECU),

(XE(3), XE(6),)

(XE(7), XE(16),)

(XE(85), XE(86),)

(XE(97), XE(113),)

(XB(26), XB(37),)

(XB(36), XB(40),)
275 1 BEGIN XLPRO
276 2 IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
277 3 INITIALIZE MASTER AND SUB STATES
278 4 PERFORM LOAD TO UPDATE LIBRARY DIRECTORY
279 5 IF PROCESSOR HAS AN INTERFACE TABLE THEN
280 6 SET RETN = 2
281 7 :PRNPT2:
282 8 CALL XTCOM TO PROMPT FOR 0 PARAMETERS
283 9 ERREXIT IF XTCOM RETURN CODE IS NOT ZERO OR
284 10 ERREXIT IF 0 PARAMETERS NOT INTEGER 1-63 TO :PRNERR:
285 11 CREATE MEANEP ENTRY WITH 0 PARAMETERS AND PROCESSOR NAME
286 12 CALL XLPS TO WRITE INSTRUCTIONS FOR ENTERING SPEC
287 13 DO FOR 0 PARAMETERS
288 14 CALL XLPS To CREATE ONE PARAMETER ENTRY
289 15 ENDDO
290 16 SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
291 17 CALL XLPL To CREATE >PROMPT FILE
292 18 SET RETN = 3
293 19 :PRNPT3:
294 20 CALL XTCOM FOR DEFAULT VALUES DECISION
295 21 ERREXIT IF RETURN CODE IS NOT ZERO OR
296 22 ERREXIT IF RESPONSE IS NOT YE OR NO To :PRNERR:
297 23 IF RESPONSE WAS YE THEN
298 24 SET FLAG To CALL INTERFACE TABLE EDITOR
299 25 EXIT XLPRO
300 26 ENDFI
301 27 ELSE
302 28 SET CODES ARRAY TO ADD ONLY ABSTRACT
303 29 CALL XLPL To CREATE PROMPT FILE
304 30 PERFORM XLPRO - NO RETURN EXPECTED
305 31 ENDFI
306 32 ENDFI
307 33 CALL XLPL To CREATE THE DEFAULT INTERFACE TABLE: FILE
308 34 2 SET VALFLAG To SET ORIGINAL REQUEST To ADD A PROCESSOR
309 35 2 PERFORM XLPRO - NO RETURN EXPECTED
310 36 :PRNERR:
311 37 CALL XRNSG TC DISPLAY ERROR MESSAGE
312 38 GO To (:PRNPT1:,:PRNPT2:,:PRNPT3:), RETN
313 39 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 ENDIF
13 SET HLIN = 1
14 :PRMT1:
15 CALL XCOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
16 EXIT XLPRO IF RETURN CODE SAYS I ENTERED
17 ERREXIT IF RETURN CODE > ZERO OR
18 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
19 CALL XREMV TO MOVE PROCESSOR NAME INTO ENTRY
20 ERREXIT IF VERSION IS NOT INTEGER VALUE 0-127 TO :PRMERR:
21 CALL XRSSET TO SET VERSION IN ENTRY
22 ERREXIT IF INTERFACE TABLE OPTION IS NOT YE OR NO TO :PRMERR:
23 SET IT BIT = 0
24 IF RESPONSE IS YES THEN
25 SET IT BIT = 1
26 ENDIF
27 CALL XRSSET TO SET BIT ON/OFF
28 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
29 INCREMENT # PROCESSORS BY 1
30 CALL XREMV TO MOVE NEW ENTRY INTO XLIBD
31 IF # PROCESSORS > 1 THEN
32 CALL PURGE TO PURGE OLD FILE
33 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
34 ENDIF
35 CALL CREAT, WRITE AND CLOSE TO CREATE NEW LIBRARY DIRECTORY
36 ERREXIT IF FILE ERROR TO :FILERR:
37 ENDIF
38 1 END LIBD
FOURTEEN CALLING SEQUENCE:

CALL XLCDB

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

INPUTS IN COMMON:

XE(3) QUAL, XE(7) NUMBR, XE(8) SECU
XE(142) ICR, XE(5) OLDfil, XE(6) NEWfil
XE(9) TOTsz

INTERNAL VARIABLES:

XB(17) FREC - FIRST RECORD NUMBER IN DATBUF
XB(18) LREC - LAST RECORD NUMBER IN DATBUF
XB(40) IDC0 - DCB FOR OLDfil
XB(56) IDC0 - DCB FOR NEWfil
XB(200) TOCBUFF-BUFFER FOR COMPLETE DATA BASE TOC

RTE ROUTINES USED:

CLOSE, CREAT, KEVT, OPE,
PURGE, READF, WRITF

FDS ROUTINES USED:

XDBOD, XCRET, XRM5G

COMMON USED:

EQUIVALENCE (XE(3), QUAL );
+XE(7), NUMBR , XE(8), SECU
+XE(142), ICR , XB(3), OLDfil
+XB(6), NEWfil , XB(9), TOTsz
+XB(10), FILCHR , XB(12), QUALif
+XB(13), MOTOC , XB(14), SIZE
+XB(15), MGE , XB(16), TOCPTR
+XB(17), FREC , XB(18), LREC
+XB(39), JERR , XB(40), IDC0
+XB(50), IDC0 , XB(72), IBD

COMMON USED:

+XB(200), TOCBUFF
10  BEGIN LCLDB
11  CALL OPEN TO OPEN OLD FIL
12    ERREXIT IF OPEN ERROR TO :ERR:
13  CALL READ TO READ IN HEADER RECORD
14  ERREXIT IF READ ERROR TO :ERR:
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 :ERR:
19  ENDIF
20  SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
21  CALL CREATE TO CREATE NEWFIL
22  ERREXIT IF CREATE ERROR TO :ERR:
23  DO FOR EACH TOC ENTRY
24    IF REQUEST WAS PDF TO NEWFIL THEN
25      ERREXIT IF NAME > 4 CHARs OR
26      ERREXIT IF CLASS IS RDR AND NAME > 2 CHARs TO :ERR1:
27      APPEND PDF CONVENTION TO FRONT OF NAME
28      ELSE
29      REMOVE PDF CONVENTION FROM NAME
30  ENDIF
31  ENDDO
32  CALL WRITE TO WRITE NEW TOC RECORDS TO NEWFIL
33  ERREXIT IF WRITE ERROR TO :ERR:
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 CACHED (SIZE = 0)
38    SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
39    IF SIZE < LENGTH THEN
40      SET LENGTH TO SIZE
41  ENDIF
42  CALL READ TO READ LENGTH WORDS OF DATA
43  ERREXIT IF READ ERROR TO :ERR:
44  DECREMENT SIZE BY LENGTH READ
45  UPDATE FIRST AND LAST RECORD NUMBERS
46  STARSEARCH FOR TOCPTR = TOCPTR TO LAST TOC ENTRY
47  EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
48    IF CLASS IS AN INTERFACE TABLE THEN
49      COMPARE INDEX INTO DATABUF FROM FIRST RECORD #, RECORD # IN TOC ENTRY
50      AMND INDEX IN TOC ENTRY
51      SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
52  ENDIF
53  ENDFIND
54  CALL WRITE TO WRITE LENGTH WORDS OF DATA TO NEWFIL
55  ERREXIT IF WRITE ERROR TO :ERR:
56  ENDDO
CALL CLOSE TO CLOSE NEWFILE
CALL CLOSE TO CLOSE OLDFILE
CALL XDDDB TO DELETE OLDFILE FROM PDB LOG FILE
CALL PURGE TO PURGE OLDFILE FROM SYSTEM
EXIT XLCDB
:ERR1:
CALL CLOSE TO CLOSE NEWFILE
CALL PURGE TO PURGE NEWFILE
:ERR2:
CALL CLOSE TO CLOSE OLDFILE
:ERR3:
IF REQUEST WAS PDB TO MDG THEN
SET QUAL TO SAT DELETE MDG FILE
ELSE (REQUEST WAS MDG TO PDB)
SET QUAL TO SAT DELETE PDB FILE
ENDIF
CALL XDDBD TO DELETE MDG/PDB FROM LOG FILE
IF ERROR WAS FILE MANAGER THEN
CALL XRMSG TO DISPLAY ERROR AND RETURN CODE
ELSE
CALL XRMSG TO DISPLAY ERROR
ENDIF
END XLCDB
GO TO 100

100 CONTINUE

IF (ISEC) GOTO 100

CALL XE20F ('XECDF')

IF (ISEC) GOTO 100

CALL XE30F ('XECDF')
BEGIN XDBF:
  IF REQUEST IS TO CREATE/MODIFY LOG FILE THEN
    CALL OPEN TO OPEN MDB/PDB LOG FILE
    IF OPEN ERROR SAYS FILE NOT FOUND THEN
      CALL XLPCR TO CREATE MDB/PDB LOG FILE
    ELSE
      ERRexit IF OPEN ERROR TO :FILE ERR:
      CALL XLPMOD TO MODIFY MDB/PDB LOG FILE
    ENDIF
  ELSE
    DO UNTIL USER REQUESTS EXIT (X)
      CALL XTPM TO PROMPT USER FOR NAME AND USER ID
      IF RESPONSE IS NOT EXIT (PERCENT) THEN
        ERRexit IF RESPONSE IS INVALID TO :ERR1:
        SAVE 4 CHARACTER NAME AND ID IN COMMON
        IF REQUEST WAS PDB TO MDB THEN
          SET QUALIFIER TO SEARCH FOR PDB NAME
        ELSE (REQUEST WAS FOR MDB TO PDB)
          SET QUALIFIER TO SEARCH FOR MDB NAME
        ENDIF
        CALL XDBV TO VERIFY EXISTANCE OF MDB/PDB DEPENDING ON QUALIFIER
        ERRexit IF NAME WAS NOT FOUND TO :ERR1:
        ERRexit IF FILE MANAGER ERROR TO :FILE ERR:
        IF REQUEST WAS PDB TO MDB THEN
          SET QUALIFIER TO ADD PDB TO LOG FILE
        ELSE (REQUEST WAS MDB TO PDB)
          SET QUALIFIER TO ADD PDB TO LOG FILE
        ENDIF
        CALL XDBA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
        ERRexit IF DUPLICATE NAME OF
        ERRexit IF MAXIMUM NUMBER OF ENTRIES EXIST TO :ERR1:
        ERRexit IF FILE MANAGER ERROR TO :FILE ERR:
        IF REQUEST WAS PDB TO MDB THEN
          CALL XRSTR TO SET OLDQIL TO MDB NAME
          SET NEWQIL TO MDB NAME
        ELSE (REQUEST WAS MDB TO PDB)
          SET QIL TO MDB NAME
          CALL XRSTR TO SET NEWQIL TO PDB NAME
        ENDIF
        CALL XLCDF TO COPY OLDQIL TO NEWQIL
      ENDIF
    ENDU}
  ENDIF
END XDBF
588 1 CD**********
589 1 CD0
590 1 CD0  FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
591 1 CD0
592 1 CD0  CALL XELDS ('XLDEL')
593 1 CD0
594 1 CD**********
595 1 CD1
596 1 CD1  XLDEL DELETES A PROCESSOR FROM THE LIBRARY DIRECTORY AND
597 1 CD1  THE PROMPT FILE. IF THE PROCESSOR HAS AN INTERFACE TABLE,
598 1 CD1  IT DELETES THE DEFAULT INTERFACE TABLE FILE ALSO.
599 1 CD1
600 1 CD**********
601 1 CD4
602 1 CD4  INTERNAL VARIABLES:
603 1 CD4  COMMAN - (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, CREATE, OPEN, READF, WRITF, CLOSE, PURGE
613 1 CD5
614 1 CD5  FDS ROUTINES USED:
615 1 CD5  XRCPR, XREXT, XRMOV, XRMSG, XRPCK, XRSET, XRPK, XTCOM
616 1 CD5
617 1 CD5  XE AND XB COMMON USED
618 1 CD5
619 1 CD5  EQUVALENCE (XE(3), ISECU ),
620 1 CD5  + (XE(142), ICR ), (XE(143), COMBUF ),
621 1 CD5  + (X5(48), LIBD1 ), (X5(51), LIBD2 )
622 1 CD5
623 1 CD5
624 1 CD**********
626 1 BEGIN XLDEL

627 2 :PROMPT:
628 2 CALL XICOM TO PROMPT FOR PROCESSOR NAME
629 2 IF XICOM RETURN CODE IS NOT ZERO OR
630 3 RESPONSE IS NOT A VALID PROCESSOR NAME THEN
631 3 CALL XRMGR TO WRITE "ERROR MESSAGE"
632 2 GO TO :PROMPT:
633 2 EXIT XLDEL IF RETURN CODE SAYS % ENTERED
634 2 EMDIF
635 2 CALL OPEN, READ AND CLOSE TO READ IN LIBRARY DIRECTORY
636 2 ERROR IF FILE ERROR TO :FILERR:
637 2 IF PROCESSOR IS NOT IN LIBRARY DIRECTORY THEN
638 2 CALL XRMGR TO DISPLAY ERROR
639 2 GO TO :PROMPT:
640 2 EMDIF
641 2 DECREMENT # PROCESSES BY 1
642 2 CALL PURGE TO PURGE OLD LIBRARY DIRECTORY
643 2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
644 1 EXIT XLDEL IF # PROCESSES IS ZERO
645 2 CALL CREATE, WRITE AND CLOSE TO RECREATE LIBRARY DIRECTORY
646 2 ERREXIT IF FILE ERROR TO :FILERR:
647 2 CREATE THE PROMPT FILE NAME
648 2 CALL PURGE TO PURGE THE PROMPT FILE
649 2 ERREXIT IF PURGE ERROR TO :FILERR:
650 2 CALL XRMGR TO SAY FILE PURGED SUCCESSFULLY
651 2 IF PROCESOR HAD AN INTERFACE TABLE THEN
652 3 CREATE DEFAULT IF NAME
653 3 CALL PURGE TO PURGE DEFAULT IF
654 3 ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:
655 3 CALL XRMGR TO DISPLAY 'FILE NOT MGED' MESSAGE
656 2 EMDIF
657 2 GO TO :PROMPT:
658 2 :FILERR: CALL XRMGR TO WRITE FILE ACCESS ERROR
659 1 END XLDEL
FORTRAN CALLING SEQUENCE:

CALL XLIIF

XLIIIF CREATES THE DEFAULT INTERFACE TABLE FILE

INTERNAL VARIABLES

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

FDS ROUTINES USED:

XREX, XMSG

RTE ROUTINES USED:

CLOSE, CREAT, WRITF

COMMON USED:

EQUIVALENCE (X(3), ISECU),

+ (X(142), ICR ),

+ (X(90), LITPR ),

+ (X(91), LITLEN ),

+ (X(96), NOPARM ),

+ (X(101), MEDP ),

+ (X(108), PARM ),

+ (X(1), IERR ),

+ (X(2), IDCB )


**Fortran Calling Sequence:**

1 CD**********
2 CD
3 CD CALL XLNS
4 CD
5 CD**********
6 CD XLNS DISPLAYS INSTRUCTIONS FOR ENTERING PARAMETER SPECS
7 CD
8 CD**********
9 CD
10 CD INTERNAL VARIABLES:
11 CD
12 CD PROMPT - (INTEGER, 120 WORDS) ALL 7 LINES IN AN ARRAY
13 CD TO BE DISPLAYED
14 CD
15 CD**********
16 CD
17 CD RTE ROUTINES USED:
18 CD
19 CD EXEC
20 CD
21 CD COMMON USED:
22 CD
23 CD EQUIVALENCE (R(1), L(1))
24 CD
25 CD**********
26 CD
27 CD
28 CD
29 CD BEGIN XLNS
30 CD CALL EXEC TO DISPLAY ALL 7 LINES
31 CD END XLNS
BEGIN VERSION

SET RTN = 2

CALL XCTOM TO PROMPT FOR VERSION NUMBER

IF RETURN CODE IS NOT CR THEN

ERROR IF RETURN CODE IS NON-ZERO ON

ERROR IF VERSION IS INVALID TO :PRMRR:

IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN

CALL XRSET TO PUT NEW VERSION IN ENTRY

CALL OPEN, WRITE, CLOSE TO UPDATE LIBRARY DIRECTORY

ERROR IF FILE ERROR TO :FILEERR:

ENDIF

ENDIF

END VERSION

BEGIN DELPRM

SET RTN = 3

:PRM3:

DO UNTIL RETURN CODE IS CR ENTERED

IF NUMBER OF PARAMETERS > 1 THEN

CALL XCTOM TO PROMPT FOR DELETE PARAMETER NAME

IF RETURN CODE IS NOT CR ENTERED THEN

PERFORM RSPMD TO INTERPRET RESPONSE

PERFORM CHCMT TO CHECK FOR EXISTING DATA

SET ARCHO TH WORD IN CODES TO SAY 'DELETED'

CALL XRMov TO MOVE DATA TO DELETE PARAMETER

DECREMENT # PARAMETERS BY 1

ENDIF

ELSE

CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED

EXIT DELPRM

ENDIF

ENDO

END DELPRM

BEGIN MODPRM

SET RTN = 4

:PRM4:

DO UNTIL RETURN CODE IS CR ENTERED

CALL XCTOM TO PROMPT FOR MODIFY PARAMETER NAME

IF RETURN CODE IS NOT CR ENTERED THEN

PERFORM RSPMD TO INTERPRET RESPONSE

PERFORM CHCMT TO CHECK FOR EXISTING DATA

SET ARCHO TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'

CALL VLEMT TO PROMPT USER FOR SPECIFICATIONS

SET IT COMPLETE BIT OFF

ENDIF

ENDO

END MODPRM

END VERSION
959 1 BEGIN ADDPRM
960 2 SET RTN = 5

961 2 :PROMS:
962 2 DO UNTIL RETURN CODE IS CR ENTERED
963 3 IF NUMBER OF PARAMETERS < 43 THEN
964 4 CALL XTCOM TO PAROMPT FOR ADD PARAMETER BEFORE/AFTER PARAMETER NAME
965 4 IF RETURN CODE IS NOT CR ENTERED THEN
966 5 PERFORM RSPND TO INTERPRET RESPONSE
967 5 SET ARGNO TH NON-DELETED WORD IN CODES TO SAY 'ADDED'
968 5 INCREMENT NUMBER OF PARAMETERS BT 1
969 5 CALL XMID TO MOVE DATA TO MAKE SPACE FOR NEW PARAMETER
970 5 CALL XLSPS TO GET NEW SPECS FOR THIS PARAMETER
971 5 SET IT COMPLETE BIT OFF
972 4 ENDIF
973 3 ELSE
974 4 CALL XMSG TO DISPLAY NO MORE PARAMETERS CAN BE ADDED
975 3 EXIT ADDPRM
976 3 ENDIF
977 2 ENDDO
978 1 END ADDPRM
979 1 *
980 1 *
981 1 *
982 1 BEGIN DEFAULT
983 2 SET RTN = 7

984 2 :PROMS:
985 2 CALL XTCOM TO PORMAT FOR ADD/MODIFY/DELETE DEFAULT VALUES
986 2 CALL XTCOM TO PORMAT FOR ADD/MODIFY/DELETE DEFAULT VALUES
987 2 ERREXIT IF RETURN CODE IS NON-ZERO TO :PROMS:
988 2 IF RESPONSE IS YES THEN
989 3 SET VALFLAG TO SAY CALL INTERFACE TABLE EDITOR
990 2 EXIT XMOD
991 2 ENDDO
992 1 END DEFAULT
CD************
CD FORTRAN CALLING PROCEDURE
CD** CALL XE ( 'XLMSG' )
CD**
CD** XE ( 'XLMSG' ) PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XRMSG
CD**
CD****
CD** INPUT
CD** XE COMMON - LU, ISEQU, FLAGS, TOKENS, ICR
CD**
CD** TERMINAL - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES
CD** UPDATE MODE, MESSAGE NUMBER AND TEXT
CD**
CD** MESSAGE FILE - DIRECTORY AND OLD TEXT
CD**
CD** OUTPUT
CD** XE COMMON - COMBUF
CD**
CD** MESSAGE FILE - DIRECTORY AND TEXT UPDATES
CD**
CD** LOCAL
CD** AREA - NUMERICAL AREA INDICATOR FOR MESSAGE
CD** DIRECT - MESSAGE DIRECTORY (SEE SD 6.2.4.12)
CD** I - INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRAY
CD** IDCB - FILE MANAGER DATA CONTROL BLOCK
CD** IER - FILE MANAGER & XTCOM RETURN CODE
CD** NUM - MESSAGE NUMBER WITHIN MESSAGE AREA
CD** NBLK - BLOCK NUMBER WITHIN FILE
CD** CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION
CD** UPDATE MODE - NUMBER OF BLOCK CONTAINING MESSAGE
CD** RECPOS - MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 33, 65 OR 97)
CD**
CD****
CD** NOTES
CD** USES APOSW, CLOSE, CREAT, EXEC, IAND, KCVT, OPEN, READF, WRITF,
CD** XERIF, XKG, XRMSG, XTCOM, XUGBG
CD**
CD** WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE
CD** EXISTING TEXT IN PLACE.
CD** MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X
CD**
1067 1 BEGIN XLMSG
1068 2 OPEN XLMSG
1069 3 IF FILE NOT FOUND
1070 4 THEN
1071 5 OUTPUT "MESSAGE FILE CREATION"
1072 6 DO FOR EACH OF THE 32 DIRECTORY ENTRIES
1073 7 PROMPT FOR AREA ID AND MAXIMUM NUMBER OF MESSAGES FOR THIS ENTRY NUMBER
1074 8 IF RESPONSE WAS NULL
1075 9 THEN
1076 10 CLEAR ENTRY
1077 11 ELSE
1078 12 STORE ID
1079 13 COMPUTE AREA ORIGIN AND STORE
1080 14 CLEAR LAST MESSAGE NUMBER
1081 15 COMPUTE NUMBER OF BLOCKS AND STORE
1082 16 ENDF
1083 17 EMDDO
1084 18 CREATE A CLEARED FILE OF TOTAL REQUIRED SIZE
1085 19 ELSE
1086 20 READ DIRECTORY
1087 21 ENDF
1088 22 DO UNTIL USER INPUTS X
1089 23 PROMPT FOR MESSAGE NUMBER
1090 24 SEPARATE AREA AND MESSAGE NUMBER AND COMPUTE BLOCK NUMBER AND MESSAGE LOC
1091 25 IF VALID AREA AND BLOCK NUMBER <= NUMBER OF BLOCKS
1092 26 THEN
1093 27 READ BLOCK
1094 28 IF FIRST WORD OF MESSAGE IS NOT NULL (MESSAGE ALREADY EXIST)
1095 29 THEN
1096 30 DISPLAY OLD MESSAGE TEXT
1097 31 ENDF
1098 32 PROMPT FOR TEXT
1099 33 IF NON-NULL RESPONSE
1100 34 THEN
1101 35 STORE TEXT IN BLOCK
1102 36 REWRITE BLOCK
1103 37 IF MESSAGE NUMBER > LAST MESSAGE NUMBER
1104 38 THEN
1105 39 REPLACE LAST MESSAGE NUMBER WITH NEW NUMBER
1106 40 ENDF
1107 41 ENDF
1108 42 ELSE
1109 43 OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110 44 ENDF
1111 45 ENDDO
1112 46 REWRITE DIRECTORY BLOCK
1113 47 END XLMSG
1149 1 BEGIN XL CR
1150 2 CALL CREAT TO CREATE MDB/PDB LOG FILE
1151 2 ERREXIT IF CREATE ERROR TO :FILERR:
1152 2 RETURN
1153 2 INITIALIZE LOG RECORD BUFFER TO ZEROS
1154 2 SET # MDB FILES CURRENTLY USED TO ZERO
1155 2 SET MAXIMUM NUMBER MDBS TO 20
1156 2 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
1157 2 ERREXIT IF WRITF ERROR TO :FILERR:
1158 2 DO FOR EACH REMAINING LOG RECORD
1159 3 CALL XTCOM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
1160 4 SET MAXIMUM # PDB FILES TO RESPONSE
1161 4 SET # PRO FILES CURRENTLY USED TO ZERO
1162 3 ENDDO
1163 3 CALL WRITF TO WRITE 1 PDB RECORD TO LOG FILE
1164 3 ERREXIT IF WRITF ERROR TO :FILERR:
1165 2 ENDDO
1166 2 CALL CLOSE TO CLOSE FILE
1167 2 ERREXIT IF CLOSE ERROR TO :FILERR:
1168 1 EXIT XLPCR
1169 2 :FILERR:
1170 2 CALL XRMSG TO DISPLAY FILE ERROR
1171 2 CALL CLOSE TO CLOSE FILE
1172 1 END XLPCR
1174 1 CD************
1175 1 CD0
1176 1 CD0
1177 1 CD0
1178 1 CD0
1179 1 CD0
1180 1 CD************
1181 1 CD1
1182 1 CD0
1183 1 CD0
1184 1 CD1
1185 1 CD************
1186 1 CD2
1187 1 CD2
1188 1 CD2
1189 1 CD2
1190 1 CD2
1191 1 CD2
1192 1 CD2
1193 1 CD2
1194 1 CD2
1195 1 CD2
1196 1 CD2
1197 1 CD2
1198 1 CD2
1199 1 CD2
1200 1 CD2
1201 1 CD2
1202 1 CD2
1203 1 CD2
1204 1 CD************
1205 1 CD5
1206 1 CD5
1207 1 CD5
1208 1 CD5
1209 1 CD5
1210 1 CD5
1211 1 CD5
1212 1 CD5
1213 1 CD5
1214 1 CD5
1215 1 CD5
1216 1 CD5
1217 1 CD5
1218 1 CD5
1219 1 CD5
1220 1 CD5
1221 1 CD5
1222 1 CD5
1223 1 CD5
1224 1 CD5
1225 1 CD************

FORTRAN CALLING SEQUENCE:

CALL XLPFL (NOCOD, CODES)

NOCOD - NUMBER OF CODES IN THE CODE ARRAY
CODES - ARRAY OF CODES THAT REPRESENT:
CO"ES(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) PRCHN,
XE(142) ICR, XE(96) NOPARM, XE(108) PARMS

RTE FUNCTIONS USED:

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

FDS FUNCTIONS USED:

XERTM, XRPCP, IPMOV, IRMSSG,
XRPA, XRUP, XICOM

COMMON USED:

XERTM, ISECU, XE(16), PRCHN, XE(96), NOPARM,
XE(108), PARMS
1227 1 BEGIN XLPL
1228 2 COMPUTE SIZE OF FILE AS 0 PARAMETERS *3
1229 3 CALL CREAT TO CREATE PROMPT FILE
1230 4 ERREXIT IF CREAT ERROR TO :FILERR:
1231 5 STUFF SYNTAX RECORD 0 AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS
1232 6 CALL WRTIF TO WRITE SHORT PROMPT RECORD(s)
1233 7 ERREXIT IF WRTIF ERROR TO :FILERR:
1234 8 IF ABSTRACT CODE IS MODIFY THEN
1235 9 CALL OPEN TO OPEN OLD PROMPT FILE >XLTMP
1236 10 ERREXIT IF OPEN ERROR TO :FILERR:
1237 11 CALL READ TO READ EXISTING ABSTRACT
1238 12 ERREXIT IF READ ERROR TO :FILERR:
1239 13 CALL XTCOM TO DISPLAY EXISTING ABSTRACT
1240 14 ERREXIT IF WRTIF ERROR TO :FILERR:
1241 15 IF RESPONSE IS CR (NO MODIFICATION) THEN
1242 16 CALL WRTIF TO WRITE EXISTING ABSTRACT TO NEW FILE
1243 17 ELSE (NEW ABSTRACT WAS ENTERED)
1244 18 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1245 19 EDFIF
1246 20 ELSE (ABSTRACT CODE IS AD)
1247 21 CALL XTCOM TO PROMPT USER TO ENTER NEW ABSTRACT
1248 22 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1249 23 EDFIF
1250 24 EDFIF
1251 25 IF PARAMETERS IS NOT ZERO THEN
1252 26 PERFORM EXTPRM TO GET DEFINITIONS FOR EACH PARAMETER
1253 27 EDFIF
1254 28 CALL CLOSE TO CLOSE PROMPT FILE
1255 29 ERREXIT IF CLOSE ERROR TO :FILERR:
1256 30 IF THERE WAS AN OLD FILE THEN
1257 31 CALL CLOSE TO CLOSE OLD FILE >XLTMP
1258 32 CALL PURGE TO PURGE OLD FILE >XLTMP
1259 33 EDFIF
1260 1 EXIT XLPL
1261 2 :FILERR:
1262 3 CALL XRMESG TO DISPLAY ERROR CODE
1263 4 CALL CLOSE TO CLOSE NEW FILE
1264 5 CALL CLOSE TO CLOSE OLD FILE >XLTMP
1265 6 CALL PURGE TO PURGE OLD FILE >XLTMP
1266 7 SET VALFLG = 0
1267 8 CALL XCTN TO RETURN TO MAIN ***NO RETURN TO HERE***
1268 1 END XLPL
1270 1 BEGIN FORMAT
1271 2 BEGIN INITIALIZE 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 Xmov 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 : ERREXIT IF TOKEN POINTER TO NEXT CHARACTER STRING
1280 2 ENDDO
1281 3 SET REMAINDER OF BUFFER TO NULL
1282 4 CALL Writf TO WRITE NEW RESPONSE TO PROMPT FILE
1283 5 ERREXIT IF Writf ERROR TO :FILEERR:
1284 2 END FORMAT
1285 1 :ERR1:
1286 1 CALL Xmmsg TO DISPLAY INVALID RESPONSE
1287 1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
1299 1 BEGIN EXPRM
1300 2 DO FOR EACH ENTRY IN CODES ARRAY
1301 3 BEGIN CASE (:NHCHG:, :MOD:, :DEL:, :ADD:), CODES+1
1302 4 :NHCHG: (SPS HAD NO CHANGES)
1303 5 CALL READ TO READ EXISTING DEFINITION FROM >XLTMP
1304 6 ERREXIT IF READF ERROR TO :FILERR
1305 7 CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
1306 8 ERREXIT IF WRITE ERROR TO :FILERR
1307 4 :MOD: (SPS WERE MODIFIED)
1308 5 CALL READ TO READ EXISTING DEFINITION FROM >XLTMP
1309 6 ERREXIT IF READF ERROR TO :FILERR
1310 7 CALL EXEC TO DISPLAY EXISTING DEFINITION
1311 8 IF RESPONSE IS CR (NO RESPONSE) THEN
1312 9 CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
1313 10 ELSE (NEW DEFINITION WAS ENTERED)
1314 11 PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
1315 12 ENDIF
1316 4 :DEL: (SPS WERE DELETED)
1317 5 CALL POINT TO POSITION >XLTMP OVER THIS ENTRY
1318 6 ERREXIT IF POSN ERROR TO :FILERR
1319 4 :ADD: (SPS WERE ADDED)
1320 5 CALL XLTMP TO PROMPT USER TO ENTER NEW DEFINITION
1321 6 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1322 7 END CASE
1323 3 DO UNTIL RESPONSE IS CR (EXIT)
1324 4 EXIT EXPRM IF RESPONSE IS CR
1325 5 SEARCH SHOPT PROPTS FOR RESPONSE
1326 6 ERREXIT IF NOT FOUND TO :ERR:
1327 7 CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
1328 8 ERREXIT IF READF ERROR TO :FILERR
1329 9 CALL EXEC TO DISPLAY EXISTING DEFINITION
1330 4 CALL XLTMP TO PROMPT USER TO MODIFY DEFINITION
1331 5 IF RESPONSE IS POSITIVE
1332 6 CALL POINT TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
1333 7 ERREXIT IF POSN ERROR TO :FILERR
1334 8 PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
1335 9 ENDIF
1336 2 ENDOB
1337 1 END EXPRM
**FORTRAN CALLING SEQUENCE:**

CALL XLPMO

**INPUTS FROM COMMON:**

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

**INTERNAL VARIABLES IN COMMON:**

XE(51) UNO - NUMBER OF USER ID A-Z (1-26)
XE(52) RECMO - RECORD # WHERE UNO'S PDB LIST IS (2-94)
XB(52) PHAR - FILE NAME OF USER'S PDB (FULL 6 CHAR)"X"
XB(55) DIFF - NUMBER OF FILES THAT MUST BE DELETED
XB(244) OUTBUF-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

**RTE ROUTINES USED:**
CLOSE, EXEC, KEVT, PURGE, READF, WRITE

**FDS ROUTINES USED:**
XCPRA, XCREX, XRMOV, XMREE,
XRDFN, XRFRN, XRCOM

**COMMON USED:**

XRE(1), LU, XE(3), QUAL, XE(8), SECU, XE(142), ICA, XB(99), IERR, XB(100), INCB, XB(304), OUTBUF.
BEGIN XLPMO
DO UNTIL USER RESPONDS EXIT (2)
CALL XTCOM TO PROMPT USER FOR USER ID
ERROR IF RESPONSE IS INVALID TO :ERR1:
SAVE USER ID AND NUMBER (1-26) IN COMMON
CALL READ TO READ IN THE RECORD CONTAINING THIS ID
ERROR IF READ ERROR TO :FILERR:
DISPLAY MAX # ALLOWED FOR THIS USER AND # CURRENTLY USED
CALL XTCOM TO PROMPT USER FOR NEW MAXIMUM
IF RESPONSE IS A MODIFICATION (NOT CR) THEN
ERROR IF RESPONSE IS NOT VALID TO :ERR1:
COMPUTE DIFFERENCE AS # CURRENTLY USED - NEW MAXIMUM
IF DIFFERENCE <> 0 THEN
CALL WRITE TO REWRITE UPDATED RECORD TO FILE
ERROR IF WRITE ERROR TO :FILERR:
ELSE
DISPLAY LIST OF CURRENT FILES
SET \PURGED = 0
DO UNTIL DIFFERENCE <> 0 OR RESPONSE IS EXIT (PERCENT)
CALL XTCOM TO PROMPT USER TO DELETE \PURGED FILES
DO FOR EACH FILE NAME IN RESPONSE
SEARCH LIST FOR FILE NAME
ERROR IF INVALID NAME TO :ERR1:
MOVE FILE NAME TO PURGE LIST
INCREMENT \PURGED BY 1
DECREMENT DIFFERENCE BY 1
COMPARE OLD FILE NAME OUT OF LIST
ENDDO
ENDDO
IF RESPONSE WAS NOT EXIT THEN
CALL WRITE TO REWRITE UPDATED LOG RECORD
ERROR IF WRITE ERROR TO :FILERR:
DO FOR \PURGED FILES
CALL XTKFN TO CREATE FILE NAME
CALL PURGE TO PURGE FILE
ENDDO
ENDIF
ENDIF
END
CALL CLOSE TO CLOSE MDB/PDB LOG FILE
EXIT XLPMO
:ERR1:
CALL XMSG TO DISPLAY ERROR IN RESPONSE
RETURN TO REISSUE LAST PROMPT
:FILERR:
CALL XMSG TO DISPLAY FILE ERROR
CALL CLOSE TO CLOSE MDB/PDB FILE
END XLPMO
1474 1 BEGIN XLSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
1476 2 :PRMPT1:
1477 2 SET RETN TO 1
1478 2 CALL XTMOD TO PROMPT FOR PROMPT,CLASS,TYPE AND I/O FLAGS
1479 2 ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
1480 2 INITIALIZE THIS SHORT PROMPT AND ENTRY
1481 2 ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
1482 2 ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
1483 2 CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
1484 2 ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
1485 2 CALL XRSET TO SET CLASS IN ENTRY
1486 2 ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
1487 2 ERREXIT IF CLASS IS UDDE AND TYPE IS SYMBOLIC STRING TO :PRMERR:
1488 2 CALL XRSET TO SET TYPE IN ENTRY
1489 2 ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
1490 2 ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
1491 2 CALL XRSET TO SET I/O FLAGS IN ENTRY
1492 2 IF CLASS IS DATA ELEMENT THEN
1493 3 IF TYPE IS NOT SYMBOLIC STRING THEN
1494 4 SET RETN TO 2
1495 4 :PRMPT2:
1496 4 CALL XTMOD TO PROMPT FOR I AND J DIMENSIONS
1497 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1498 4 ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
1499 4 I; J DIMENSION IS NOT ENTERED THEN
1500 5 SET J DIMENSION TO 1
1501 5 ELSE
1502 5 ERREXIT IF J DIMENSION IS NOT VALID TO :PRMERR:
1503 5 IF J DIMENSION < 2 THEN
1504 6 SET J DIMENSION TO 1
1505 6 ELSE
1506 6 STORE I DIMENSION IN ENTRY
1507 6 :IF
1508 4 ENDIF
1509 4 SET SIZE = I DIMENSION * J DIMENSION * TYPE LENGTH
1510 4 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1511 3 ELSE
1512 4 SET RETN TO 3
1513 4 :J:
1514 4 CALL XTMOD TO PROMPT FOR MAXIMUM SIZE
1515 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1516 4 ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
1517 4 SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
1518 3 ENDIF
1519 2 ENDIF
1520 1 EXIT XLSPS
1521 2 :PRMERR: CALL XRMSGE TO DISPLAY ERROR MESSAGE
1522 2 GO TO (:PRMPT1,:PRMPT2,:PRMPT3) RETN
1523 1 END XLSPS
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
DO1  FDS MANAGER, SCHEDULED BY FDS CONFIGURATION MANAGER. ONE PROGRAM
DO1  PER SIGNED ON USER.
DO1

**********
**INPUT
**FDS STATUS TABLE (SEE SDD 6.2.2)
**FDS MANAGEMENT REQUESTS (SEE SDD 6.2.6.2)

**********
**OUTPUT
**FDS MANAGER RESPONSES (SEE SDD 6.2.6.3)

**********
**NOTES
**USES XMNG, XMAVA, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR, XPAR
**XMAVA IS A MANAGER GLOBAL DATA AREA CONTAINING THE AWA AND VARIOUS
**CONTROL VALUES

1
2 *
3 * BEGIN XMGR
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 *

INPUT:
1 = LU
2 = XVSTO ENTRY ADDRESS
3 = USER ID
4 = OPTIONS
5 = NUMBER OF DWA TRACKS

CALL RM?AR
LOCK THE MANAGER INTO THE PARTITION
SET XVSTA (FDS STATUS TABLE ENTRY ADDRESS)
RESOLVE XVSTO ADDRESS
CALL XMNG: INITIATE DWA FUNCTIONS
IF ERROR RETURN THEN
EXIT TO .XREND
ENDIF
GET EXEC'S CLASS NUMBER
GET PROCESSOR'S CLASS NUMBER
CALL SLIBR GET PRIVILEGED
SET EXEC'S CLASS NUMBER IN STBEC
SET PROCESSOR'S CLASS NUMBER IN STBPC
SET CURRENT TASK IN STBAT FROM EXEC ID STBEX
CALL SLIBX ENABLE
FORM EXEC'S NAME FROM 'KEY' & ASCII LU(STBLA)
SCHEDULE EXEC WITH WAIT
CALL XMNG
DO UNTIL EXEC REQUESTS TERMINATION OR ABENDS(P1=9 OR -32768)

IF REQUEST IS IN P1
IF REQUEST IS FOR AWA MANAGEMENT (P1=1) THEN
PERFORM XMNG
ELSE
IF REQUEST IS FOR SEQUENCE TABLE EXECUTION (P1=2) THEN
PERFORM 'MXS'
ELSE
IF REQUEST NOT TERMINATION (P1=9 OR -32768) THEN
ISSUE MESSAGE '*** XMNG INVALID REQUEST (II) FROM XENVN'
II=REQUEST & NW IS LU

SET UP FOR DDS TERNMINATION
EXIT TO :XMEM
ENDIF
ENDIF
ENDDO

:XMEM
IF DUMP OPTION THEN
CALL XUDMP TO DUMP OUR PARTITION
ENDIF
RELEASE DWA TRACK SPACE
FREE EXEC'S & PROCESSOR'S CLASS NUMBERS
CALL RNRR CLEAR LOCK ON STATUS TABLE
UNLOCK THE MANAGER FROM THE PARTITION
CALL SLIBR DISABLE
CLEAR EXEC'S ID SEGMENT
CALCULATE EQT ADDRESS FOR ATTENTION ID
CLEAR THE STATUS TABLE ENTRY
CALL SLIBR ENABLE
CALL RNRR CLEAR THE LOCK ON THE STATUS TABLE
ISSUE MESSAGE '***XMOZ SIGN OFF FOR LU "NW"'
CALL SLIBR GET PRIVLEDGED AND DISABLED
CALL SLIST TO MAKE US DORMANT
CLEAR OUR OWN ID
RESET EQT TO REMOVE ATTENTION ID
EXIT :SEXEC TO THE RTE DISPATCHER
END :XMEGR
BEGIN XMIK
CALL XMTN TO FIND &SETB TOC ENTRY
FIN Address IF &SETB
CALL XMST TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
CALL XMST 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)
CALL XMST TO SEARCH AWA FOR INTERFACE TABLE (CHAIN 4)
EXIT TO :ERROR: IF TABLE NOT FOUND (PARMS = 2)
IF TABLE NOT IN AWA, THEN
CALL INDR 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 SCHEDUL
EXIT TO :ERROR: IF NOT FOUND
CALL SLIBR TO BE PRIVILEGED
CALL SLIBX 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 XASF TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XMSTN TO FIND &SETB TOC ENTRY
FIND ADDRESS OF &SETB
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARM = 1)
CALL XHAG TO HONOR AWA REQUEST
ENDIF
CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
EXIT TO :ERROR: IF PROCESSOR REQUESTED TERMINATION (PARM = 1)
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 XMST 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 XMSTX TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAN TO RESCHEDULE PROCESSOR
EXIT TO :ERROR: TO TERMINATE SEQUENCE
ENDIF
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDDO
172 2 PERFORM END: SET PARM=0
173 2 :ERROR: SET PARM=-1 AND PAMS TO APPROPRIATE REASON CODE
174 2 CALL XMQT TO POST EXEC AND WAIT FOR NEXT REQUEST
175 2 CALL XMQT TO POST EXEC AND WAIT FOR NEXT REQUEST
176 2 CALL XMQT TO POST EXEC AND WAIT FOR NEXT REQUEST
177 2 SET CURRENT TASK IN MR-STATUS TABLE, AND AMA TO EXEC
178 2 CALL XMQT TO POST EXEC AND WAIT FOR NEXT REQUEST
179
180 1 END XMQT

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
CALLING PROCEDURE

182  1  *00  JSB XMAFR
183  1  *00  DEF **3
184  1  *00  DEF ADDR
185  1  *00  DEF SIZE

186  1  *00  ********
187  1  *01  PLACE A FE ON THE FE CHAINS AND MERGE WITH ANY ADJACENT FES
188  1  *00  ********
189  1  *02  INPUT
190  1  *02  ADDR  - ADDRESS OF AREA BEING FREED
191  1  *02  SIZE  - SIZE OF AREA BEING FREED. IF LESS THAN 3 THE FOLLOWING
192  1  *02  WORD(S) WILL ALSO BE FREED SUCH THAT THE MINIMUM FE SIZE
193  1  *02  OF 3 WORDS IS MAINTAINED.
194  1  *02  ********
195  1  *02  EXTERNAL SYMBOLS FROM XMAWA
196  1  *02  XMBCP, XMFCP, XMFNC, XMCRE
197  1  *02  ********
198  1  *03  OUTPUT (EXTERNAL SYMBOLS FROM XMAWA)
199  1  *03  XMBCP, XMFCP, XMFNC, XMCRE
200  1  *03  ********
201  1  *03  NOTES
202  1  *03  USES .ENTR
203  1  *03  ********
204  1  *05  ********
205  1  *05  ********
206  1  *05  ********
207  1  *05  ********
208  1  *05  ********
209  1  *05  ********
210  1  *05  ********
211  1  *05  ********
213 1 BEGIN XMAFR
214 2   SET NEW FE SIZE FIELD TO MAX(SIZE, 3)
215 3   INCREMENT TOTAL FREE SPACE BY NEW FE SIZE
216 4   INDEX TO FORWARD CHAIN POINTER (FCP) HEAD
217 5   START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)
218 6   EXIT IF 'ADDR' < FCP VALUE
219 7   SET CURRENT FCP VALUE TO 'ADDR'
220 8   SET NEXT FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE
221 9   PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE
222 10  OR ELSE
223 11   INDEX TO NEXT FE FCP
224 12  END LOOP
225 13  SET NEXT FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
226 14  SET CURRENT FE'S FCP VALUE TO 'ADDR'
227 15  SET NEW FE BCP TO BCP HEAD VALUE
228 16  SET BCP HEAD TO 'ADDR'
229 17  END SEARCH
230 18  IF NEW FE BCP NOT = END-OF-CHAIN
231 19  THEN
232 20   PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
233 21  END IF
234 22  IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW (OR CONSOLIDATED) FE, THEN
235 23  MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
236 24  ENDIF
237 25 1 END XMAFR
238 26 1 BEGIN MERGE
239 27 2   IF FE 1 IS ADJACENT TO FE 2
240 28 3   THEN
241 29 4   INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
242 30 5   SET FE 1 FCP TO VALUE OF FE 2 FCP
243 31 6   IF FE 1 FCP NOT = END-OF-CHAIN
244 32 7   THEN
245 33 8   SET FE 3 BCP TO ADDRESS OF FE 1
246 34 9   ELSE
247 35 10  SET BCP HEAD TO ADDRESS OF FE 1
248 36 11  ENDIF
249 37 2  ENDIF
250 3 1 END MERGE
CALLING PROCEDURE

JBB XNAGT  
DEF *+3  
DEF OPTM  
DEF SIZE

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

INPUT

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

EXTERNAL SYMBOLS FROM XMAYA
XMFCP, XMFCN, XMFG

OUTPUT
A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
INDICATING SOME AVAILABLE
EXTERNAL SYMBOLS FROM XMAYA
XMFCP, XMFCN, XMFG
XMFPK, XMFK

NOTES
IF SOME ARE EXACTLY "SIZE" WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
ERROR RETURN (A-REG = -32768) IS TAKEN
USES .EMIR
1 BEGIN XMAGT
2 IF TOTAL FREE SPACE < MAX(SIZE,3)
3 THEN
4 CALL XMAGT PURGE DNA ELEMENTS FROM AMA
5 SET PHASE$1$ FLAG
6 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
7 RETURN VALUE= -32768
8 EXIT TO :XMAGR
9 ENDIF
10 ENDIF
11 PERFORM XMERC(OPTY,SIZE)
12 IF RETURN CODE IS NOT FOUND, THEN
13 CALL XMAGT
14 IF RETURN CODE IS NOT FOUND, THEN
15 RETURN VALUE= -32768
16 EXIT TO :XMAGR
17 ENDIF
18 RETURN VALUE= AREA
19 :XMAGR
20 ENDIF
21 CLEAR PHASE1 FLAG
22 1 END XMAGT
BEGIN XMSRC

1. SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FHEAD(OPTH)
2. START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
3. WHILE TOC SPACE FENCE HAS NOT BEEN CROSSED
4. EXIT IF FS SIZE = MAX('SIZE', 3)
5. DECERNENT TOTAL FREE SPACE BY MAX('SIZE', 3)
6. DECHAIN FE
7. RETURN ADDRESS OF AREA
8. EXIT IF FS SIZE >= MAX('SIZE', 3) + 3
9. DECERNENT TOTAL FREE SPACE BY MAX('SIZE', 3)
10. IF ALLOCATING FROM HEAD OF SPACE (OPTH = 0)
11. THEN
12. CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
13. RECHAIN NEW FE
14. RETURN ADDRESS OF AREA
15. ELSE
16. CHANGE SIZE FIELD TO FS SIZE - MAX('SIZE', 3)
17. COMPUTE AND RETURN ADDRESS OF AREA
18. ENDF
19. OR ELSE
20. INDEX TO NEXT FE
21. END LOOP
22. SET RETURN CODE TO 0(NOT FOUND)
23. END SEARCH
24. IF TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
25. INCREMENT TOC SPACE FENCE ON 'SIZE'
26. ENDF
27. END XMSRC
CALLING PROCEDURE

JSM XMAPG
DEF CLNG

*********

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

01 REQUEST LIST (SEE TABLE 6.2 - III)

*********

02 CLNSO - CLASS I/O NUMBER CONTAINING REQUEST LIST

02 REQUEST LIST FIELD 8

*********

03 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)

03 REQUEST LIST FIELD 8

*********

05 NOTES

05 USES EXEC,XMAFR,XMACM,XMAPK,XMIFK

05 XMON,XMDAL,XMODA,XMODT,XMODT

05 NEITHER EXEC NOR THE PROCESSES WILL MAKE AWA MANAGEMENT REQUESTS

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

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

05 THE DWA AND COPY THEM BACK TO THE AWA AS NECESSARY.
1 BEGIN XNAMG
2 RETRIEVE AWS 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 (O), 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 AWS IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
11 ENDF
12 WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O
13 STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT
14 SET REQUESTED SIZE FROM TOC SIZE
15 SET NEW WRITE FLAG
16 ELSE
17 IF REQUEST TO CLEAR (CODE 17)
18 THEN
19 GET AC(BDMA) FROM XMDWA
20 SAVE HEADER AND DIRECTORIES-SIZE
21 CLEAR XMDWA THRU XMDWB
22 BUILD AN FE AT XMDWA FOR AWS SIZE
23 CALL XNAMG TO ALLOCATE A TOC ENTRY FOR BDWA
24 CALL XNAMG TO ALLOCATE SPACE FOR BDWA
25 SET LOCATION, SIZE, & KEY IN THE TOC
26 SET DIRECTORY ADDRESSES AT XMDWA
27 CLEAR THE DIRECTORIES
28 SET LU, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORIA
29 ELSE
30 CALL XMTKN TO SEARCH TOC FOR INDICATED ENTRY
31 CASE (:VERIFY; :VERALO; :VERALO; :RNAME; :DELIVER; :DELIVER; :STORE;)
32 (:RETRIVE; :RETRIVE;) REQUEST CODE
33 :VERIFY:
34 IF ENTRY NOT FOUND
35 THEN
36 SET RETURN PARM1 AND PARM2 (2 & INDEX)
37 ENDF
38 :VERALO:
39 IF ENTRY ALREADY EXISTS
40 THEN
41 IF ALLOCATE REQUEST (3)
42 THEN
43 SET RETURN PARM1 AND PARM2 (3 & INDEX)
44 ELSE
45 IF TYPE, SIZE AND 1-DIM FIELDS DO NO MATCH
46 THEN
47 SET RETURN PARM1 AND PARM2 (4 & INDEX)
48 ENDF
49 ELSE
50 CALL XNAMG TO ALLOCATE TOC SPACE
51 IF CLASS EQ 3 OR 8, THEN
52 CHAIN IN NEW TOC ENTRY
53 SET DATA SPACE ADDRESS TO ZERO
54 ELSE
55 CALL XNAMG TO ALLOCATE DATA SPACE
56 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 XMDAL DWA ALLOCATION
ENDIF
ENDIF

:RENAME:
IF ENTRY NOT FOUND
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ELSE
CALL XMTFN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION
ENDIF
ENDIF

:DELIVER:
IF ENTRY FOUND
THEN
GENERATE KEY 1 LESS THAN FOUND KEY
CALL XMTFN FOR GENERATED KEY
DECHAIN TOC ENTRY
CALL XMAFR TO RETURN TOC ENTRY SPACE TO FE POOL
IF CLASS EQ 4 OR 6, THEN
CALL XMDDA DWA DEALLOCATION
ENDIF
ELSE
IF DELETE REQUEST (5)
THEN
SET RETURN PARM1 AND PARM2 (2 & INDEX)
ENDIF
ENDIF

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

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 XMDRT MOVE INTO AMA
IF NO SPACE THEN
SET RETURN PARM1 AND PARM2 TO (1, INDEX)
EXIT TO XMRREX
ENDIF
ENDIF
WRITE VALUES TO CLASS I/O
STORE TYPE IN LOW BYTE OF REQUEST WORD SIX
STORE CLASS NUMBER IN REQUEST WORD EIGHT
SET REWRITE FLAG
ENDIF
ELSE
WRITE TOC ENTRY TO CLASS I/O
STORE CLASS NUMBER IN REQUEST WORD EIGHT
SET REWRITE FLAG
SIDERE
ENDIF
550  IF
551  ENDIF
552  END CASE
553  ENDIF
554  INCREEENT TO NEXT REQUEST CODE
555  ENDDO
556  STORE REWRITE FLAG IN RETURN PAMRS
557  IF REWRITE FLAG SET
558  THEN
559  WRITE REQUEST LIST BACK TO COMMUNICATIONS CLASS I/O NUMBER
560  ENDIF
561  END ASSOCIATED TASK WITH RETURN PAMRS
562  END
CALLINGPROCEDURE

**546** 1 **d0** 1 **d0** 1 JSB XMTFN

**547** 1 **d0** 1 **d0** 1 DEF **2**

**548** 1 **d0** 1 **d0** 1 DEF KEYS

**549** 1 **d0**

**570** 1 **d1**

**571** 1 **d1** EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'

**572** 1 **d1**

**573** 1 **d1**

**574** 1 **d1**

**575** 1 **d2** INPUT

**576** 1 **d2** KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC

**577** 1 **d2**

**578** 1 **d2** EXTERNAL SYMBOLS FROM XMA

**579** 1 **d2** XMD

**580** 1 **d2**

**581** 1 **d3** OUTPUT

**582** 1 **d3**

**583** 1 **d3** A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR

**584** 1 **d3** ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY

**585** 1 **d3** WHERE 'KEYS' COULD BE CHAINED IN

**586** 1 **d3**

**587** 1 **d3**

**588** 1 **d3** NOTES

**589** 1 **d5** USES .ENTR

**590** 1 **d5**

**591** 1 **d5**

**592** 1 **d5**

**593** 1 **d5**

**594** 1 **d5**

**595** 1 **d5**

**596** 1 BEGIN XMTFN

**597** 2 ISOLATE CLASS FROM KEY AND INDEX "O A-PRIORITE CHAIN HEAD

**598** 2 START SEARCH WHILE CHAIN POINTER .A) != END-OF-CHAIN (-32768)

**599** 3 COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY

**600** 3 EXIT IF MATCH

**601** 3 RETURN ADDRESS OF ENTRY

**602** 3 EXIT IF 'KEYS' < TOC ENTRY (EXPECTED ENTRY NOT IN CHAIN)

**603** 3 RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET

**604** 3 OR ELSE

**605** 3 INDEX TO NEXT TOC ENTRY ON CHAIN

**606** 0 END LOOP

**607** 0 RETURN ADDRESS OF LAST (PREVIOUS) ENTRY WITH INDIRECT BIT SET

**608** 1 END XMTFN

**609** 1 END SEARCH
611 1  #00 CALLING PROCEDURE
612 1  #00
613 1  #00 JSB XMSST
614 1  #00
615 1  #00
616 1  #00 ADJUST SEQUENCE LOCATION COUNTER
617 1  #00
618 1  #00
619 1  #00
620 1  #00 INPUT
621 1  #02 A-REG - INTEGER VALUE OF SEQUENCE NUMBER TO BE LOCATED
622 1  #02 EXTERNAL SYMBOLS
623 1  #02 SERAD - ADDRESS OF EXECUTING SEQUENCE TABLE IN AMA
624 1  #02 ENDB - DISPLACEMENT TO LAST ENTRY OF EXECUTING SEQUENCE TABLE
625 1  #02
626 1  #02
627 1  #02
628 1  #03 OUTPUT
629 1  #03 A-REG - DISPLACEMENT TO ENTRY IF SEQUENCE NUMBER IS FOUND, ELSE -32768
630 1  #03
631 1  #03
632 1  #03
633 1  #03
634 1  #03
635 1  #03
636 1  BEGIN XMSST
637 2  SAVE RESET SEQUENCE NUMBER IN LOCAL VARIABLE SEGNO
638 2  SET DISPLACEMENT IN A-REG TO ZERO
639 2  GET ADDRESS OF FIRST ENTRY IN SEQUENCE TABLE INTO X-REG
640 2  START SEARCH UNTIL CURRENT ENTRY DISPLACEMENT > LAST ENTRY DISPLACEMENT
641 2  EXIT IF INDICATED NUMBER FOUND
642 2  OR ELSE
643 3  INCREMENT DISPLACEMENT IN A-REG TO NEXT ENTRY
644 3  END LOOP
645 3  SET A-REG TO -32768 TO INDICATE NUMBER NOT FOUND
646 3  END SEARCH
647 1 END XMSST
649 1 BEGIN   XMPAW
650 2   #00 ENTRY: JSB XMPAW OR CALL XMRPWN
651 2   #02     DEF RETURN ADDRESS
652 2   #02     DEF RETURN ADDRESS
653 2   #02     INPUTS: XMRPN HAS REPLY
654 2   #02     OUTPUTS: XMDPN HAS REQUEST,
655 2   #02     AND XUVS IS UPDATED.
656 2   #02     DO UNTIL GOOD PARMS RECEIVED OR TOP AT TERMINATES
657 2   #02     IF ABORT CURRENT FLAG C(- STBLU) THEN
658 2   #02     SET ABORT FLAG IN XMPRE
659 2   #02     TURN OFF ABORT CURRENT FLAG
660 2   #02     ENDIF
661 2   #02     GET CURRENT TOP ASSOCIATED TASK(AT)
662 2   #02     SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
663 2   #02     JSB XVSPA
664 2   #02     DEF *+3 RETURN
665 2   #02     DFC 0 MANAGER CALL
666 2   #02     DEF XMPW PARM FIELD
667 2   #02     THIS IS AN IMPLIED WAIT
668 2   #02     :XSPC GET CURRENT XUVS ENTRY (XUSTA)
669 2   #02     IF THERE HAS BEEN A CALL TO PRNA (P1 FIELD IS NOT TOP AT) OR
670 2   #02     IF SEGMENT IS DORMANT OR
671 2   #02     CID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
672 2   #02     IF PARM IS NOT 0,7,8,9, OR -32768 THEN
673 2   #02     SET PARM TO -32768
674 2   #02     PRINT ERROR "INVALID REQUEST"
675 2   #02     ELSE
676 2   #02     SET PARM FIELD FROM MANAGER'S ID SEGMENT
677 2   #02     EVID
678 2   #02     ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PAM)
679 2   #02     IF PARM CONTAIN TO ABORT TOP AT
680 2   #02     CALL XUUI TO ABORT TOP AT
681 2   #02     PRINT ERROR "INVALID REQUEST"
682 2   #02     ELSE
683 2   #02     SET PARM FROM CURRENT ID SEGMENT
684 2   #02     EVID
685 2   #02     EVID
686 2   #02     EVID
687 2   #02     EVID
688 2   #02     EVID
689 2   #02     EVID
690 2   ENDDD
691 2   ENDD
CALLING PROCEDURE

JSB XMDIN

FUNCTION

INITIALIZE TOC

& DATA FUNCTIONS

OUTPUT

BREG O=COMPLETE

MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

XTFN, XMAGT

BEGIN XMDIN

GET NUMBER OF DATA TRACKS FROM P5

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

CALL XMTFN (EDWA)

CALL XMAGT (0,8) TOC ENTRY FOR EDWA

CALL XMAGT (1 SIZE) DATA AREA FOR EDWA

INITIALIZE & CHAIN EDWA TOC ENTRY

SET # OF TRACKS FOR EDWA

CLEAR EDWA

CALL EXEC (DISC TRACK ALLOCATION)

IF DISC ADDRESS .EQ. -1 TRACKS NOT AVAILABLE

THEN

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

ERROR RETURN

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DATA MANAGEMENT

ENDIF

END XMDIN
1 BEGIN XBAL
2 * DWA ALLOCATION
3 40 CALLING PROCEDURE
3 40 JSB XBAL
3 41 FUNCTION
3 41 ALLOCATE DWA SPACE
3 41 INPUTS
3 42 A(TOC ENTRY OF THE DWA ELEMENT)
3 42 IN TREG
3 42 OUTPUTS
3 43 UPDATE TOC ENTRY FOR "DWA"
3 43 RETURNS BREG= ZERO= ALLOCATION COMPLETE
3 43 = MINUS= ERROR CONDITION
3 43 NOTES
3 45 CALLS XMNS3, XMBST
745 2 IF (DWA) .NE. 0 THEN
749 3 SET START-ADDRESS TO FIRST TRACK WORD IN DWA DIRECTORY
750 3 GET SIZE(IN WORDS FROM TOC ENTRY)
751 3 SIZE(IN SECTORS)= (SIZE+63/64)
752 3 DO FOR NUMBER OF TRACKS IN DWA OR DWA SIZE .GE. SIZE
753 4 SET BITNUM TO ZERO
754 4 DO DOWHILE DWA SIZE .GE. SITE OR BITNUM .GE. 96
755 5 CALL XR:XB (0,BITNUM,START-ADDRESS)
756 5 STARTBIT=BITNUM
757 5 CALL XR:x (1,BITNUM,START-ADDRESS)
758 5 DWA-SIZE=BITNUM-STARTBIT
760 4 IF BITNUM .GE. 96, THEN
761 4 START-ADDRESS=START ADDRESS+1 TRACK ADDRESS
763 4 ENDF
764 3 ENDDO
765 3 IF 0;"C AREA FOUND, THEN
766 4 START-ADDR="F5" & BITNUM DEFINES THE TRACK & SECTOR
767 4 SET DISC ADDRESS & SIZE IN TOC ENTRY
768 4 CALL XPS:TOC-ENTRY)
769 4 SET NORMAL RETURN
770 3 ELSE
771 4 ENDIF
772 4 ISSUE MESSAGE ***XNOS NO DWA SPACE REMAINING"
773 4 SET ERROR RETURN
774 3 ENDF
775 2 ENDIF
1 BEGIN XMDQA
2 = XWALLOCATION
3 CALLING PROCEDURE
4 JSB XMDXA
5 'FUNCTION
6 DELETE DNA ELEMENT WHICH CORRESPONDS
7 TO THE AMA ELEMENT
8
9 INPUTS
10 TOC ENTRY ADDRESS OF AMA ELEMENT
11 IN TREG
12
13 NOTES
14 USES XMBST
15
16 IF A(SDWA) NE 0 THEN
17 GET DISC ADDRESS
18 GET DATA SIZE
19 * FREE THE DISK AREA
20 CALL XMBST (TOC ENTRY)
21 2 EMDIF
22
23 1 END XMDQA
BEGIN XMST
CALLING PROCEDURE

FUNCTION
SET ALLOCATION & DEALLOCATION IN DMA DIRECTARY

INPUTS
DMA TOC ENTRY

OUTPUTS
UPDATES DMA DIRECTORY

GET TRACK-ADDRESS FROM TOC
SUBTRACT START OF TRACKS FROM DMA DIRECTARY FOR RELATIVE TRACK
DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)
QUOTIENT IS NUMBER OF RELATIVE WORDS
REMAINDER IS BIT-POSITION(BP)
WORD ADDRESS=DMA ADDRESS+RELATIVE TRACK+RELATIVE WORDS
SUBTRACT BIT-POSITION FROM 16 GIVING NBFW(NUMBER BITS IN FIRST WORD)
GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)
BIT-NBFW=REMAINING BITS(RE)
IF RB, I.E., 0, THEN
SET NUMBER OF WORDS(NW) TO ZERO
SET NUMBER OF BITS LAST WORD(NBLW) TO ZERO
SET NBFW TO BITS
ELSE
DIVIDE RB BY 16
SET NW TO QUOTIENT
SET NBLW TO REMAINDER
ENDIF
L-WORD DATA POINTED TO BY WORD ADDRESS
ROTATE LEFT (BP+NBFW-1 BITS) SAVE BP BITS & POSITION NBFW BITS
EXCLUSIVE OR SIGN BIT; CON TO OFF; OFF TO ON
IF NBFW .GT. 1, THEN
SHIFT RIGHT(ARITHMETIC) BY NBFW-1 PROPAGATE BITS
ENDIF
IF DATA IS NEGATIVE, THEN
SET FILL WORD TO -1
ELSE
SET FILL WORD TO 0
ENDIF
ROTATE RIGHT BY BP; RESET SAVED BITS
STORE WHERE WORD-ADDRESS POINTS
DO WHILE NW .GT. 0
WORD-ADDRESS=WORD-ADDRESS+$
SET FILL WORD INTO WHERE WORD-ADDRESS POINTS
ENDIF
IF NBFW .GT. 0, THEN
WORD-ADDRESS=WORD-ADDRESS+$
LOAD DATA POINTED TO BY WORD ADDRESS
IF NBFW .GT. 1, THEN
ROTATE LEFT NBFW-1 BITS
ENDIF
EXCLUSIVE OR SIGN BIT
IF NBFW .GT. 1, THEN
SHIFT RIGHT ARITHMETIC BY NBFW-1
ENDIF
STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00
3 JSB XMAPK
4 DEF OPTION
5 DEF PHASE 1 FLAG
6 GENERAL COLLAPSE INTERFACE
7 OUTPUT
8 RETURNS AREA ADDRESS FOR AREA FOUND
9 RETURNS -32768 AREA NOT FOUND
10 NOTES
11 USES XMPK1,XMPK2,XMPK3,XMSRC
12 IF PHASE1 FLAG NOT SET, THEN
13 CALL XMPK1 PURGE DWA ELEMENTS FROM ANA
14 PERFORM XMSRC(OPTN,SIZE)
15 ELSE
16 SET RETURN CODE TO NOT FOUND
17 ENDIF
18 IF RETURN CODE IS NOT FOUND, THEN
19 IF OPTN=1(BACKWARD CHAIN), THEN
20 CALL XMPK2 PACK TOC CHAIN
21 CALL XMSRC(OPTN,SIZE)
22 ENDIF
23 IF RETURN CODE IS NOT FOUND, THEN
24 CALL XMPK3 PACK ANA DATA AREAS
25 PERFORM XMSRC(OPTN,SIZE)
26 ENDIF
27 ENDIF
28 SET RETURN VALUE TO RETURN CODE
29 END XMAPK
960 1 BEGIN XMPK1
961 2 *DO CALLING PROCEDURE
962 2 *DO JSB XMPK1
963 2 *DO
964 2 *DO PHASE 1 OF COLLAPSE(PURGE DNA ELEMENTS)
965 2 *DO OUTPUT
966 2 *DO UPDATES ADDRESS FIELD IN TOC
967 2 *DO FOR EACH DNA DATA ELEMENT
968 2 *DO
969 2 *DO NOTES
970 2 *DO USES XMAFR, XMND6, XHND6, XMMP4
971 2 *DO CURRENTLY ONLY CLASS 4 & 6 ELEMENTS
972 2 *DO (INTERFACE TABLES & SEQUENCE TABLES)
973 2 *DO ARE DNA ELEMENTS.
974 2 *DO
975 2 *DO INCREMENT XMMP4 UPDATE PHASE 1 COUNT
976 2 *DO IF DMA DIRECTORY ADDRESS NOT EQ 0, THEN
977 2 *DO SAVE X & Y REGS
978 3 *DO FOR ALL CLASS 4 ELEMENTS
979 4 *DO CALL XMAFR(LATA, SIZE)
980 5 ENDDO
981 3 *DO FOR ALL CLASS 4 ELEMENTS
982 4 *DO CALL XMAFR(DATA, SIZE)
983 5 ENDDO
984 3 *DO RESTORE X & Y REGS
985 4 ENDF
986 2 ENDP
987 1 ENDP XMPK1
BEGIN XMPK2
  CALLING PROCEDURE
  JSB XMPK2

  PHASE 2 OF COLLAPSE(COLLLAPSE TOC ENTRIES)

  OUTPUT
  REORDERS TOC ENTRIES
  UPDATES XMFNC(TOC FENCE ADDRESS)

  NOTES
  USES XMFNC,XMFCP,XMTFM,XMBCP,XMAFR,XMP2

  INCREMENT XMP2
  UPDATE PHASE 2 COUNT

  1 DO WHILE XMFNC(.GT. XMFCP) AND XMFCP(.NE. -32768)
  2   ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
  3   WILL A TOC COMPRESS BE DONE.
  4
  5 GET FIRST-FREE(FRC=1) FROM XMFCP
  6 GET JLDSZ FROM THE FREE ELEMENT
  7 NWSIZ=OLDSZ-8
  8 IF NWSIZ .GT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
  9
  10 GET NEXT-FREE FROM FIRST-FREE'S FCP
  11 SET INTO XMFCP
  12 DELETE FROM THE FORWARD CHAIN
  13 IF XMFCP(.NE. -32768), THE: IF NOT THE LAST FREE ELEMENT
  14 SET NEXT-FREE'S BCP TO -32768
  15 ELSE
  16 SET XMFCP TO -32768 DELETE FROM BACKWARD CHAIN:
  17 ENDIF
  18 ELSE
  19 OLDsz=NWSIZ
  20 UPDATE LENGTH IN FREE ELEMENT
  21 ENDIF
  22 NWTFC=(FIRSTP+1)/NWSIZ
  23 THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
  24 OLTFC=XMFNC-8
  25 OLD TOC IS THE ENTRY MOVE THE FENCE
  26 TMPKEY=OLTFC'S KEY -1 CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
  27 CALL XMFNM(TMPKEY) FIND PREVIOUS
  28 SET PRTOC FROM ARGC
  29 COPY TOC ENTRY FROM OLTFC TO NWTFC
  30 STORE NWTFC ADDRESS INTO PRTOC'S CHAIN
  31 CALL XMAFR(OLTFC,TOC-SIZE) FREE THE OLD TOC ENTRY
  32 ENDIF

END XMPK2
1033 1 BEGIN XMPK3
1034 2 *00  CALLING PROCEDURE
1035 2 *00     JSB XMPK3
1036 2 *00
1037 2 *01     FUNCTION
1038 2 *01     COMPRESS AMM DATA AREA BY MIGRATING DATA AREAS
1039 2 *01 TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
1040 2 *01
1041 2 *03     OUTPUT
1042 2 *03     UPDATES XMBCP,XMFNC,XMFRE,XMPK3
1043 2 *05
1044 2 *05     NOTES
1045 2 *05     USES XMAFP
1046 2 *05
1047 2 *01     INCREMENT XMMP3 (NUMBER OF PHASE 3)
1048 3 *01     IF XMPK3 .NE. -32768, THEN THERE ARE FREE AREAS
1049 4 *01     CALL XMPK3, INSURE TOC IS COMPRESSED
1050 3 *01     DO WHILE XMBCP .GT. XMFNC UNTIL 1 FREE AREA ADJACENT TO THE FENCE
1051 4 *01     IF XCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
1052 5 *01     SET HIGH-WATER TO XMFCN
1053 4 *01 ELSE
1054 5 *01     SET HIGH-WATER TO BCP OF LAST-FREE
1055 4 *01     ENDIF
1056 4 *01     SET LOW-WATER TO A(LAST-FREE)
1057 4 *01     PERFORM XMB:CT(LOW-WATER,HIGH-WATER)
1058 4 *01     EXIT IF TOC-ADDRESS .EQ. 0
1059 4 *01     PERFORM XMVC(TOC-ADDRESS)
1060 4 *01     ENDDO
1061 2 ENDIF
1062 1 END XMPK3
1064 1 BEGIN XMTSC  TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR 0(NONE FOUND).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XMWA, THE.
1070 2 * LAST TOC-ENTRY IS AT XMFC-B.
1071 2 TOC-ENTRY=XMWA
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 DO UNTIL TOC-ENTRY .GE. XMFC
1074 3 IF DATA ADDRESS IN TOC-ENTRY IS
1075 4 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 ENDIF
1083 3 ADD 8 TO TOC-ENT
1084 2 ENDOO
1085 2 TOC-ADDRESS=TEST-TOC  RETURN 0 OR A TOC ADDRESS
1086 1 END XMTSC
1088    1 BEGIN XMANY
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 XMFRM(DATA ADDRESS,SIZE)
1096    2 GET FREE AREA FROM XMBCP
1097    2 DECREASE FREE AREA LENGTH BY MAX(DATA SIZE,3)
1098    2 CALCULATE NEW ADDRESS FROM FREE AREA + FREE LENGTH
1099    2 MOVE DATA FROM DATA ADDRESS TO NEW ADDRESS
1100    2 MOVE SAVED FIRST THREE WORDS TO NEW ADDRESS
1101    2 UPDATE DATA ADDRESS IN TOC WITH NEW ADDRESS
1102    1 ENQ XMANY
FORTRAN CALLING PROCEDURE

CALL XPAR (LU, LINTBUF, LINTLG, MBUFF, IMIN, NAME, TYPE, SIZE, IDIN, D8PT)

XPAR allows processors to obtain the attributes of the parameters referenced by the interface table.

--

INPUT

HU - LOGICAL UNIT NUMBER OF USER TERMINAL
INTBUF - 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 INTBUF
MBUFF - MANAGER REQUEST BUFFER (8 WORDS) USED TO COMMUNI-
WHEN THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACCESS PROCESSOR SERVICE CALLS.
IMIN - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE WHOSE
ATTRIBUTES ARE REQUESTED.

OUTPUT

NAME - AN ALPHANUMERIC NAME OF UP TO SIX CHARACTERS WHICH
IDENTIFIES THE DATA ELEMENT OR DDE WHERE THE DATA IS TO
BE OBTAINED/STORED. A ZERO ENTRY INDICATES INPUT DATA
WHICH IS LITERAL DATA STORED WITHIN THE INTERFACE TABLE.
TYPE - DATA TYPE CODE OF THE PARAMETER
SIZE - TOTAL NUMBER OF WORDS OF LITERAL DATA,
TOTAL NUMBER OF WORDS OF REFERENCED INPUT DE,
TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DE IF
SUBSCRIPTED OR ZERO IF NOT SUBSCRIPTED.
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DDE OR
ZERO IF OUTPUT DDE.
IDIM - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DDE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIPTED OUTPUT.
DSPT - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIPTED DATA ELEMENTS ELSE ZERO.
FOR AN INPUT DDE THE RTE FILE MANAGER TYPE CODE IS
RETURNED IN D8PT.

EXTERNAL SYMBOLS

(SEE XPAR)

INTERNAL VARIABLES

NOTES

USES ENJN, XPAR1(XPAR), XPAR2(XPAR), XPAR3(XPAR), XPAR4,
AP68(XPAR), XPAR5, XPAR6

XPAR3 IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
POINTS XPAR, XPAR AND XPAR
BEGIN XPATHR
PERFORM XPINI(XPGET) TO INITIALIZGIBLS AND INTERFACE TABLE
EXIT TO :XPG13 (XPGET) IF PARAMETER IS OUT OF RANGE
EXTRACT NAME FROM INTERFACE TABLE
SET DSPTT TO ZERO
IF LITERAL PARAMETER (NAME IS ZERO)
THEN
COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
ELSE
IF SUBSCRIPTED (INTERFACE TABLE DISP OR $ FIELDS ARE NON-ZERO)
THEN
PERFORM XPIBC(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
STORE TYPE, SIZE, IDIM AND DSPTT
ELSE
IF DBDE
THEN
PERFORM XPQFM(XPGET) TO QUALIFY FILE NAME
ENDIF
IF INPUT PARAMETER
THEN
CALL XPREQ TO RETRIEVE TOC ENTRY
COPY TYPE, SIZE, IDIM AND DSPTT FROM TOC ENTRY
ELSE
SET TYPE, SIZE AND IDIM TO ZERO
ENDIF
ENDIF
ENDIF
END XPATHR
FORTRAN CALLING PROCEDURE

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

********
XPGET ALLOWS PROCESSORS TO OBTAIN DATA FROM THE BDS AWA. IT
VERIFIES THE EXISTENCE OF THE DATA ELEMENTS AND ORDE'S
REFERENCED BY THE 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 A 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 BORE NAMES ARE SEARCHED FOR IN THE AWA TOC.
IF A "?" FIND A, A MESSAGE IS ISSUED AND PROCESSING TERMINATED.

********
INPUT
LU   - LOGICAL UNIT NUMBER OF USER TERMINAL
INBUF - INPUT/OUTPUT BUFFER OF 7 (8 PARAMETERS + 1) WORDS,
INTLNG - LENGTH OF INBUF
MRUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCAN
AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
N   - NUMBER OF PARAMETERS TO BE RETRIEVED. IF NON, 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 SUMCM IN DETERMINING WORD DISPLACEMENTS FOR
VARIOUS TYPES OF SUBSCRIBED ARRAYS
CLSTL - ADDRESS OF THE CLASS "FILE" (SECOND WORD) OF THE REQUEST
ARRAY RST
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
COUNTER OF THE NUMBER OF PASSES THROUGH THE PARAMETER
155 104 PROCESSING LOOP
156 104 IM = VALUE OF -P, WHERE P IS THE NUMBER OF PARAMETERS IN THE
157 104 INTERFACE TABLE
158 104 NAMEFL = ADDRESS OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
159 104 ARRT[IM] = ARRAY REQUEST
160 104 NARY = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
161 104 SEQUENCE
162 104 REEST = EIGHT WORD ARRAY USED FOR CONSTRUCTING AND MANAGING
163 104 REQUESTS FOR XPSET
164 104 USRID = USER FILE IDENTIFIER CHARACTER (*IPARM(3))
165 104 EXTERNAL VARIABLES (SEE XPSET)
166 104 XPCLS
167 104 XPLU
168 104 XPQD
169 104 **
170 104 ** NOTES
171 104 **
172 105 USES .ENTRY, EXEC, XPSET, XPSIT, XVSTO
173 105 **
174 105 XPSET MUST BE_INCLUDED IN PROCESSOR AT FDS BUILD TIME.
175 105 **
176 105 SPME INSTRUCT MUST BE USED BY XPSET, XPSIT, AND XPATR AND XEGOS
177 105 TO BE_INITIALIZED ONLY ONCE BY ANY OF THE THREE ROUTINES.
178 105 **
179 105 SINCE REQUESTS FOR INPUT DATA FROM THE AWA MAY BE PASSED FOR UP TO
180 105 **
181 105 EIGHT AT A TIME, XPSET RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
182 105 **
183 105 REQUESTED IN MULTIPLES OF EIGHT.
184 **
185 **
FORTRAN CALLING PROCEDURE

CALL XPUT(Call, INTRU, INTRL, NRBUF, N, IMUX, OUT(1), ..., OPENIN)

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

XPUT allows processors to store data into data elements and
out data is referenced in the interface table as follows:
1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC.
2) IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING
TERMINATES.
3) IF NOT FOUND, IT IS ADDED TO THE TOC. IF THE NAME IS
found or after being added to the TOC, the name is
prevented with a / symbol, suffixed with a one.
4) CHARACTER USER CODE AND RETURNED SO THE CALLING
PROGRAM CAN STORE DATA.

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

INPUT - LOGICAL UNIT NUMBER OF USER TERMINAL
INTRU - INPUT/OUTPUT BUFFER OF # PARAMETERS + 12 WORDS,
INTRU - INTERFACE TABLE HEADER, FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTRU - LENGTH OF INTRU
NRBUF - 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.
IMUX - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
INTERFACE TABLE REPEATING DATA ELEMENTS AND DIB'S.
OUT(N) - INTERFACIE ARE YOU BEING STORED. THE ORDER OF THE
OUT(N) - LABELS OR VARIABLE NAMES WHERE OUTPUT DATA IS TO BE
STORED FROM.

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

XPGFF - FLAG INDICATING XPGF/XPGFT NORMAL PROCESSING (-1) OR
XPGF/XPGFT SPECIAL PROCESSING BY-PASSING SUBSCRIPT
RESOLUTION (0=10)

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

OUTPUT
NOME

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

INTERNAL VARIABLES

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

MOVES

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

(SEE XPGF)
BEGIN XPGET
SET FOR 'GET'
PERFORM ACCESS TO RETRIEVE DATA
END XPGET
BEGIN XPPUT
SET FOR 'PUT'
PERFORM ACCESS TO STORE DATA
END XPPUT
BEGIN XPSET
PERFORM XPINIT TO INITIALIZE GLOBALS AND INTERFACE TABLE
DO FOR EACH PARAMETER REQUESTED
IF SELECTED PARAMETER IS OUT OF RANGE
CALL XPREF TO PURGE QUEUED REQUESTS
EXIT TO :XPET1;
ENDIF
IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/'PUT' PROCESSING
THEN
CALL XPREF TO PURGE QUEUED REQUESTS
EXIT TO :XPET2;
ENDIF
IF OVERRUNNING CALLING SEQUENCE
THEN
CALL XPREF TO PURGE QUEUED REQUESTS
EXIT TO :XPET3;
ENDIF
IF PROCESSING FOR 'GET'
THEN
IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
THEN
IF INPUT IN LITERAL FORM
THEN
BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT
ELSE
IF NORMAL XPGET/PUT PROCESSING (XPGPF = -1)
THEN
IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
THEN
PERFORM XPBSC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
ENDIF
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST WITH DE NAME AND DISPLACEMENT
ENDIF
CALL XPREF TO QUEUE RETRIEVAL AND STORAGE OF INPUT
ELSE PARAMETER IS DODE (CLASS 3)
BUILD REQUEST FOR TDC ENTRY
CALL XPREF TO IMMEDIATELY RETRIEVE TDC ENTRY
PERFORM XPXRM TO CONSTRUCT AND STORE QUALIFIED FILE NAME
STORE FILE ATTRIBUTES
ENDIF
ELSE PROCESSING FOR 'PUT'
IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)
THEN
IF NORMAL XPGET/PUT PROCESSING (XPGPF = -1)
THEN
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
IF SUBSCRIBED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XREO TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRDE
CALL XREO TO QUEUE ALLOCATION
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AHA
CALL XREO TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS ORDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XREO TO QUEUE DELETION
BUILD RF ST TO REALLOCATE ORDE WITH NEW ATTRIBUTES
EXIT TO XERR: IF FILE TYPE NOT 1-13, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
CALL XREO TO QUEUE REALLOCATION OF ORDE
ENDIF
ENDDO
CALL XREQ TO COMPLETE QUEUED REQUESTS
EXIT ACCESS
:JERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:XERR13: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:JERR14: CALL XREO TO PURGE QUEUED REQUESTS
:JERR15: TERMINATE PROCESSOR FOR INVALID ORDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
333 BEGIN XPINI
334 INITIALIZE GLOBAL VALUES FROM LU AND XVS TB
335 TERMINATE PROCESSOR WITH XP10 ERROR IF LU NOT IN XVS TB
336 IF INTERFACE TABLE BUFFER NOT INITIALIZED
337 THEN
338 RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER
339 IF RETRIEVAL NOT SUCCESSFUL
340 THEN
341 TERMINATE PROCESSOR WITH *XP10 PROCESSOR INITIALIZATION ERROR*
342 ENDIF
343 ENDIF
344 EXIT TO XPE13: IF N < 0
345 END XPINI
346
347
348
349
350 BEGIN XPQFN
351 SET / IN FIRST CHARACTER POSITION
352 MOVE FOUR WORD NAME INTO MIDDLE POSITIONS
353 LOCATE FIRST BLANK CHARACTER
354 REPLACE BLANK WITH USER ID
355 END XPQFN
356
357
358
359
360 BEGIN XPBC
361 IF DOUBLY SUBSCRIPTED
362 THEN
363 BUILD REQUEST FOR SUBSCRIPTS STORED IN LITERAL BLOCK
364 CALL XPRN TO QUEUE RETURN OF SUBSCRIPTS
365 ELSE
366 SINGLE SUBSCRIPT IS CONTAINED IN DISPLACEMENT FIELD & J-SUBSCRIPT IS = 1
367 ENDIF
368 BUILD REQUEST TO RETURN TOC ENTRY
369 CALL XPRN TO IMMEDIATELY RETRIEVE TOC ENTRY
370 IF DOUBLY DIMENSIONED (IDIM > 0)
371 THEN
372 DISPLACEMENT = (IDIM*(JSUB-1) + JSUB-1) * # WORDS PER ELEMENT
373 ELSE
374 DISPLACEMENT = SIZE*(JSUB-1) + (JSUB-1) * # WORDS PER ELEMENT
375 ENDIF
376 END XPBC
377 1 END ACCESS
FORTRAN CALLING PROCEDURE

CALL XPGT1 (LU, INBUF, INTLNG, MBUF, INUM, IN, SIZE, DISP)

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

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

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

INPUT

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

** INBUF - INPUT/OUTPUT BUFFER OF 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 **

** MBUF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE **

** WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY **

** THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS **

** INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM **

** WHICH TO RETURN DATA **

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

** DISP - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER **

** AT WHICH TO BEGIN DATA RETRIEVAL. A VALUE OF ZERO **

** INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER **

** IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT **

** BE SPECIFIED, I.E., RETRIEVAL FROM AN AREA PREVIOUS TO **

** THE USERS SUBSCRIPTS IS NOT SUPPORTED **

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

OUTPUT

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

** BE STORED **

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

INTERNAL

** BDSP - DISPLACEMENT FROM THE REAL ORIGIN OF THE DATA ELEMENT AS **

** A RESULT OF USER SUBSCRIPTING. DISP IS ADDED TO THIS **

** VALUE IN THE REQUEST TO THE MANAGER **

** JPT - DISPLACEMENT INTO JEPST DEPENDING ON WHETHER GET (D) OR **

** PUT (I) PROCESSING IS INVOLVED **

** JEPST - TWO WORD VECTOR CONTAINING 'JSB XPGT' OR 'JSB XPGT' **

** INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGT AND **

** XPGT CALLS **

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

NOTES

** USES EXEC, .ENTR, XPATR, XPE13(XPATR), XPGT(XPATR), .XPGT(XPATR) **

** USES .ENTR, EXEC, XPATR, XPE13(XPATR), XPGT(XPATR), XPGT(XPATR), **

** XPGT(XPATR) **

*************
FORTAN CALLING PROCEDURE

CALL XPPTI (LU, INBUF, INTLNG, MRBUFF, INUM, OUT, SIZE, DISP)

**INPUT**

- **LU** - LOGICAL UNIT NUMBER OF USER TERMINAL
- **INBUF** - INPUT/OUTPUT BUFFER OF 7* (# PARAMETERS + 1) WORDS,
- **INTLNG** - INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
- **MRBUFF** - LENGTH OF INTBUF
- **INUM** - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE INTO
- **OUT** - ARRAY OF AT LEAST SIZE WORDS FROM WHICH OUTPUT DATA IS TO
- **SIZE** - TOTAL NUMBER OF WORDS TO BE STORED
- **DISP** - DISPLACEMENT FROM THE BEGINNING OF THE OUTPUT PARAMETER
- **NONE** - AT WHICH TO BEGIN DATA STORAGE. A VALUE OF ZERO
- **INTERNAL** - INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER
- **SEE XPPTI** - IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT
- **XPGTI** - BE SPECIFIED, I.E., STORAGE INTO AN AREA PREVIOUS TO THE
- **NOTES** - USERS SUBSCRIBERS IS NOT SUPPORTED.
- **XPPTI IS AN ENTRY POINT INTO XPGTI**
480 1 BEGIN XPGTI
481 2 SET FOR GET PROCESSING
482 2 PERFORM XPGXI TO RETRIEVE DATA
483 1 END XPGTI
484 1 BEGIN XPPTI
485 2 SET FOR PUT PROCESSING
486 2 PERFORM XPGXI TO STORE DATA
487 1 END XPPTI
488 1 BEGIN XPIXI
489 2 SET XPSF(XPST) TO BY-PASS NORMAL XPGET/XPPUT SUBSCRIPT PROCESSING
490 2 CALL XPATR TO ASSURE INITIALIZATION OF INTBUF AND RETURN BASE DISPLACEMENT
491 1 EXIT TO :XPE13: (XPATR) IF INDICATED PARAMETER IS A DACE FILE
492 2 SAVE INTERFACE TABLE ENTRY CLASS/TYPE WORD, FLAG/DISP WORD AND SIZE WORD
493 2 IF LITERAL (NAME = 0)
494 2 THEN
495 2 EXIT TO :ERROR: IF DISP + SIZE > SAVED SIZE
496 2 ELSE
497 3 INCREMENT BASE DISPLACEMENT TO CONVERT SUBSCRIPT
498 3 SET TYPE FIELD TO FREE
500 2 EK00F
501 2 CLEAR ENTRY SUBSCRIPT BIT
502 2 SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
503 2 SET SIZE FIELD TO SIZE
504 2 CALL XPGET/XPPUT TO TRANSFER DATA
505 2 RESTORE ORIGINAL INTERFACE TABLE ENTRY
506 2 RESTORE XFGET(XPGET) TO NOMINAL VALUE
507 1 EXIT XPIXI
508 2 :ERROR: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
509 1 END XPIXI
CALLING PROCEDURE

JSB XPREQ

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

DEF OTHN REQUIRED OPTION

DEF RST REQUIRED REQUEST

DEF ADRES OPTIONAL ADDRESS (SEE OTHN)

*********

XPGET/XPPUT BUFFERED AWA MANAGEMENT SERVICE

O2 IM-U T

O2 OTHN - OPTION WORD

SIGN BIT - 0 QUEUE REQUEST AND RETURN

O2 1 QUEUE REQUEST AND CLOSE BUFFER

O2 OTHN-0 QUEUE REQUEST AND TRANSFER DATA TO/FRON

O2 ADRES 1 QUEUE REQUEST ONLY

O2 REQT - EIGHT WORD REQUEST (SEE SDD 6.2.6.2)

O2 ADRES STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT

O2 VALUES CITED ONLY WHEN OTHN(RIGHT BIT) = 0

O2 EXTERNAL SYMBOLS

O2 XPLU TERMINAL LOGICAL UNIT NUMBER

O2 XPRED ADDRESS OF 64 WORD MANAGER REQUEST BUFFER

O2 XPLCS CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

*********

O3 OUTPUT

O3 IF A REQUEST FAILS, A MESSAGE IS ISSUED AND PROCESSING IS

O3 TERMINATED, OTHERWISE INCOMING DATA WILL BE STORED IN THE

O3 ADDRESSES SUPPLIED.

O3 LOCAL VARIABLES

O4 ADDS ADDRESS OF STORAGE AREA ADDRESS TABLE (ADS)

O4 ADDS TABLE OF ADDRESSES FOR STORAGE OF FETCHED DATA

O4 APRMS ADDRESS OF PARMS ARRAY

O4 CLASS CLASS NUMBER TO TRANSMIT DATA TO MANAGER

O4 PARMS PARAMETER RETURN AND SCRATCH AREA

O4 PTR SAVE AREA FOR ADDS AND XPRED POINTER

*********

O5 NOTES

O5 USES ENTR, EXE, XPST, XPUN

O5 IF DATA IS TO BE TRANSFERRED TO/FRON 'ADRES', THEN 'REQT' SIZE

O5 FIELD (WORD 6) MUST BE SET.

*********
1 BEGIN XPREQ
2  MOVE REQUEST INTO BUFFER
3  IF OPTION IS TO TRANSFER DATA
4  THEN
5    IF REQUEST IS TO RETRIEVE DATA (B)
6      THEN
7      STORE ADDRESS IN TABLE
8      ELSE SHOULD BE A REQUEST TO STORE DATA (7)
9      OUTPUT DATA TO CLASS I/O
10     STORE CLASS NUMBER IN REQUEST WORD 8
11    ENDIF
12  ENDIF
13  INCREMENT POINTER
14  IF BUFFER FULL OR OPTION IS TO CLOSE NON-EMPTY BUFFER
15  THEN
16  CLOSE BUFFER
17  TRANSMIT BUFFER TO MANAGER
18  PAM 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    ENDIF
40  ENDIF
41  ENDDO
42  EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
43  ENDIF
44  ENDIF
45  END XPREQ
608 1 CD************
609 1 CD0
610 1 CD0 FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE
611 1 CD0 CALL XPDRS (LU, PRMLEN, PROMPT, TYPE, DATLEN, IDIM, DATA, RETCS)
612 1 CD0 ************
613 1 CD0 CALLS XPROM, DECODES COMMUNICATIONS BUFFER LOOKING FOR A SPECIFIC
614 1 CD0 "TYPE" AND STORES IT IN "DATA" FOR "DATLEN" LOGICAL ELEMENTS
615 1 CD0 ************
616 1 CD0 INPUTS FROM CALLING SEQUENCE:
617 1 CD0
618 1 CD0 LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
619 1 CD0 PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
620 1 CD0 STRING USED FOR THE USER PROMPT
621 1 CD0 PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
622 1 CD0 USED AS THE USER PROMPT
623 1 CD0 TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE
624 1 CD0 EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
625 1 CD0 0 - FREE
626 1 CD0 1 - INTEGER
627 1 CD0 2 - REAL
628 1 CD0 3 - DOUBLE PRECISION
629 1 CD0 4 - CHARACTER STRING LENGTH 2
630 1 CD0 5 - CHARACTER STRING LENGTH 6
631 1 CD0 6 - CHARACTER STRING LENGTH 12
632 1 CD0 7 - CHARACTER STRING LENGTH 18
633 1 CD0 8 - CHARACTER STRING LENGTH 24
634 1 CD0 9 - CHARACTER STRING LENGTH 30
635 1 CD0 10 - CHARACTER STRING LENGTH 36
636 1 CD0 11 - CHARACTER STRING LENGTH 42
637 1 CD0 12 - CHARACTER STRING LENGTH 48
638 1 CD0 13 - CHARACTER STRING LENGTH 54
639 1 CD0 DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN
640 1 CD0 THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE
641 1 CD0 NUMBER OF WORDS.
642 1 CD0 IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA
643 1 CD0 AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0
644 1 CD0 IF IT IS A VECTOR.
645 1 CD0 ************
646 1 CD0 OUTPUTS FROM CALLING SEQUENCE:
647 1 CD0
648 1 CD0 DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE)
649 1 CD0 AREA TO CONTAIN THE USER'S RESPONSE
650 1 CD0 RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO
651 1 CD0 CALLER:
652 1 CD0 0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
653 1 CD0 1 - USER ENTERED 1 CONTENTS OF BUFFER UNPREDICTABLE.
654 1 CD0 2 - USER ENTERED A CR, THERE IS NO RESPONSE.
655 1 CD0 3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
656 1 CD0 34 CHARACTERS OR 17 WORDS.
657 1 CD0 4 - PARAMETER LIST IS INVALID. EITHER:
658 1 CD0 A. IDIM LESS THAN ZERO
659 1 CD0 B. DATLEN LESS THAN ONE
660 1 CD0 C. TYPE VALID TYPE SPECIFIED.
661 1 CD0 ************
662 1 CD0 INTERNAL VARIABLES
DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT ELEMENT IS TO BE STORED
EFFTP - THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER STRING RESPONDER, IF TYPE IS FREE, IT IS THE CLOSEST SUPPORTED LENGTH, ELSE IT IS THE LENGTH SPECIFIED BY TYPE.
IND - CONTAINS THE INDEX INTO MSGS ARAY OF THE CURRENT ERROR MESSAGE.
LEN - LENGTH OF CURRENT ERROR MESSAGE.
LENGTH - 9 WORD ARRAY REPRESENTING THE LENGTH IN WORDS OF THE 9 DATA TYPES 0-8 RESPECTIVELY
STKIND - COUNT OF NUMBER OF NESTED REPEATS AND INDEX TO THE CURRENT TOP OF THE STACK
STKPRM - 4 WORD ARRAY FOR STACK OF PARENTHESIS FLAGS
STKREP - 4 WORD ARRAY FOR STACK OF REPEAT COUNTS
STKTOK - 4 WORD ARAY FOR STACK OF TOKEN POINTERS
TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED
TYPET - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE

---

SUBROUTINES AND FUNCTIONS CALLED:
EXEC, XPRDS, XRNOD

POL ROUTINES INCLUDED:
XPRDS, STRING, SUBSCR, REPET, ERRMSG
BEGIN XPRODS
IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
CALL XPRDS TO PROMPT USER AND DECODE RESPONSE
DO UNTIL EOS TOKEN IS FOUND
CASE TOKEN (NUMBER; NUMBER; NUMBER; STRING; NULL;)
NUMBER:
ERROR IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERRMSG
SET MESSAGE NUMBER TO XPD2
ERROR IF THERE IS NO ROOM IN DATA AREA PERFORM ERRMSG
SET PREVIOUS TOKEN TO DATA
CALL XPMOV TO MOVE DATA INTO DATA AREA
INCREMENT POINTER IN DATA AREA
INCREMENT POINTER TO NEXT TOKEN
STRING:
PERFORM STRING
SMALL:
SET MESSAGE NUMBER TO XPD2
ERROR IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERRMSG
SET PREVIOUS TOKEN TO DATA
INCREMENT POINTER IN DATA AREA
INCREMENT POINTER TO NEXT TOKEN
SUBSCR:
PERFORM SUBSCR
REPEAT:
PERFORM REPET
CLOSECASE
DO UNTIL TOKEN IS NOT A CLOSED PARENTHESIS
IF STACK IS NOT EMPTY AND
(PREVIOUS TOKEN IS DATA AND TOP OF STACK PAREN FLAG = 0) OR
(TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG = 1) THEN
INCREMENT TOP OF STACK REPEAT COUNT BY 1
IF TOP OF STACK REPEAT COUNT > 0 THEN
RESET TOKEN POINTER TO TOP OF STACK INDEX
ELSE
POP TOP ENTRY ON STACK
IF TOKEN IS A CLOSED PAREN THEN
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
ENDIF
ENDIF
ENDDO
SET XPRDS RETURN CODE = XPRDS RETURN CODE
ELSIF
SET XPRDS RETURN CODE TO BAD INVALID PARAMETER 1ST
ENDIF
ENDIF
END XPRDS
733 1 BEGIN STRING
734 2 ERREXIT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERRMSG
735 2 DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
736 2 ERREXIT IF TYPE OF RESPONSE > TYPE REQUESTED AND
737 2 ERREXIT IF TYPE IS NOT FREE PERFORM ERRMSG
738 2 IF TYPE IS NOT FREE THEN
739 3 SET EFFECTIVE LENGTH = LENGTH REQUESTED
740 3 END IF
741 2 SET MESSAGE NUMBER TO XPOZ
742 2 ERREXIT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERRMSG
743 2 CALL XMOV TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH
744 2 CALL XMOV TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH
745 2 SET PREVIOUS TOKEN IS DATA
746 2 INCREMENT POINTER IN DATA AREA
747 2 INCREMENT TO NEXT TOKEN
748 1 END STRING
749 1 *
750 1 *
751 1 *
752 1 BEGIN SUBSCR
753 2 INCREMENT POINTER TO NEXT TOKEN
754 2 SET MESSAGE NUMBER TO XPOZ
755 2 ERREXIT IF TOKEN IS NOT AN INTEGER TO PERFORM ERRMSG
756 2 IF I-DIMENSION > 1 THEN
757 3 SET I TO INTEGER VALUE
758 3 INCREMENT POINTER TO NEXT TOKEN
759 3 ERREXIT IF TOKEN IS NOT AN INTEGER OR
760 3 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
761 3 SET MESSAGE NUMBER TO XPOZ
762 3 ERREXIT IF INTEGER > I-DIMENSION TO PERFORM ERRMSG
763 3 ERREXIT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERRMSG
764 2 ELSE
765 3 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
766 3 SET MESSAGE NUMBER TO XPOZ
767 3 ERREXIT IF SUBSCRIPT IS OUT OF RANGE
768 2 END IF
769 2 ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT
770 2 INCREMENT POINTER BY 3 TOKENS
771 2 SET PREVIOUS TOKEN = SUBSCRIPT
772 2 SET MESSAGE NUMBER TO XPOZ
773 2 ERREXIT IF TOKEN IS AN EOS OR
774 2 ERREXIT IF TOKEN IS A REPEAT OR
775 2 ERREXIT IF TOKEN IS A CLOSE PAREN OR
776 2 ERREXIT IF TOKEN IS A SUBSCRIPT TO PERFORM ERRMSG
777 1 END SUBSCR
778 1 XPBD
1 BEGIN REPET
2 SET MESSAGE NUMBER TO XPOS
3 ERREXIT IF TOKEN IS AN EOS TO PERFORM ERRMSG
4 INCREMENT STACK POINTER
5 SET MESSAGE NUMBER TO XPOS
6 ERREXIT IF THERE ARE TOO MANY NESTED REPEATS PERFORM ERRMSG
7 PUSH REPEAT COUNT ON STACK
8 SET PARENTHESIS FLAG TO 0
9 IF TOKEN IS AN OPEN PARENTHESIS THEN
10 INCREMENT POINTER TO NEXT TOKEN
11 SET PARENTHESIS FLAG TO 1
12 ENDIF
13 PUSH TOKEN INDEX AND PAREN FLAG ON STACK
14 SET PREVIOUS TOKEN IS A REPEAT
15 End REPET
16 Call EXEC TO WRITE ERROR MESSAGE
17 BEGIN ERRMSG
18 PERFORM XPRDS - NO RETURN
19 End ERRMSG

END
FORTRAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE

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

WRTES "PRMLEM" 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
PRMLEM - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
STRING USED FOR THE USER PROMPT!
PROMPT - (INTEGER, PRMLEM WORDS) IS THE CHARACTER STRING
USED AS THE USER PROMPT.
COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE
COMMUNICATIONS BUFFER (COMBUF).

OUTPUTS FROM CALLING SEQUENCE:

COMBUF - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE
ENCODED USER RESPONSE
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 ENTERED CR. BUFFER CONTAINS NO INFORMATION
3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
34 CHARACTERS OR 17 WORDS.

INTERNAL VARIABLES

COUNT - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN
A CHARACTER STRING AND NUMBER DIGITS IN A NUMBER.
DOLIMT - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE
DBLVD - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE
VALUE
FLGCOM - COMMA FLAG
O - LAST CHARACTER NOT A COMMA
1 - LAST CHARACTER WAS A COMMA
FLGCOM - CONTINUE FLAG
O - THIS IS NOT A RESPONSE TO A CONTINUE
1 - THIS IS A CONTINUE RESPONSE
FLGNUM - NUMBER FLAG
O - POSITIVE NUMBER
T - NEGATIVE NUMBER
FLGPW - POWER FLAG
O - POSITIVE POWER
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>898</td>
<td>1 BEGIN XPRM</td>
</tr>
<tr>
<td>899</td>
<td>IF PROMPT IS NOT TOO LONG THEN</td>
</tr>
<tr>
<td>900</td>
<td>CALL XMOV &quot;&gt;&quot; MOVE PROMPT INTO OUTPUT AREA</td>
</tr>
<tr>
<td>901</td>
<td>CALL EXEC TO WRITE PROMPT</td>
</tr>
<tr>
<td>902</td>
<td>SET XPRM RETURN CODE TO NORMAL RETURN</td>
</tr>
<tr>
<td>903</td>
<td>SET CONTINUE FLAG OFF</td>
</tr>
<tr>
<td>904</td>
<td>INITIALIZE COMMUNICATIONS BUFFER</td>
</tr>
<tr>
<td>905</td>
<td>:COMLP:</td>
</tr>
<tr>
<td>906</td>
<td>CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS</td>
</tr>
<tr>
<td>907</td>
<td>CALL EXEC TO READ RESPONSE</td>
</tr>
<tr>
<td>908</td>
<td>CALL XRPK TO CONVERT A2 RESPONSE TO R1 FORMAT</td>
</tr>
<tr>
<td>909</td>
<td>IF NUMBER OF WORDS READ IS NOT ZERO THEN</td>
</tr>
<tr>
<td>910</td>
<td>SET COMMA FLAG ON</td>
</tr>
<tr>
<td>911</td>
<td>DO WHILE CONTINUE FLAG IS OFF</td>
</tr>
<tr>
<td>912</td>
<td>IF INPUT BUFFER IS COMpletely SCANNED THEN</td>
</tr>
<tr>
<td>913</td>
<td>SET CONTINUE FLAG ON</td>
</tr>
<tr>
<td>914</td>
<td>IF COMMA FLAG IS ON THEN</td>
</tr>
<tr>
<td>915</td>
<td>CALL EXEC TO WRITE CONTINUE</td>
</tr>
<tr>
<td>916</td>
<td>GO TO :COMLP:</td>
</tr>
<tr>
<td>917</td>
<td>ENDIF</td>
</tr>
<tr>
<td>918</td>
<td>ELSE</td>
</tr>
<tr>
<td>919</td>
<td>ERREXIT IF COMBUF IS FULL PERFORM COMFUL</td>
</tr>
<tr>
<td>920</td>
<td>IF INPUT CHARACTER IS A COMMA THEN</td>
</tr>
<tr>
<td>921</td>
<td>IF COMMA FLAG IS ON THEN</td>
</tr>
<tr>
<td>922</td>
<td>ERREXIT IF COMBUF CANNOT HOLD TOKEN PERFORM COMFUL</td>
</tr>
<tr>
<td>923</td>
<td>STORE NULL FIELD TOKEN IN COMBUF</td>
</tr>
<tr>
<td>924</td>
<td>INCREMENT #WORDS IN COMBUF BY 1</td>
</tr>
<tr>
<td>925</td>
<td>INCREMENT #TOKENS IN COMBUF BY 1</td>
</tr>
<tr>
<td>926</td>
<td>ENDIF</td>
</tr>
<tr>
<td>927</td>
<td>SET COMMA FLAG ON</td>
</tr>
<tr>
<td>928</td>
<td>SET NEXT INPUT CHARACTER</td>
</tr>
<tr>
<td>929</td>
<td>ELSE</td>
</tr>
<tr>
<td>930</td>
<td>SET COMMA FLAG OFF</td>
</tr>
<tr>
<td>931</td>
<td>PERFORM TOKENS</td>
</tr>
<tr>
<td>932</td>
<td>ENDIF</td>
</tr>
<tr>
<td>933</td>
<td>ENDIF</td>
</tr>
<tr>
<td>934</td>
<td>END</td>
</tr>
<tr>
<td>935</td>
<td>ELSE</td>
</tr>
<tr>
<td>936</td>
<td>IF CONTINUE FLAG IS OFF THEN</td>
</tr>
<tr>
<td>937</td>
<td>SET XPRM RETURN CODE TO SAY USER ENTERED CR</td>
</tr>
<tr>
<td>938</td>
<td>ENDIF</td>
</tr>
<tr>
<td>939</td>
<td>ENDIF</td>
</tr>
<tr>
<td>940</td>
<td>ELSE</td>
</tr>
<tr>
<td>941</td>
<td>SET XPRM RETURN CODE TO SAY PROMPT IS TOO LONG</td>
</tr>
<tr>
<td>942</td>
<td>ENDIF</td>
</tr>
<tr>
<td>943</td>
<td>1 END XPRM</td>
</tr>
</tbody>
</table>
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 NOT A QUOTE)
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
ERROR IF LENGTH OF CHARACTER STRING IS 0 OR ERROR IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVALID
EXIT IF THER IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUN
STORE CHARACTER STRING TOKEN IN COMBUF
STORE NUMBER OF CHARACTERS IN COMBUF
CALL XRCPK 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 BEGIN DIGIT
1020 PERFORM QCOL IF DIGIT IS NOT EXHAUSTED THEN
1022 PERFORM DIGIT IF DIGIT IS A - THEN
1025 ELSE IF DIGIT IS NOT A "E" OR A "D" THEN
1028 IF DIGIT IS A "E" THEN
1031 PERFORM END.
1034 ELSE IF DIGIT IS "D" THEN
1037 STORE REPEATED DIGIT IN GROUP BY 2
1040 INCREASE NUMBER OF DIGITS IN GROUP BY 1
1043 ELSE GET NEXT CHARACTER
1046 ELSE END.
1049 END DIGIT.
1052 SET INTEGER = DIGIT AND
1055 IF DIGIT IS A DIGIT AND
1058 ELSE DIGIT IS NOT EXHAUSTED THEN
1061 BEGIN QCOL.
1064 END QCOL.
1067 END DIGIT.
1070 END QCOL.

REPRODUCIBILITY OF THE
ORIGINAL PAGE IS BAD.
1059 1 BEGIN DECPT
1060 2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061 2 GET NEXT CHARACTER
1062 2 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063 3 IF INPUT CHARACTER IS A DIGIT THEN
1064 4 PERFORM DCOL
1065 4 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066 4 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067 3 ENDIF
1068 3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069 4 PERFORM EORD
1070 3 ELSE
1071 4 PERFORM REAL
1072 3 ENDIF
1073 2 ELSE
1074 3 PERFORM REAL
1075 2 ENDIF
1076 1 END DECPT
1076 1 BEGIN EORD
1077   IF INPUT CHARACTER IS AN "E" THEN
1078     SET TYPE FLAG TO "E"
1079   ELSE
1080     SET TYPE FLAG TO "D"
1081   ENDIF
1082   GET NEXT CHARACTER
1083   ERREXIT IF INPUT BUFFER IS EXHAUSTED PERFORM INVAL
1084   IF INPUT CHARACTER IS A - THEN
1085     SET NEGATIVE POWER FLAG ON
1086   ELSE
1087     IF INPUT CHARACTER IS A + THEN
1088       SET NEGATIVE POWER FLAG OFF
1089     GET NEXT CHARACTER
1090   ENDIF
1091   ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
1092   ERREXIT IF INPUT CHARACTER IS NOT A DIGIT PERFORM INVAL
1093   PERFORM DCOL
1094   IF NEGATIVE POWER FLAG IS ON THEN
1095     SET POWER = -POWER
1096   ENDIF
1097   IF TYPE FLAG IS "E" THEN
1098     PERFORM REAL
1099   ELSE
1100     BEGIN DBL
1101       ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
1102       ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
1103       ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
1104       ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1105       SET DOUBLE = DOUBLE * 10 ** POWER
1106       ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1107       IF NEGATIVE NUMBER FLAG IS ON THEN
1108         SET DOUBLE = -DOUBLE
1109       ENDIF
1110       STORE DOUBLE TOKEN IN COMBUF
1111       INCREMENT #WORDS IN COMBUF BY 4
1112       INCREMENT #TOKENS IN COMBUF BY 1
1113     END DBL
1114   ENDIF
1115 1 END EORD
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
FORTRAN CALLING PROCEDURE
CALL XPIIT (LU, RPAMS)

国立 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

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

OUTPUT
RPAMS IS RETURNED TO THE FATHER TASK

NOTES
USES EXEC, PRN

XPIIT DOES NOT RETURN TO THE CALLER.

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

CALL XBIT (BIT, BITHNUM, STRING)

**********

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

**********

INPUT

BIT - INTEGER VALUE OF ZERO OR ONE TO BE SET IN BIT NUMBER
'BITHNUM' OF 'STRING'

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

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

**********

OUTPUT

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

**********

NOTES

USES .ENT, XSET

**********

BEGIN XBIT

TRANSFER CALLING SEQUENCE

COMPUTE ADDRESS OF WORD CONTAINING BITHNUM

COMPUTE BIT NUMBER WITHIN WORD

CALL XSET TO SET/CLEAR BIT

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

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

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

NOTES
USES .ENTR

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

CALL XRDB (DOUBLE, ASCII)

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

INPUT

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

OUTPUT

ASCII - NINE WORD ASCII CHARACTER STRING REPRESENTATION OF

'DOUBLE'

LOCAL

D - WORKING LOCATION FOR ABSOLUTE VALUE OR 'DOUBLE'

REPEATEDLY MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

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

BEGIN XRDB

SET SIGN FIELD

MOVE ABSOLUTE VALUE OF 'DOUBLE' INTERNAL

COMPUTE EXPONENT

SET SIGN AND VALUE OF EXPONENT FIELD

REDUCE VALUE TO RANGE OF 1 < VALUE < 10

EXTRACT FIRST DIGIT, MERGE WITH SIGN AND STORE FIELD

EXTRACT SECOND DIGIT, MERGE WITH DECIMAL AND STORE FIELD

DO FOR NEXT FIVE PAIRS OF DIGITS

MULTIPLY BY 100 TO EXTRACT PAIR

EXTRACT DIGITS AND STORE FIELD

END XRDB
FORTRAN CALLING PROCEDURE

1 CD0 CALL XREQ

2 CD3

3 CD0

4 CD0

5 *********

6 CD1 MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE

7 CD1

8 CD1

9 CD1

10 *********

11 CD2 INPUT

12 CD2 COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPTR

13 CD2 ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER

14 CD2

15 CD2

16 *********

17 CD3 OUTPUT

18 CD3 COMMON XE - REQPTR

19 CD3 CLASS I/O WRITE/READ TO CLASS 'CLASNO'

20 CD3

21 CD3 REQBUF AND RESPONSE IS PRINTED IF REQUESTED

22 CD3

23 *********

24 CD5 COMMUNICATES WITH FDS MANAGER FATHER TASK

25 CD5 USES EXEC, IAND, XRMOV, XRMSG, XRO6, XRSFL, XUDPL, XVPAW

26 CD5

27 CD5

28 *********

29 CD5

30 * *

31 *

32 *

33 * *

34 BEGIN XRQ

35 PERFORM TRACE

36 OUTPUT REQUESTS TO MANAGER

37 REQUEST AMA MANAGEMENT AND WAIT FOR RESPONSE

38 RETRIEVE MANAGER RESPONSE

39 PERFORM TRACE

40 RETURN RESPONSE IN REQPTR

41 END XRQ

42 BEGIN TRACE

43 IF TRACE REQUESTED THEN

44 DO FOR EACH REQUEST

45 CALL XUDPL TO FORMAT LINE

46 OUTPUT LINE

47 ENDDO

48 OUTPUT PARM1 AND PARM2

49 EXIT

50 END TRACE
INTEGER FUNCTION

XREX(START, LENGTH, SOURCE)

**INPUT**

1. **D0** START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
2. **D0** (SIGN BIT = 0)
3. **D0** LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
4. **D0** SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED

**NOTES**

USES .ENTR

BEGIN XREX

TRANSFER CALLING SEQUENCE
2
IF START NOT = 0
2
THEN
2
CONSTRUCT SHIFT
2
LOAD A WITH SOURCE
2
SHIFT BA LEFT START BITS
2
ELSE
2
LOAD A WITH SOURCE
2
ENDIF
2
SAVE A
2
CLEAR B
2
CONSTRUCT SHIFT
2
RESTORE A
2
SHIFT BA LEFT LENGTH BITS
2
MOVE RESULT FROM B TO A

END XREX
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>217</td>
<td>FORTRAN CALLING PROCEDURE.</td>
</tr>
<tr>
<td>218</td>
<td>CALL XRE14 (REAL, ASCII)</td>
</tr>
<tr>
<td>220</td>
<td>CONVERT A SINGLE PRECISION REAL NUMBER TO AN ASCII STRING IN</td>
</tr>
<tr>
<td>221</td>
<td>1PE14.6 FORMAT</td>
</tr>
<tr>
<td>222</td>
<td>INPUT</td>
</tr>
<tr>
<td>223</td>
<td>REAL - TWO WORD SINGLE PRECISION REAL NUMBER TO BE CONVERTED</td>
</tr>
<tr>
<td>225</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>226</td>
<td>ASCII - SEVEN WORD ASCII CHARACTER STRING REPRESENTATION OF</td>
</tr>
<tr>
<td>227</td>
<td>'REAL'</td>
</tr>
<tr>
<td>228</td>
<td>LOCAL</td>
</tr>
<tr>
<td>229</td>
<td>R - WORKING LOCATION FOR ABSOLUTE VALUE OR 'REAL' REPEATEDLY</td>
</tr>
<tr>
<td>230</td>
<td>MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS</td>
</tr>
<tr>
<td>231</td>
<td>NOTES</td>
</tr>
<tr>
<td>232</td>
<td>USES FLOAT, IAND, IFIX, IOR, KCVT, XREXT, XRSFL</td>
</tr>
<tr>
<td>245</td>
<td>*</td>
</tr>
<tr>
<td>246</td>
<td>*</td>
</tr>
<tr>
<td>247</td>
<td>*</td>
</tr>
<tr>
<td>248</td>
<td>*</td>
</tr>
<tr>
<td>249</td>
<td>BEGIN XRE14</td>
</tr>
<tr>
<td>250</td>
<td>SET SIGN FIELD</td>
</tr>
<tr>
<td>251</td>
<td>MOVE ABSOLUTE VALUE OF 'REAL' INTERNAL</td>
</tr>
<tr>
<td>252</td>
<td>COMPUTE EXPONENT</td>
</tr>
<tr>
<td>253</td>
<td>SET SIGN AND VALUE OF EXPONENT FIELD</td>
</tr>
<tr>
<td>254</td>
<td>REDUCE VALUE TO RANGE OF 1 &lt;= VALUE &lt; 10</td>
</tr>
<tr>
<td>255</td>
<td>EXTRACT FIRST DIGIT AND STORE FIELD</td>
</tr>
<tr>
<td>256</td>
<td>SET DECIMAL FIELD</td>
</tr>
<tr>
<td>257</td>
<td>DO FOR NEXT THREE PAIRS OF DIGITS</td>
</tr>
<tr>
<td>258</td>
<td>MULTIPLY BY 100 TO EXTRACT PAIR</td>
</tr>
<tr>
<td>259</td>
<td>EXTRACT DIGITS AND STORE FIELD</td>
</tr>
<tr>
<td>260</td>
<td>END XRE14</td>
</tr>
</tbody>
</table>
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 XRPACK
289 1 CD5
290 1 C********
291 1 * 1
292 1 * 1
293 1 * 1
294 1 * 1
295 1 BEGIN XR16
296 2 BLANK WORKING SPACE
297 2 CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING
298 2 SET SIGN OF 'INTEGER' IN 'ASCII'
299 2 CALL XRPACK TO CONVERT FROM A1 TO A2 FORMAT
300 1 END XR16
FORTRAN CALLING PROCEDURES

**CALL XRLCK (RCODE)**

**CALL XRLUK (RCODE)**

---

**XRLCK AND XRLUK PROVIDE A MECHANISM FOR SERIALIZING THE UPDATE**

**OF FDS GLOBAL SYSTEM TABLES AND FILES. THE RESOURCE NUMBER**

**STORED IN THE XVSTB RESIDENT STATUS TABLE IS USED AS THE**

**LOCKING MECHANISM**

---

**INPUT**

**XVSTB RESOURCE NUMBER**

**OUTPUT**

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

**LOCAL**

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

**1 - RN LOCKED**

**4 - RN UNLOCKED**

**NOTES**

**USES .ENTR, RNRR.**

**THIS ROUTINE MAY NOT BE OVERLayed**
1 BEGIN XLCK
2 SET REQUEST FOR LOCK FUNCTION
3 BEGIN XLCK
4 PERFORM RLOCK
5 END XLCK
6 BEGIN XLCK
7 SET REQUEST FOR UNLOCK FUNCTION
8 BEGIN XLCK
9 PERFORM RLOCK
10 END XLCK
11 IF REQUEST IS CONSISTENT WITH STATUS
12 THEN
13 SET NEW STATUS
14 IF FDS IN IBL IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
15 THEN
16 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
17 ENDIF
18 CLEAR RETURN CODE
19 ELSE
20 SET RETURN CODE
21 ENDIF
22 END RLOCK
23 END RLOCK
24 END RLOCK
25 END RLOCK
26 END RLOCK
27 END RLOCK
28 END RLOCK
29 END RLOCK
30 END RLOCK
31 END RLOCK
32 END RLOCK
33 END RLOCK
34 END RLOCK
35 END RLOCK
36 END RLOCK
37 END RLOCK
38 END RLOCK
39 END RLOCK
40 END RLOCK
41 END RLOCK
42 END RLOCK
43 END RLOCK
44 END RLOCK
45 END RLOCK
46 END RLOCK
47 END RLOCK
48 END RLOCK
49 END RLOCK
50 END RLOCK
51 END RLOCK
52 END RLOCK
53 END RLOCK
54 END RLOCK
55 END RLOCK
56 END RLOCK
57 END RLOCK
INTEGER FUNCTION

*00  XRLOC(A)

*00  RETURN THE 16-BIT MAPPED ADDRESS OF A

*00  INPUT

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

*00  OUTPUT

*00  XRLOC - 16-BIT ADDRESS OF A

*00  NOTES

*00  NO EXTERNAL REFERENCES

*00

*00

*00

BEGIN XRLOC

2  TRANSFER CALLING SEQUENCE

2  LOAD THE ADDRESS OF THE CALLING PARAMETER

1  END XRLOC
```plaintext
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>390</td>
<td>BEGIN XMOV</td>
</tr>
<tr>
<td>391</td>
<td>TRANSFER CALLING SEQUENCE</td>
</tr>
<tr>
<td>392</td>
<td>MOVE LENGTH WORDS FROM SOURCE TO OBJECT</td>
</tr>
<tr>
<td>393</td>
<td>USES, ENTR</td>
</tr>
<tr>
<td>394</td>
<td></td>
</tr>
<tr>
<td>395</td>
<td></td>
</tr>
<tr>
<td>396</td>
<td></td>
</tr>
<tr>
<td>397</td>
<td></td>
</tr>
<tr>
<td>398</td>
<td></td>
</tr>
<tr>
<td>399</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>401</td>
<td></td>
</tr>
<tr>
<td>402</td>
<td></td>
</tr>
<tr>
<td>403</td>
<td></td>
</tr>
<tr>
<td>404</td>
<td></td>
</tr>
<tr>
<td>405</td>
<td></td>
</tr>
<tr>
<td>406</td>
<td></td>
</tr>
<tr>
<td>407</td>
<td></td>
</tr>
<tr>
<td>408</td>
<td></td>
</tr>
<tr>
<td>409</td>
<td></td>
</tr>
<tr>
<td>410</td>
<td></td>
</tr>
<tr>
<td>411</td>
<td></td>
</tr>
<tr>
<td>412</td>
<td></td>
</tr>
<tr>
<td>413</td>
<td></td>
</tr>
<tr>
<td>414</td>
<td></td>
</tr>
<tr>
<td>415</td>
<td></td>
</tr>
<tr>
<td>416</td>
<td></td>
</tr>
<tr>
<td>417</td>
<td></td>
</tr>
<tr>
<td>418</td>
<td></td>
</tr>
<tr>
<td>419</td>
<td>END XMOV</td>
</tr>
</tbody>
</table>
```

In the context of the image, it seems to be a part of a Fortran program, specifically a subroutine for moving words from a source array to an object array. The subroutine is named XMOV, and it is called with parameters that include the length of the words to be moved.
FORTRAN CALLING PROCEDURE
CALL XRM SG (NUMBER, LOCATE, LENGTH, SOURCE)

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

INPUT
NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
A - AREA INDICATOR AS FOLLOWS
   1 - AS
   2 - XA
   3 - XE
   4 - XI
   5 - XS
   6 - YT
   7 - X
   8 - XA
   9 - DF
   10 - SC
NN - MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'
WORDS OF 'SOURCE' TO BE OUTPUT
LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEDE
'SOURCE' (NOT USED IF 'NN' OF 'NUMBER' IS ZERO)
LENGTH - INTEGER NUMBER OF 'SOURCE' TO BE INSERTED INTO
MESSAGE. ZERO INDICATES NO INSERTION
SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED
IF 'LENGTH' IS ZERO)
COMMON

LU - USERS LOGICAL UNIT NUMBER

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

NOTES
USES FDS SYSTEM MESSAGE FILE XRM SG
USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XMOV, XUDOG

REFERENTIAL INTEGRITY OF THE
1 BEGIN XRMSE
2 SEPERATE 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 XRMOV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
15 ELSE
16 EXIT TO :ERROR:
17 ENDIF
18 CALL XRMOV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
19 IF MESSAGE NUMBER > 0
20 THEN
21 CALL XRMOV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
22 ENDIF
23 ELSE
24 ENDIF
25 SET AREA IN PREFIX
26 :ERROR: CALL XRMOV TO MOVE 'XRMSE ERROR' INTO BUFFER
27 CALL XRMOV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
28 ENDIF
29 OUTPUT BUFFER TO USER'S TERMINAL
30 IF DEBUG IS REQUESTED
31 THEN
32 CALL XUDOPT
33 ENDIF
34 END XRMSE
INTEGER FUNCTION

XMPF80 (BIT, BITNUM, STRING)

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

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

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

INPUT

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

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

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

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

OUTPUT

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

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

NOTES

USES .ENTR

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.
BEGIN XMKDB
1 COMPUTE STARTING WORD NUMBER
2 COMPUTE STARTING BIT NUMBER
3 LOAD STARTING WORD
4 SHIFT WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
5 INITIALIZE BIT COUNT
6 DO WHILE SIGN BIT IS NOT EQUAL TO BIT
7 IF WORD NOT FINISHED
8 THEN
9 SHIFT WORD LEFT
10 INCREMENT BIT COUNT
11 ELSE
12 DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
13 LOAD NEXT WORD
14 END DO
15 CLEAR BIT COUNT
16 ENDIF
17 END DO
18 RETURN VALUE OF MATCHING BIT NUMBER
19 END XMKDB
542 1 000 FORTRAN CALLING PROCEDURE
543 1 000 CALL XRO6 (BINARY, OCTAL)
544 1 000
545 1 000
546 1 000
547 1 001 CONVERT A WORD FROM BINARY TO SIX CHARACTER OCTAL REPRESENTATION
548 1 001
549 1 001
550 1 001
551 1 001
552 1 001
553 1 001 INPUT
554 1 001 BINARY - BINARY WORD TO BE CONVERTED
555 1 001
556 1 001
557 1 001
558 1 001
559 1 001 OCTAL - THREE WORD ARRAY CONTAINING OCTAL REPRESENTATION OF
560 1 001 "BINARY" IN ASCII FORMAT (66)
561 1 001
562 1 001
563 1 001 NOTES
564 1 001 USES .ENTRY
565 1 001
566 1 001
567 1 001
568 1 001
569 1 001 BEGIN XRO6
570 1 001 TRANSFER CALLING SEQUENCE
571 1 001 FORM 18-BIT WORD USING 8 AND 2 MOST SIGNIFICANT BITS OF A
572 1 001 SET BYTE FLAG HIGH
573 1 001 DO FOR EACH PAIR OF OCTAL DIGITS (3)
574 1 001 SET PREFIX BITS (00000000)
575 1 001 SHIFT IN DIGIT
576 1 001 IF BYTE FLAG SET HIGH
577 1 001 THEN
578 1 001 SHIFT FOR ACCUMULATION OF LOW BYTE
579 1 001 ELSE
580 1 001 STORE PAIR OF DIGITS IN OCTAL(1)
581 1 001 CLEAR FOR NEXT PAIR OF DIGITS
582 1 001 ENDIF
583 1 001 FLIP BYTE FLAG
584 1 001 LWD XRO6
585 1 001
586 1 001
587 1 001
588 1 001
589 1 001
590 1 001
FORTRAN CALLING PROCEDURE

CALL XRPCX (LENGTH, UNPKED, PACKED)

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

+1 Length - positive integer number of characters in unpked
+2 UNPKED - array of characters in r1 format
+3 PACKED - array of (LENGTH+1)/2 words in A2 format. if length is
odd, the last word will be blank filled.

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

BEGIN XRPCX

TRANSFER CALLING SEQUENCE

SET BYTE FLAG FOR HIGH BYTE

INITIALIZE PACKED POINTER

DO FOR EACH CHARACTER IN UNPKED

IF BYTE FLAG SET HIGH

THEN

LOAD A WITH NEXT WORD OF UNPKED

SHIFT CHARACTER INTO HIGH BYTE

ELSE

INCLUSIVE OR NEXT WORD OF UNPKE INTO A

STORE A IN PACKED

INCREMENT POINTER

ENDIF

FLIP BYTE FLAG

ENDDO

IF BYTE FLAG SET LOW

THEN

INCLUSIVE OR BLANK INTO LOW BYTE

ENDIF

STORE A IN PACKED

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

*0 CALL XRSET (START, LENGTH, SOURCE, OBJECT)

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

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

*0  OUTPUT
*0  OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED

*0  NOTES
*0  USES .ENTR

**
*0  BEGIN XRSET

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

*1  END XRSET
INTEGER FUNCTIONS

1 \*D0
2 \*D0
3 \*D0
4 \*D0
5 \*D0
6 \*D1

1 D1 SHIFT SOURCE LEFT/RIGHT LOGICALLY COUNT BITS

1 D2 INPUT
2 D2 COUNT - POSITIVE INTEGER SPECIFYING NUMBER OF BITS TO SHIFT
3 D2 SOURCE - WORD TO BE SHIFTED

1 D3

1 D5 NOTES
2 D5 USES .ENTR

1 D6

1 D6 BEGIN XRSFL
2 SET FOR LEFT SHIFT
3 PERFORM SHIFT (FLAG, COUNT, SOURCE)
4 END XRSFL
5 BEGIN XRSFR
6 SET FOR RIGHT SHIFT
7 PERFORM SHIFT (FLAG, COUNT, SOURCE)
8 END XRSFR
9 BEGIN SHIFT
10 TRANSFER CALLING SEQUENCE
11 CONSTRUCT SHIFT INSTRUCTION
12 LOAD A WITH SOURCE
13 CLEAR B
14 SHIFT BA AS SPECIFIED
15 END SHIFT
FORTRAN CALLING PROCEDURE

CALL XRUN (NAME6, NAME4)

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

INPUT

NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME

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

OUTPUT

NOTES

USES .ENTR

LOCATE LAST NON-BLANK CHARACTER OF NAME4

BLANK THAT CHARACTER

ENDIF

END XRUN
FORTRAN CALLING PROCEDURE

CALL XRPK (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
PACKED

WARNING:
CAUTION: XRPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER
STRINGS.
839 1 BEGIN XRUPIK
840 2 TRANSFER CALLING SEQUENCE
841 2 TURN ON BLANK REMOVAL
842 2 INITIALIZE COUNT
843 2 DO FOR EACH WORD OF PACKED
844 3 LOAD A WITH NEXT WORD
845 3 ROTATE A 8 BITS
846 3 DO FOR EACH BYTE OF WORD
847 4 AND OFF HIGH BYTE
848 4 IF A = QUOTE MARK
849 4 THEN
850 5 CHANGE BLANK REMOVAL OPTION
851 4 ENDIF
852 4 IF BLANK REMOVAL IS ON
853 4 THEN
854 5 IF A NOT = BLANK
855 5 THEN
856 6 INCREMENT COUNT
857 6 STORE A IN UNPKED
858 5 ENDIF
859 4 ELSE
860 5 INCREMENT COUNT
861 5 STORE A IN UNPKED
862 4 ENDIF
863 4 RELOAD A WITH WORD
864 3 ENDDO
865 2 ENDDO
866 2 RETURN VALUE OF COUNT
867 1 END XRUPIK
FORTRAN CALLING PROCEDURE

CALL XRISP (CHSTR, LENGTH)

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

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 LENGTH WORDS IN A2 FORMAT
LENGLISH - LENGTH OF CHSTR IN WORDS

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

OUTPUT

CHSTR - CHARACTER STRING OF LENGTH WORDS IN A2 FORMAT WITH
DUPLICATE BLANKS REMOVED
LENGLISH - 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 CHARACTERS WITHIN QUOTATION MARKS DOES NOT
BEGIN UNTIL THE FIRST OCCURRENCE OF QUOTATION MARKS ON EACH
ENTRY
3. THE FIRST/LAST CHARACTER IN A CHARACTER STRING HAS THE
HIGH ORDER BIT OF THE QUOTE CHARACTER 'ON' TO INDICATE
BEGIN/END OF A CHARACTER STRING. THIS BIT IS TURNED 'OFF'
BY THIS PROGRAM PRIOR TO EXIT. THIS CHANGE MADE TO
INTEGRATE QUOTE MARKS WITHIN CHARACTER STRINGS.
OTHER PROCESSORS CHANGED FOR THIS MODIFICATION WERE:
A. XTLAN, XPROM
B. XILSS/XILSB

************
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 , THEN:
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 ENDIF
18 ELSE
19 CLEAR BLANK FLAG
20 ENDIF
21 ENDIF
22 ENDDO
23 STORE CHARACTER AT INDEXED POSITION
24 INCREMENT STORE INDEX
25 ENDDO
26 IF NUMBER OF CHARACTERS STORED IS 000
27 STORE ONE MORE BLANK
28 ENDIF
29 CONVERT STORE INDEX TO WORD COUNT AND RETURN
30 END XR1SP
1 CDO  FORTRAN CALLING PROCEDURE
2 CDO
3 CDO
4 CDO
5 CDO  CALL XRLBS(XSERE)
6 CDO
7 CDO
8 CDO  XSERE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR
9 CDO
10 CDO
11 CDO
12 CDO
13 CDO
14 CDO
15 CDO
16 CDO
17 CDO
18 CDO
19 CDO
20 CDO
21 CDO
22 CDO
23 CDO
24 CDO
25 CDO
26 CDO
27 CDO
28 CDO
29 CDO
30 CDO  NOTES
31 CDO
32 CDO
33 CDO
34 CDO
35 CDO
36 CDO
37 CDO
38 CDO
39 CDO
40 CDO
41 CDO
42 CDO
43 CDO
44 CDO
45 CDO
BEGIN XSERE
   DO UNTIL A % OR 'EXIT' IS ENTERED
   CALL XSSPN TO BUILD A PROMPT BASED ON PROMPT MODE FOR THE
   NEXT TABLE ENTRY (INDICATED BY TBMDX)
   CALL XTCPN TO ISSUE THE PROMPT AND RETURN RESPONSE
   ERNFIT IF ERROR IN XTCPN :ERR10:
   EXIT XSSPN IF X WAS ENTERED
   IF NOTHING (ONLY CR) ENTERED, THEN
      IF PROMPT MODE IS NOT 'ALL', THEN
         CALL XRSNG - 'INVALID INPUT'
      ENDIF
   ELSE
      CALL XSSPN TO PROCESS INPUT BASED ON PROMPT MODE,
      CURRENT TABLE ENTRY (TBMDX), AND PROMPTED SERJENCE
      NUMBER (PRNUM)
   ENDIF
   ENDDO
   BUILD AWS REQUEST TO DELETE/VERIFY ABSENCE OF MEDZTAB
   CALL XSPCFK TO PACK THE TABLE BUFFER (REMOVE DELETED ENTRIES)
   BUILD AWS REQUEST TO ALLOCATE NEWTAB
   IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
      CALL XSSPN TO PROCESS THE REQUESTS
   ENDIF
   IF THE ALLOCATE REQUEST FAILED, THEN
      CALL XRSNG - 'AWS/WA FULL, SERJENCE TABLE NOT STORED'
   ELSE
      CALL XRSNG - 'VACUOUS TABLE -- NOT STORED'
   ENDIF
   ENDF
   EXIT XSERE
   :ERR10: CALL XRSNG - 'SYSTEM ERROR'
   END XSERE
FORTRAN CALLING PROCEDURE
CALL ISPRM

ISPRM BUILDS SEQUENCE TABLE EDITOR PROMPTS.

INPUT
COMMON KX - LU
COMMON KB - DEBUG, NUMENT, PRNTMD, TABNDX, WBUF

OUTPUT
COMMON KB - PRMLEN, PROMPT, SERRNO, TABNDX

USES ROUTINES
XR16
XRN0V
XRM5G
XASET
XUG6G
BEGIN XSPRM

IF PROMPT MODE IS ALL, THEN

IF NUMBER OF ENTRIES (NUMENT) > 0, AND

ENTRIES EXIST BEYOND TABLE ENTRY INDEX (TABINDX), THEN

DO UNTIL A NON-DELETED ENTRY IS FOUND

INCREMENT TABLE ENTRY INDEX (TABINDX) TO NEXT ENTRY (+7)

ENDDO

BUILD PROMPT OF THE FORM ' "NUMNUM-PROC,INT"'

SET PROMPT SEQUENCE NUMBER (SEQNO) TO SEQUENCE NO. OF ENTRY

ELSE

SET PROMPT NO. TO CREATE

ENDIF

IF PROMPT MODE IS CREATE, THEN

SET TABLE ENTRY INDEX (TABINDX) TO NEXT ENTRY (NUMENT * 7 + 1)

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

IF SEQUENCE NO. OF Last Entry > 32760, THEN

CALL XPSRD = 'UNABLE TO BUILD SEQUENCE NO. > 32760'

SET PROMPT MODE TO UPDATE

ELSE

SET PROMPT SEQUENCE NUMBER (SEQNO) TO NEXT MULTIPLE OF 100

BEYOND SEQUENCE NUMBER OF LAST TABLE ENTRY

ENDIF

ELSE

SET PROMPT SEQ. NO. (SEQNO) TO BE 100

ENDIF

IF PROMPT MODE IS NOT UPDATE, THEN

BUILD PROMPT OF THE FORM ' "NUMNUM"'

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      CALL XSNPT
158 1 CD0      ********
159 1 CD0      XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE
160 1 CD0      TABLE EDITOR
161 1 CD0      ********
162 1 CD0      INPUT
163 1 CD0      COMMON XE - COMBUF, COMPTR, LU, TOKENS
164 1 CD0      COMMON XB - DEBUG, DIRECT, NUMDIR, NUMENT, PJWTMD
165 1 CD0      SEQNO, TABNOX, WKBUF
166 1 CD0      ********
167 1 CD0      OUTPUT
168 1 CD0      COMMON XE - COMPTK
169 1 CD0      COMMON XB - INSERT, IRETC, NUMENT, PRMTMD, SEQNO,
170 1 CD0      TABNOX, TABSIZ, WKBUF
171 1 CD0      ********
172 1 CD0      NOTES
173 1 CD0      USES ROUTINES
174 1 CD0     :"35 XRM SG
175 1 CD0      XSDEL
176 1 CD0      XSEM T
177 1 CD0      XSLIS
178 1 CD0      XSNUM
179 1 CD0      XSPMT
180 1 CD0      XUDGR
181 1 CD0      ********
200 1 BEGIN XSMPT
201 2 IF PROMPT MODE IS UPDATE, THEN
202 3 IF TOKEN INPUT IS AN INTEGER, THEN
203 4 ERREXIT IF INTEGER < 1 :ERROR1:
204 5 RETAIN INTEGER AS SEQUENCE NO. (SEGNO)
205 6 INCREMENT TO NEXT TOKEN
206 7 ERREXIT IF TOKEN IS NOT "#" :ERROR1:
207 8 INCREMENT TO NEXT TOKEN
208 9 START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED
209 10 EXIT IF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SEGNO.)
210 11 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
211 12 SET INSERT FLAG TO ZERO INDICATING REPLACEMENT OF ENTRY
212 13 EXIT IF SEQUENCE NO. OF ENTRY > SEQUENCE NO. INPUT (SEGNO)
213 14 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
214 15 SET INSERT FLAG TO 1 INDICATING INSERT NECESSARY
215 16 GORELSE
216 17 INCREMENT TO NEXT TABLE ENTRY
217 18 ENDLOOP
218 19 SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE
219 20 SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE
220 21 ENDSEARCH
221 22 CALL XSEND TO REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG
222 23 ELSE
223 24 ERREXIT IF TOKEN IS NOT A NAME :ERROR1:
224 25 START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED
225 26 EXIT IF NAME INPUT IS DIRECTIVE
226 27 SET INDEX TO DIRECTIVE LIST ENTRY
227 28 GORELSE
228 29 INCREMENT TO NEXT DIRECTIVE
230 30 ENDLOOP
231 31 ENDSEARCH
232 32 CASE (XSLIS, XSDL, XSPMT, XSNUM, XSMRG), INDEX
233 33 ENDIF
234 34 ELSE, PROMPT MODE IS NOT UPDATE
235 35 IF TOKEN IS #, THEN
236 36 SET PROMPT MODE TO UPDATE
237 37 F AN & IS INPUT, THEN
238 38 ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR1:
239 39 MARK THIS TABLE ENTRY AS DELETED
240 40 DO FROM END OF TABLE UNTIL A NONDELETED ENTRY IS FOUND
241 41 IF TABLE ENTRY IS MARKED FOR DELETION, THEN
242 42 DECREMENT NUMBER OF TABLE ENTRIES BY ONE
243 43 ENDF
244 44 ENDF
245 45 ELSE
246 46 IF PROMPT MODE IS ALL, THEN
247 47 SET INSERT FLAG TO TWO TO INDICATE EXTENSION OF TABLE
248 48 ELSE
249 49 SET INSERT FLAG TO ZERO TO INDICATE REPLACEMENT OF TABLE ENTRY
250 50 ENDF
251 51 CALL XSEND TO BUILD ENTRY BASE ON INSERT FLAG
252 52 EXIT XSMPT
253 53 :ERROR1: CALL XRMSEG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
CD0  FORTRAN CALLING PROCEDURE
CD0
CD0  CALL XSENT
CD0
CD0  C*****
CD0  C*****
CD0  C***** XSENT CONSTRUCTS SEQUENCE TABLE ENTRIES
CD0  CD2  COMMON XE - COMBUF, COMPTR, TOKENS, LU
CD0  CD2
CD0  CD2  COMMON XB - DEBUG, LIBID, NUMENT, PRMTMD,
CD0  CD2  SEGO, TABDIX, WBUF
CD0
CD0  C*****
CD0  CD3  OUTPUT
CD0  CD3  COMMON XE - COMPTR
CD0  CD3
CD0  CD3  COMMON XB - INTNAME, NUMENT, PRCNAM, PRMTMD,
CD0  CD3  SEGO, TABDIX, WBUF
CD0  CD3
CD0  C*****
CD0  CD5  NOTES
CD0  CD5
CD0  CD5  USES ROUTINES
CD0  CD5  XRCPR
CD0  CD5  XREXT
CD0  CD5  XRMV
CD0  CD5  XRMG
CD0  CD5  XRSET
CD0  CD5  XRPCK
CD0  CD5  XUE9G
CD0  CD5
CD0  C*****
CD0  CD7  BEGIN XSENT
CD0  CD7
CD0  CD7  ERREXIT IF TOKEN INPUT IS NOT A NAME :ERROR1:
CD0  CD7  RETAIN THIS NAME AS PROC. NAME
CD0  CD7  INCREMENT TO NEXT TOKEN
CD0  CD7  START SEARCH UNTIL ALL ENTRIES OF LIBRARY SEARCHED
CD0  CD7  EXIT IF LIBRARY ENTRY = PROC. NAME
CD0  CD7  ORELSE
CD0  CD7  4  INCREMENT TO NEXT ENTRY
CD0  CD7  ENDLOOP
CD0  CD7
CD0  CD7  ERREXIT :ERROR3:
CD0  CD7  ENDS;
CD0  CD7  IF COMMA IS NEXT TOKEN, THEN
CD0  CD7
CD0  CD7  ERREXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERROR4:
CD0  CD7  INCREMENT TO NEXT TOKEN
CD0  CD7  ERREXIT IF NEXT TOKEN IS NOT A NAME :ERROR1:
CD0  CD7  RETAIN THIS NAME AS INTERFACE TABLE NAME
CD0  CD7  INCREMENT TO NEXT TOKEN
CD0  CD7
CD0  CD7  ELSE
CD0  CD7
CD0  CD7  SET INTERFACE TABLE NAME TO ZERO
CD0  CD7
ENDIF
323 3 EXIT IF NEXT TOKEN IS NOT EOS :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 ERROR :ERROR5:
330 5 ENDIF
331 4 ENDIF
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 ENDIF
344 5 ENDIF
345 4 ENDIF
346 4 SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SEQWO) INPUT/PROMPTED
347 3 ENDIF
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 ENDIF
352 2 EXIT XSENT
353 3 :ERROR: CALL XRMGR - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
354 3 :ERROR: CALL XRMGR - '.... IS NOT A VALID PROCESSOR NAME'
355 3 :ERROR4: CALL XRMGR - '... DOES NOT USE AN INTERFACE TABLE'
356 3 :ERROR5: CALL XRMGR - 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
357 2 END XSENT
359   2  CD0      FORTRAN CALLING PROCEDURE
360   2  CD0
361   2  CD0       CALL XSPCK
362   2  CD0
363   2  CD0       ********
364   2  CD0
365   2  CD1       XSPCK COMPACTS THE WORKING BUFFER BY REMOVING ALL SEQUENCE
366   2  CD1       TABLE ENTRIES MARKED FOR DELETION
367   2  CD1
368   2  CD1       ********
369   2  CD2
370   2  CD2       INPUT
371   2  CD2
372   2  CD2       COMMON XE - LU
373   2  CD2
374   2  CD2       COMMON XB - DEBUG, NUMENT, TABNOX, WBUF
375   2  CD2
376   2  CD2       ********
377   2  CD3
378   2  CD3       OUTPUT
379   2  CD3
380   2  CD3       COMMON XB - NUMENT, TABNOX, WBUF
381   2  CD3       ********
382   2  CD5
383   2  CD5       NOTES
384   2  CD5
385   2  CD5       USES ROUTINES
386   2  CD5
387   2  CD5       XRMOV
388   2  CD5
389   2  CD5       XRMSG
390   2  CD5
391   2  CD5       XUCR
392   2  CD5
393   2  CD5       ********
BEGIN XSPCK
IF THE TABLE IS NOT EMPTY, THEN
DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
IF THIS ENTRY IS MARKED DELETED, THEN
SET MOVE LENGTH (MOVLEN) TO 7
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT MOVLEN BY 7
ENDDO
MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
THE DELETED ENTRY
DECREMENT NUMENT BY MOVLEN/7
IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
ENDIF
ENDIF
ENDDO
ENDIF
END XSPCK
BEGIN XSLST
   IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
      DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
   SET LIMITS (BEGIN AND ENDNO) OF SEQ. ENTRIES LISTED
      SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
ENDIF
WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
   DO FROM BEGIN TO ENDNO
      MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
      IF INT. TABLE NAME = 0, THEN
         SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARs.)
      ELSE
         SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARs.)
      ENDIF
      CALL XRIS TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
      WRITE PRINT BUFFER
   ENDIF
ENDIF
END XSLST
<table>
<thead>
<tr>
<th>Line</th>
<th>2</th>
<th>CDO</th>
<th>FORTRAN CALLING PROCEDURE</th>
<th>XSCAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>466</td>
<td>2</td>
<td>CDO</td>
<td>CALL XSCAN</td>
<td>XSCAN</td>
</tr>
<tr>
<td>467</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>468</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>469</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>470</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>471</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>472</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>473</td>
<td>2</td>
<td>CDO</td>
<td>XSCAN PERFORMS SYNTACTICAL PROCESSING FOR THE LIST AND</td>
<td>XSCAN</td>
</tr>
<tr>
<td>474</td>
<td>2</td>
<td>CDO</td>
<td>DELETE DIRECTIVES OF THE SEQUENCE TABLE EDITOR</td>
<td>XSCAN</td>
</tr>
<tr>
<td>475</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>476</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>477</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>478</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>479</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>480</td>
<td>2</td>
<td>CDO</td>
<td>COMMON XE - COMBUF, COMPTR, LU, TOKENS</td>
<td>XSCAN</td>
</tr>
<tr>
<td>481</td>
<td>2</td>
<td>CDO</td>
<td>COMMON XE - DEBUG, NUMENT, TABSIZ, WRBUF</td>
<td>XSCAN</td>
</tr>
<tr>
<td>482</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>483</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>484</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>485</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>486</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>487</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>488</td>
<td>2</td>
<td>CDO</td>
<td>COMMON XB - BEGNO, ENDMNO, IRETC</td>
<td>XSCAN</td>
</tr>
<tr>
<td>489</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>490</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>491</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>492</td>
<td>2</td>
<td>CDO</td>
<td>Notes</td>
<td>XSCAN</td>
</tr>
<tr>
<td>493</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>494</td>
<td>2</td>
<td>CDO</td>
<td>USES ROUTINES</td>
<td>XSCAN</td>
</tr>
<tr>
<td>495</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>496</td>
<td>2</td>
<td>CDO</td>
<td>XRMSEG</td>
<td>XSCAN</td>
</tr>
<tr>
<td>497</td>
<td>2</td>
<td>CDO</td>
<td>XUDBG</td>
<td>XSCAN</td>
</tr>
<tr>
<td>498</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
<tr>
<td>499</td>
<td>2</td>
<td>CDO</td>
<td></td>
<td>XSCAN</td>
</tr>
</tbody>
</table>
BEGIN XSCAN
SET LIST LIMITS (BEGNO AND ENMNO) 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 :ERR06:
SET BEGIN LIMIT (BEGNO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
IF NEXT TOKEN IS A COMMA, THEN
INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
ERREXIT IF VALUE IS < 1 :ERR06:
SET END LIMIT (ENMNO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
ENDIF
IF BEGIN LIMIT (BEGNO) = 0, THEN
SET BEGIN LIMIT (BEGNO) TO 1 (BEGNO IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXITIF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGNO)
SET BEGIN LIMIT (BEGNO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERR06:
END SEARCH
ENDIF
IF ENDM LIMIT (ENMNO) = 0, THEN
SET END LIMIT (ENMNO) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGNO) TO LAST TABLE ENTRY
EXITIF SEQ. NO. OF THIS ENTRY = END LIMIT (ENMNO)
SET END LIMIT (ENMNO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERR06:
END SEARCH
ENDIF
SET RETURN CODE TO INDICATE NO ERROR
SET RETURN CODE TO INDICATE AN ERROR
EXIT XSLIE
:ERR01: CALL XRMSG - "SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD"
:ERR06: CALL XRMSG - "INVALID SEQUENCE NUMBER"
:ERR08: CALL XRMSG - "INVALID SEQUENCE NUMBER RANGE"
END XSCAN
FORTRAN CALLING PROCEDURE
CALL XSPMT

XSPMT PROCESSES THE SEQUENCE TABLE EDITOR PROMPT DIRECTIVE

INPUT
COMMON AE - CONBUF, COMINF, LUN, TOKENS
COMMON XB - DEBUG

OUTPUT
COMMON XB - PRMTND, TABNOX

NOTES
USES ROUTINES
XRESD, XUDSG

BEGIN XSPMT
ERREXIT IF TOKEN IS NOT COMMA :ERROR1:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT A NAME :ERROR1:
INCREMENT TO NEXT TOKEN
ERREXIT IF TOKEN IS NOT EOS :ERROR1:
IF NAME IS 'M', THEN
SET PROMPT MODE TO CREATE
ELSE
ERREXIT IF NAME IS NOT 'A' :ERROR9:
SET PROMPT MODE TO ALL
SET TABLE ENTRY INDEX (TABNOX) TO 0
ENDIF
EXIT XSPMT
:ERROR1: CALL XRESD - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERROR9: CALL XRESD - 'SYNTAX ERROR - INVALID QUALIFIER'

END XSPMT
FORTAN CALLING PROCEDURE

CALL XSODEL

XSODEL IS THE SEQUENCE TABLE EDITOR DELETE DIRECTIVE PROCESSOR

INPUT

COMMON XE - LU

COMMON XB - BEGNO, DEBUG, ENDNO, IRETC, NUMENT

OUTPUT

COMMON XB - NUMENT, WKBUF

NOTES

USES ROUTINES

XSCAN

XDDG

BEGIN XSODEL

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

IF NO ERROR INDICATED, THEN

DO FROM THE BEGIN LIMIT (BEGIN) TO THE END LIMIT (ENDNO)

MARK THIS SEQ. TABLE ENTRY AS DELETED

ENDDO

DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR

UNTIL A NON-DELETED ENTRY FOUND

IF ENTRY IS MARKED DELETED, THEN

DECREMENT NUMBER OF TABLE ENTRIES (NUMENT) BY ONE

ENDIF

ENDDO

ENDIF

END XSODEL
BEGIN XSLIS
   CALL XSCAM TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE
   IF NO ERROR INDICATED, THEN
      CALL XSLST TO LIST TABLE ENTRIES DEFINED BY LIMITS
END IF
ENDO XSLIS
FORTRAN CALLING PROCEDURE

CALL XSLTS

*********

XSLTS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR

*********

INPUT

COMMON XE - LU

COMMON XB - DEBUG, ETC

*********

NOTES

USES ROUTINES

XICAN

XSLST

XUBDD

*********

FORTRAN CALLING PROCEDURE

CALL X$NUM

*********

X$NUM IS THE SEQUENCE TABLE EDITOR NUMBER DIRECTIVE PROCESSOR

*********

INPUT

COMMON XE - COMBUF, COMPTA, LU, TOKENS

COMMON XB - DEBUG, NUMET

*********

OUTPUT

COMMON XB - WKBUF

*********

NOTES

USES ROUTINES

X$MSG

XUBDD
BEGIN X$NUM
ERREXIT IF TOKEN IS NOT EOS :ERROR1:
IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
  SET SEQUENCE NUMBER (SEQNO) TO 100
  DO FOR ALL ENTRIES IN TABLE
    IF TABLE ENTRY IS NOT MARKED DELETED, THEN
      SET SEQ. NO. FIELD OF ENTRY TO SEQUENCE NUMBER (SEQNO)
      INCREMENT SEQUENCE NUMBER (SEQNO) BY 100
    ENDIF
  ENDDO
EXIT X$NUM

:ERROR1: CALL X$MSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
END X$NUM
FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:

CALL XICOM (PROMPT, PRMLEN, RETCOD)

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

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

OUTPUTS IN CALLING SEQUENCE:
RETCOD - (INTEGER, 1 WORD) IS A COMPLETION CODE DEFINED
AS FOLLOWS:
0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE
1 - USER RESPONDED I. 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
4 - USER REQUESTED A CONTINUATION

INTERNAL VARIABLES
COMMlSG - 6 WORD ARRAY CONTAINING THE CONTINUATION
MESSAGE
PREFIX - 4 CHARACTERS USED AS PREFIXES TO PROMPT
CORRESPONDING TO CODES IN XE(5)
RETCFP - RETURN CODE FROM XTLM EXTENDED PROMPTING
RETCUAL - RETURN CODE FROM XTLM LEXICAL ANALYSIS
SUFFIX - _ CHARACTERS APPENDED TO END OF PROMPT

XE COMMON USED:

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

(XE(5), LLEVEL), (XE(114), LDRIV),
(XE(143), NOTOK), (XE(146), MONDS)

XS COMMON USED:
EQUIVALENCE (XS(1), INBUF), (XS(81), NOIN),
+ (XS(82), OUTBUF), (XS(122), SSFLAG),
+ (XS(123), INLEN)
INBUF - 80 WORD ARRAY THAT CONTAINS THE USER'S RESPONSE
IN R1 FORMAT
INLEN - LENGTH OF ORIGINAL PROMPT PASSED TO XTPRM
NOIN - NUMBER OF WORDS OF INBUF ACTUALLY USED FOR THE
USER'S RESPONSE
OUTBUF - 40 WORD AREA WHERE PROMPT IS CONSTRUCTED AND
WRITTEN FROM.
SSFLAG - SYMBOLIC STRING INDICATOR FLAG
0 = NOT WITHIN A SYMBOLIC STRING
NOWZERO = VALUE REPRESENTING THE INDEX INTO COMBUF
WHERE LENGTH OF SS IS TO BE STORED
SUBROUTINES AND FUNCTIONS CALLED:
EXEC, KCVT, XRMOV, XRMSG, XRUPK, XTLAM, XTPRM
PDL ROUTINES INCLUDED:
XTCOM, READSEG
FORTAN CALLING PROCEDURE FOR LEXICAL ANALYSIS:

CALL XTLAM (RETC)

OUTPUTS IN CALLING SEQUENCE:

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

       0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE.
       1 - USER RESPONDED Y. 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 REM.
FLGCOM - COMMA FLAG
LAST - LAST CHARACTER WAS NOT A COMMA
       1 - LAST CHARACTER WAS A COMMA
FLGEN - END LOOP FLAG
       0 - CONTINUE LOOP
       1 - END LOOP
FLGNEG - NEGATIVE EXPONENT FLAG
       0 - EXPONENT WAS POSITIVE
       1 - EXPONENT WAS NEGATIVE
FLGTYPE - TYPE OF REAL VALUE
       0 - SIMPLE PRECISION
       1 - DOUBLE PRECISION
POWEX - EXPONENT PART OF A REAL NUMBER
RELWD - SINGLE PRECISION LOCATION FOR REAL VALUE
SPCHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING
       THE % FORMAT REPRESENTATION FOR:
       "'-.0E0=8(0) X23.YXR; DEW
       X IS A CLOSED BRACKET
       Y IS AN OPEN BRACKET
       2 IS A BACK SLASH
206  1 CD4
207  1 CD**********
208  1 CD**********
209  1 CD5
210  1 CD5
211  1 CD5
212  1 CD5
213  1 CD5
214  1 CD5
215  1 CD5
216  1 CD5
EQUIVALENCE (X0(113), INBF), (X0(81), NOCCHAR),

+ (X0(122), FLAG55), (X0(186), SCRATCH)
SUBROUTINES AND FUNCTIONS CALLED
XRCK, XRMV
PDL ROUTINES INCLUDED:
XITKN, COMMA, ALPHA, DIGIT, DCOL, DECPT, EORD
INTEG, REAL, DBL, REPEI, INVNL, CHARS, QUOTE

REPORTING STOP
BEGIN XILAN
SET END FLAG OFF
SET LAST CHARACTER WAS A COMMA ON
REINITIALIZE INDEX INTO INPUT BUFFER
REINITIALIZE RETURN CODE TO NORMAL RETURN
DO WHILE END FLAG IS OFF
  IF INPUT BUFFER HAS BEEN COMPLETELY SCANNED THEN
    IF LAST CHARACTER WAS A COMMA OR A SYMBOLIC STRING IS STILL OPEN THEN
      SET RETURN CODE TO SAY CONTINUATION REQUESTED
    ENDIF
  SET END FLAG ON
ELSE
  ERREXIT IF COMBUF IS FULL PERFORM COMFUL
  IF INPUT CHARACTER IS A COMMA THEN
    PERFORM COMPA
  ELSE
    SET LAST CHARACTER WAS A COMMA OFF
    IF INPUT CHARACTER IS AN ALPHABETIC CHARACTER THEN
      PERFORM ALPHA
    ELSE
      IF INPUT CHARACTER IS A DIGIT THEN
        PERFORM DIGIT
      ELSE
        PERFORM SCHARS
      ENDIF
    ENDIF
  ENDIF
ENDIF
ENDO
STORE END OF BUFFER TOKEN IN COMBUF
INCREMENT #TOKEMS BY 1
END XILAN

BEGIN COMM
SET LAST CHARACTER WAS A COMMA ON
ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
STORE COMMA TOKEN IN COMBUF
INCREMENT #WORDS IN COMBUF BY 1
INCREMENT #TOKFMS BY 1
GET NEXT INPUT CHARACTER
END COMM
1 BEGIN DECL
2   SET INTEGER = 0
3 SET COUNTER = 0
4 DO WHILE CHARACTER IS A DIGIT AND
5 INPUT BUFFER IS NOT EXHAUSTED
6 SET INTEGER = (INTEGER + 10) + CURRENT CHARACTER - 48
7 EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
8 INCREMENT COUNTER BY 1
9 GET NEXT CHARACTER
10 END DO
11 BEGIN DECP
12 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
13 GET NEXT INPUT CHARACTER
14 IF INPUT BUFFER IS NOT EXHAUSTED THEN
15 IF INPUT CHARACTER IS A DIGIT THEN
16 PERFORM DCOL
17 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
18 EXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
19 ENDIF
20 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
21 PERFORM FORD
22 ELSE
23 PERFORM REAL
24 ENDIF
25 ELSE
26 PERFORM REAL
27 ENDIF
28 END DECP
349  1 BEGIN EORD
350    2  IF INPUT CHARACTER IS AN "E" THEN
351    3      SET TYPE FLAG TO "E"
352    4  ELSE
353    5      SET TYPE FLAG TO "D"
354    6  ENDIF
355    7  IF CHARACTER IS "E" OR "D" THEN
356    8      GET NEXT CHARACTER
357    9      IF CHARACTER IS A - THEN
360   10        SET NEGATIVE FLAG OFF
361   11        IF CHARACTER IS A - THEN
362   12        GET NEXT CHARACTER
363   13        ENDIF
364   14        ELSE
365   15          IF CHARACTER IS A - THEN
366          16            GET NEXT CHARACTER
367          17            ENDIF
368   18          ENDIF
369   19        IF CHARACTER IS NOT A DIGIT THEN
370        20          PERFORM BCOL
371        21          IF NEGATIVE FLAG IS ON THEN
372        22          SET POWER = -POWER
373        23          ENDIF
374        24          PERFORM REAL
375        25          ELSE
376        26          PERFORM DBL
377        27          ENDIF
378  1 END EORD
1 BEGIN INTEGER
  2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  3 CONVERT NUMBER TO INTEGER
  4 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM COMFUL
  5 STORE INTEGER TOKEN IN COMBUF
  6 INCREMENT WORDS IN COMBUF BY 2
  7 INCREMENT TOKENS BY 1
  8 END INTEGER

1 BEGIN REAL
  2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  3 SET REAL = DOUBLE PRECISION * 10 ** POWER
  4 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
  5 STORE REAL TOKEN IN COMBUF
  6 INCREMENT WORDS IN COMBUF BY 3
  7 INCREMENT TOKENS BY 1
  8 END REAL

1 BEGIN DBL
  2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  3 SET DOUBLE = DOUBLE PRECISION * 10 ** POWER
  4 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
  5 STORE DOUBLE TOKEN IN COMBUF
  6 INCREMENT WORDS IN COMBUF BY 4
  7 INCREMENT TOKENS BY 1
  8 END DBL

1 BEGIN REPET
  2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
  3 ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVALID
  4 STORE REPEAT TOKEN IN COMBUF
  5 INCREMENT WORDS IN COMBUF BY 2
  6 INCREMENT TOKENS BY 1
  7 GET NEXT CHARACTER
  8 END REPET
BEGIN SCHEARS

CHARACTER TABLE:

"-" = UNDEFINED CHARACTER
"*" = STRING END
"#" = STRING DELETE
"(" = STRING NAME
"")" = STRING NAME...
X IS A CLOSED BRACKET
Y IS AN OPEN BRACKET
Z IS A BACK SLASH

SET J=1
START SEARCH WHILE J<=# CHARACTERS IN TABLE
EXIT IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
INCREMENT J BY 1
END LOOP
PERFORM INVAL - NO RETURN
END SEARCH
SET NEXT_FLAG ON
CASE J:(A:;:F:;F:;F:;F:;F:;F:;F:;F:;F:;F:;F:;F:;F:;B:;C:;C:;D:;E:;F:;F:;F:;INVAL:;INVAL:;F:)

:A:
SET NEXT_FLAG OFF
PERFORM QUOTE

:B:
SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED
SET END_FLAG ON

:C:
IF SYMBOLIC STRING FLAG IS OFF THEN
SET SYMBOLIC STRING FLAG TO CURRENT COMBUF INDEX + 1
ELSE
SET COMBUF(SYMBOLIC STRING FLAG) = CURRENT COMBUF INDEX - SYMBOLIC STRING FLAG
SET J = J+1 TO STORE SYMBOLIC STRING CLOSE TOKEN
SET SYMBOLIC STRING FLAG OFF
ENDIF

:D:
SET RETURN CODE TO SAY X ENTERED
SET END_FLAG ON

:E:
IF FOLLOWING CHARACTER IS A DIGIT THEN
SET NEXT_FLAG OFF
SET INTEGER = 0
SET POWER = 0
PERFORM DECPT
ENDIF

:F:
ENDCASE

IF NEXT_FLAG IS ON
STORE TOKEN (J) IN COMBUF
INCREMENT WORDS IN COMBUF BY 1
IF TOKEN IS BEGIN SYMBOLIC STRING THEN
INCREMENT WORDS IN COMBUF BY 1
ENDIF
INCREMENT #TOKENS BY 1
GET NEXT CHARACTER
477 1 BEGIN QUOTE
478 2 GET NEXT CHARACTER
479 3 SET #CHARACTERS = 0
480 4 DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
481 5 INPUT BUFFER HAS NOT BEEN COMpletely SCANNED) OR
482 6 INPUT CHARACTER IS A QUOTE AND
483 7 NEXT CHARACTER IS A QUOTE AND
484 8 INPUT BUFFER HAS NOT BEEN COMpletely SCANNED)
485 9 INCREMENT #CHARACTERS BY 1
486 10 MOVE CHARACTER INTO TEMPORARY BUFFER
487 11 IF INPUT CHARACTER IS A QUOTE THEN
488 12 GET NEXT CHARACTER
489 13 ENDIF
490 14 GET NEXT CHARACTER
491 15 ENDDO
492 16 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
493 17 ERREXIT IF LENGTH OF CHARACTER STRING IS 0 OR
494 18 ERREXIT IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
495 19 STORE CHARACTER STRING TOKEN IN COMBUF
496 20 CALL XRPOCK TO PACK CHARACTERS INTO COMBUF
497 21 INCREMENT #WORDS IN COMBUF BY 2+(#CHARACTERS+1)/2
498 22 INCREMENT #TOKENS BY 1
500 2 GET NEXT CHARACTER
501 1 END QUOTE
FORTRAN CALLING PROCEDURE

CALL XTPRM

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

INPUT

XE COMMON - CARTRG, COMBUF, FLAGS, LU, NOPROC, PRKAM, SUBSTA,
TOKENS

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

XS COMMON - PRMLEM (FROM XTCOM)

VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)

OUTPUT

XE COMMON - COMBUF (USED FOR SCRATCH SPACE)

LISTING OF APPROPRIATE EXTENDED PROMPTS

INTERNAL VARIABLES

CONTINU - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY

FILE - FILE NAME OF CURRENT TABLE ENTRY

INDEX - INDEX TO CURRENT TABLE ENTRY

L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO

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

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

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

SECTION 6.2.4.3)

RECORD - RECORD NUMBER WITH WHICH TO BEGIN PROCESSING FOR CURRENT

TABLE ENTRY

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

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

LAST CHARACTER TO BE MASKED)

TABLE - PROCESSING CONTROL TABLE FOR VARIOUS SYNTAX CONDITIONS

INDEX SYNTAX CONDITION

FILE INDEX SIZE SEARCH MESSAGE CONTINUE

INDEX 1 I X?: 1 I I XDPRM 1 2 NO NONE NO I

INDEX 2 I X?: 1 I 9 I XDPRM 1 2 YES XTO6 NO I

INDEX 3 I #?: 1 I 17 I XSPRM 1 2 NO NONE YES I

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

INDEX 5 I #?: 1 I 33 I XSPRM 1 2 YES NONE YES I

INDEX 6 I $?:, $P?:, $P?:, $P?: 1 I 41 I XPROC 3 128 NO XTO7 NO I

INDEX 7 I /?: 1 I 49 I XSPRM 1 2 NO NONE YES I

INDEX 8 I /?: 1 I 57 I XPROC 1 3 NO NONE NO I

INDEX 9 I /?: 1 I 65 I XSPRM 1 2 YES NONE YES I

INDEX 10 I /?:, $P?:, $P?: 1 I 73 I XPROC 1 3 YES XTO8 NO I

WHERE P AND R INDICATE A PROMPT AND RESPONSE

NOTES
1 BEGIN XPWR
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          ENSDIF
13     ELSE
14       SET SIZE POSITIVE
15       DO FOR EACH LIST ITEM (1-M)
16       BLANK LAST CHARACTER
17     ENSD
19     ELSE
20       IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), THEN
21         START SEARCH WHILE LIST ITEMS REMAIN TO BE EXAMINED
22         EXIT IF TOKEN LOCATED IN LIST
23         POSITION TO APPROPRIATE RECORD (I+L-1) AND READ
24         EXIT TO :ERROR9: IF FAILURE
25         SET SIZE TO 128 (ENTIRE RECORD TO BE DISPLAYED)
26         IF TABLE MESSAGE NUMBER FIELD > 0, THEN
27           CALL XRMSG TO DISPLAY 'NOT VALID ...' MESSAGE
28           ENSDIF
29         END LOOP
30         EXIT TO ENDDO
31         ENSD SEARCH
32         ENSDIF
33         PERFORM DISPLAY
34         ELSE OPEN ERROR
35         IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
36           CALL XRMSG TO DISPLAY 'NOT VALID ...' MESSAGE
37         ELSE
38           :ERROR9: CALL XRMSG TO DISPLAY 'FILE MANAGER ERROR ...' MESSAGE
39           ENSDIF
40           CLOSE FILE
41         ENSDIF
42         ENDDO
43 1 END XPWR
BEGIN SETUP
EXIT TO :ERROR2: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
CASE (:X:, :S:, :F:, :/) SUBSTA

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

IF FIRST TOKEN = ?, THEN
SET TABLE INDEX TO FOURTH ENTRY
ELSE
SET TABLE INDEX TO SIXTH ENTRY
FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
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)
FORM SIXTH ENTRY FILE NAME FROM > AND TOKEN
ENDIF

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
END CASE
EXIT SETUP

:ERROR2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
END SETUP
FORTRAN CALLING PROCEDURE

CALL XUDG (I,U, ID)

CD0
CD1 XUDG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY
CD1 DUMP AND/OR MODIFICATION FACILITY
CD1
CD2 INPUT (CALLING SEQUENCE)
CD3 L - LOGICAL UNIT TO INTERACT WITH IN INVOKING XUDG OPTIONS
CD3 ID - THREE WORD ASCII ARRAY USED AS A HEADER TO IDENTIFY XUDG CALLER
CD3
CD2 INPUT (INTERACTIVE)
CD3 OPTION (S,M,E): - ONE ASCII CHARACTER IDENTIFYING XUDG OPTION
CD3 S = SNAP OUT (DUMP) MEMORY
CD3 M = MODIFY MEMORY
CD3 E = EXIT XUDG
CD3 START: - OCTAL MEMORY ADDRESS IN THE USERS MAP OF FIRST
CD3 STOP: - OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPPED
CD3 OUTPUT UNITS: - LOGICAL UNIT NUMBER TO WHICH SNAP IS TO BE
CD3 VALUES: - ARRAY OF OCTAL VALUES TO BE STORED IN MEMORY
CD3 BEGINNING AT LOCATION "START", WHEN INPUT
CD3 "VALUES" MUST NOT EXCEED 50 CHARACTERS. NULL
CD3 FIELDS, I.E., SUCCESSIVE COMMAS, INDICATE WORDS
CD3 OF ZERO TO BE STORED.

CD3 OUTPUT (TO 'LU')
CD3 HEADER - "*** XUDG FROM IDIDID"
CD3 PROMPTS - (SEE INPUT)
CD3
CD3 OUTPUT (TO 'OUTPUT UNIT')
CD3 HEADER - "*** XUDG FROM IDIDID"
CD3 SNAP - YOZ WORD DUMP FORMATTED LINE (SEE XUDPL)
CD3
CD3
CD4
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 LUI - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET
CD4 LUO - LOGICAL UNIT FOR SNAP OUTPUT
CD4 OPTN - 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 RTH - RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND
CD4 CONVERSION'

CD5 NOTES

CD5
1 BEGIN XUDG
2 WRITE SNAP HEADER TO TERMINAL
3 DO UNTIL OPTION IS EXIT (E)
4 PROMPT TERMINAL FOR OPTION
5 IF OPTION IS NOT EXIT (E)
6 THEN
7 PERFORM PROMPT AND CONVERSION FOR STARTING ADDRESS
8 IF OPTION IS MODIFY (%)
9 THEN
10 PERFORM PROMPT AND CONVERSION FOR VALUES AND STORE IN MEMORY
11 ELSE
12 PERFORM PROMPT AND CONVERSION FOR ENDING ADDRESS
13 PERFORM PROMPT AND CONVERSION FOR OUTPUT UNIT
14 WRITE SNAP HEADER TO OUTPUT UNIT
15 DO FOR EACH EIGHT WORD BLOCK OF DUMP AREA
16 CALL XUDPL TO FORMAT LINE
17 OUTPUT LINE
18 END DO
19 ENDIF
20 END DO
21 END XUDG
22 BEGIN PROMPT AND CONVERSION
23 ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
24 RETRIEVE RESPONSE
25 CLEAR SUM
26 DO FOR EACH CHARACTER
27 SHIFT SUM AND ADD NEXT DIGIT
28 END DO
29 END PROMPT AND CONVERSION
GENERAL FILE DUMP PROGRAM FOR FILE MANAGER FILES

NAME - NAME OF FM FILE TO BE DUMPED
IREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED
(NEXT RECORD IS RECORD NUMBER ONE)
NREC - NUMBER OF LOGICAL RECORDS TO DUMP
FMT - RUN TIME FORMAT FOR RECORDS (MAXIMUM OF 72 CHARACTERS) OR
BLANK INDICATING THE DEFAULT OF OCTAL AND ASCII DUMP TYPE
FORATTING 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, MAXQ, OPEN, POINT, READF, RMPAR, ZPRDS, XPRDV, XURPL

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

******

*  *
BEGIN XUDDPF
*  *
DO FOREVER
  READ FILE NAME
  EXIT XUDDPF 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
  ENDDO
END XUDDPF
FORTRAN CALLING PROCEDURE

CALL XUDPL (ADDR, LINE, BUFFER)

*********

PRODUCE AN OCTAL AND ASCII PRINT FORMATTED MEMORY DUMP LINE IMAGE

*********

INPUT

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 FORMATTED AND SPACED INTO A LINE IMAGE

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.

*********

OUTPUT

BUFFER - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE

COLUMNS CONTENTS

FIRST ADDRESS

SECOND ADDRESS

21-32 OCTAL REPRESENTATION OF "LINE"

87-102 ASCII REPRESENTATION OF "LINE"

*********

NOTES

USES XREXT, XR06, XRSET

*********

BEGIN XUDPL

CALL XR06 TO CONVERT EACH WORD OF ADDR TO OCTAL

DO FOR EACH WORD OF LINE

CALL XR06 TO CONVERT WORD TO OCTAL

DO FOR EACH BYTE OF WORD

IF BYTE < 40 OR BYTE > 136 THEN

REPLACE BYTE WITH ASCII PERIOD

ENDIF

END DB

END XUDPL
1 CD1 FORTRAN MAIN PROGRAM XUFMT IS SCHEDULED BY XDUMP TO PRINT
2 CD1 A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
3 CD1
4 C**** INPUT
5 CD2 ICNTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
6 CD2 LU NUMBER OF WHERE DUMP IS ON DISK
7 CD2 TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
8 CD2 CONTAINING THE DUMP
9 CD2
10 C**** OUTPUT
11 CD3 FORMATTED DUMP TO LU 6
12 CD2 EXTERNAL REFERENCES
13 CD5 EXEC
14 CD5 XRMPAR
15 CD5 XRCPAR
16 CD5 ZRMOV
17 CD5 XUDPL
18 BEGIN XUFMT
19 CALL XRMPAR TO GET LU AND STARTING TRACK NOS.
20 READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
21 COUNTOFID-SEGS TO BE DUMPED
22 UP TO 7 ID-SEGMENT ADDRESSES
23 LOW AND HIGH BASE PAGE ADDRESSES
24 LOW AND HIGH MAIN MEMORY ADDRESSES
25 POINT TO 2ND SECTOR OF DUMP DATA
26 DO UNTIL ALL ID-SEGS PRINTED
27 DO UNTIL A EIGHT-WORD LINES PRINTED
28 PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
29 ENDDO
30 INCREMENT TO NEXT SECTOR OF DUMP DATA
31 ENDDO
32 COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP
33 PERFORM COMPARE AND PRINT FUNCTION
34 COMPUTE NTRKS., NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ
35 DO UNTIL NTRKS ARE READ
36 READ NEXT TRACK FROM DISK
37 COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP
38 PERFORM COMPARE AND PRINT FUNCTION
39 ENDDO
40 RELEASE THE GLOBALLY ALLOCATED TRACKS
41 EXIT XUFMT
42 BEGIN COMPARE AND PRINT FUNCTION
43 DO UNTIL N LINES PROCESSED
44 IF NOT 1ST LINE, THEN
45 CALL XRCPAR TO COMPARE WITH PREVIOUS LINE
46 IF LINES ARE IDENTICAL, THEN
47 IF THIS IS 1ST OF A SERIES, THEN
48 WRITE 'DUPLICATE LINE'
49 ENDIF
50 ELSE
51 CALL XUDPL TO FORMAT THE DUMP LINE
52 WRITE FORMATTED DUMP LINE
53 ENDF
54 ELSE
55 CALL XUDPL TO FORMAT THE DUMP LINE
56 WRITE FORMATTED DUMP LINE
57 ENDIF
58 END
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1   ENTRY XVABN
2   TYPE 1A ROUTINE TO ABORT CURRENT ID AND
3   REQUEST A DUMP FOR THE ID.
4   ENTRY XVABN
5   TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
6   THE REQUESTED DATA AREAS AND THE CURRENT
7   ID SEGMENT AND ITS FATHER CHAIN SEGMENTS
8   BEGIN XVABN ABEND
9   02 INPUTS: NONE
10   03 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
11   * EXTERNALS: $ABRT, $ABRE, $XEN, $LIBR
12   02 CALL SLIBR: REENTRANT ROUTINE CALL
13   03 GET CURRENT ID FROM XERT (LOCATION 1717)
14   03 INCREMENT TO ADDRESS BOUNDARIES
15   02 GET ADDRESS BOUNDARIES
16   02 STORE $XUDMP PARAMETER LIST
17   02 PERFORM XUDMP DUMP THIS PARTITION
18   02 CALL $ABRT ABORT THIS PARTITION
19   02 CALL $ABRE FREE CURRENT REENTRANT DATA BLOCK
20   EXIT TO $XEN GO TO DISPATCHER
21   END XVABN
22   * XVABN
23   * XVABN
24   * XVABN
25   * XVABN
26   * XVABN
27   * XVABN
28   * XVABN
29   BEGIN XUDMP
30   ENTRY XUDMP
31   02 INPUTS: ADDRESS OF DOUBLE WORD CONTAINING START AND END ADDRESSES
32   03 DEFINING AREA TO BE DUMPED
33   03 OUTPUTS: OUTPUTS TO DISK
34   03 HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW
35   03 CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS
36   03 MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION
37   03 MEMORY IMAGE AS SPECIFIED BY INPUT ADDRESS RANGE
38   03 SCHEDULES XUFMT TO FORMAT DATA FROM DISK TO PRINT
39   03 EXTERNALS: $LIBR, $LIBX, EXEC, XUFMT
40   02 CALL SLIBR TO NOTIFY OF REENTRANCY AND BECOME PRIVILEGED
41   02 RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH
42   02 CALL SLIBX TO BECOME NON-PRIVILEGED
43   02 CALL EXEC FOR A GLOBAL ALLOCATION OF DISK
44   02 CALL EXEC TO WRITE HEADER DATA USING CLASS 1/0
45   02 DO UNTIL ALL ID SEGMENTS PROCESSED
46   02 WRITE ID SEGMENT TO DISK
47   02 ENDO
48   02 WRITE BASE PAGE SEGMENT TO REMAINING PROTON OF THIS TRACK
49   02 DO UNTIL END ADDRESS REACHED
50   02 WRITE ONE SECTOR OF DATA
51   02 ENDO
52   02 SCHEDULE XUFMT WITH LU AND STARTING TRACK NOS.
53   02 CALL SLIBX TO RETURN FROM REENTRANT ROUTINE
54   ENDO XUDMP
55   ENDO XVABN
56   ENDO XVABN
08 TYPE 1A ROUTINE TO CONTROL COMMUNICATION BETWEEN AND
09 EXECUTION OF FDS MANAGER AND IT'S ASSOCIATED TASKS
10 (EXECUTIVE, PROCESSORS, AND UTILITIES).
11 ENTRY XVPM AND XVSTB
12 INPUTS
13 FROM AN ASSOCIATED TASK
14 CALL XVPM(PARMS)
15 ASSEMBLY FORM
16 JSB XVPM
17 DEF #2 RETURN ADDRESS
18 DEF PARMS A(PARMS)
19 WHERE PARMS ARE P1,P2,P3,P4,P5
20 P1 IS THE SERVICE REQUEST
21 0= NORMAL TERMINATION (P2-P5 NOT USED)
22 1= WORK AREA REQUEST (P2-P5 NOT USED)
23 2= EXECUTE A SEQUENCE TABLE
24 (P2-P4 HAS TABLE NAME)
25 (P5 INDICATES EXECUTION CONTROL IN CLAS/0 BUFFER)
26 3= RESET SEQUENCE POINTERS
27 (P2 HAS SEQUENCE NUMBER)(P3-P5 NOT USED)
28 8= TERMINATE SEQUENCE (P2-P5 NOT USED)
29 9= TERMINATE FDS FUNCTION (P2-P5 NOT USED)
30 32767= ABNORMAL TERMINATION OF ASSOCIATED TASK
31 FROM AN FDS MANAGER
32 ASSEMBLY FORM
33 JSB XVPM
34 DEF (RETURN POINT)
35 OCT 0
36 DEF PARMS A(FDS MANAGER RESPONSE)
37 OUTPUTS
38 TO AN FDS MANAGER
39 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
40 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
41 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
42 MANAGER IS ACTIVATED VIA SLIST
43 TO AN ASSOCIATED TASK
44 RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
45 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
46 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
47 ASSOCIATE TASK IS ACTIVATED VIA SLIST
103 1 BEGIN XVPAW
104 2 CALL BLIBM BECOME PRIVILEGED
105 2 SET STOP-ID FROM XERT(OCT 1717)
106 2 IF THIS IS A MANAGER RESPONSE
107 2 THEN SET UP TO ACTIVATE ASSOCIATED TASK AND SUSPEND MANAGER
108 3 CALL SEQUENCE IS RETURN,0,(PARMS)
109 3 PERFORM XVPAW POST REQUESTOR AND WAIT
110 3 ELSE SET UP TO ACTIVATE MANAGER AND SUSPEND ASSOCIATED TASK
111 3 CALL SEQUENCE IS RETURN,0,(PARMS)
112 3 PERFORM XVPAW 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 BEActivated.
116 2 CALL BLS(T,SCHEDULE,AWAKEN-ID)
117 2 MOVE RETURN ADDRESS TO XSUSP OF STOP-ID.
118 2 CALL BLS(WAIT,STOP-ID)
119 1 EXIT TO :XEN RTE DISPATCHER
120 1 END XVPAW
BEGIN XVPMAW

* DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
* FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
* SET TARGET-ID FROM CURRENT-ID-SEGMENT
* DO WHILE FATHER-ID-NUMBER .NE. 0 OR FATHER IS WAITING
  * COMPUTE FATHER-ID-SEGMENT FROM FATHER-ID-NUMBER IN TARGET-ID
  * PERFORM MGRFND(FATHER-ID-SEGMENT,COUNT)
* EXIT IF CURRENT-ID .GT. 0
* SET TARGET-ID TO FATHER-ID-SEGMENT
* ENDIF
* IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
  * CALL $SYM (12,*XVO3,SEGMENT-NAME) 'XV03,RNAME' REQUESTING PROG
  * PERFORM PUMP
  * EXIT TO SIBM TO ENABLE AND REDISPATCH
* ENDIF
* SET AWAKEN-ID FROM FSD-ENTRY SYMB
* GET REQUEST PARMs MOVE INTO ID-SEGMENT
* SET STBAT FROM CURRENT-ID
FND XVPMAW

1

BEGIN XVPMAW

* DETERMINE IF CALLER IS A VALID FDS MANAGER
* PERFORM MGRFND (CURRENT-ID,COUNT)
* COUNT WILL BE 0 FOR NO MATCH.
* COUNT NOT EQUAL ZERO IMPLIES A MATCH
* AND FDS-ENTRY HAS MATCHING FSDTAB ENTRY ADDRESS
* IF COUNT .LT. 0 THEN CALLER IS NOT A FDS MANAGER
* CALL $SYM (12,*XVO1,SEGMENT-NAME) 'XV01 PROGRAM' REQUESTING PROG.
* PERFORM PUMP
* EXIT TO :SIBR THE DISPATCHER
ENDIF
* FSDTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
* SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
* IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
* CALL $SYM (12,*XVO2,SEGMENT-NAME) 'XV02 PROGRAM' ASSOCIATED PROG.
* PERFORM PUMP
* EXIT TO :SIBR THE DISPATCHER
ENDIF
* AWAKEN-ID-SEGMENT=STBAT
* IF MANAGER HAS REQUEST FOR ABORT, THEN
* CALL SABRT FOR CURRENT AF
* ENDIF
* MOVE FDS MANAGERS INPUT PARMs TO ASSOCIATED TASK ID SEGMENT
* MOVE PARMs TO ID-SEGMENT WORDS 2-6

END XVPMAW
172  1 BEGIN PDUMP
173  2 *  
174  2 *  SET A CALL TO XVABM AT CALLER'S SUSPEND POINT
175  2 *  
176  1 END PDUMP
177  1 *
178  1 *
179  1 *
180  1 *
181  1 *
182  1 BEGIN MGRFND  INPUT IS TARGET-ID
183  2 *  
184  2 *  GET XVSTB TABLE OF ACTIVE FDS MANAGERS
185  2 *  
186  2 *  SET COUNT TO NUMBER IN TABLE NUMBER MANAGERS
187  2 *  
188  2 *  EXIT IF TARGET-ID .EQ. FDS-MANAGER-ID-SEGMN
189  2 *  
189  3 *  SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
190  2 *  
191  2 *  ENDDO
192  2 *  
193  2 *  FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
194  2 *  COUNT = 0 MEANS NO MATCH
195  2 *  
196  1 *
197  1 *
198  1 BEGIN XVSTB
199  2 *  
200  2 N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSTB
201  2 *  
202  2 STBNM EQU N NUMBER-IN-TABLE
203  2 *  
204  2 STBC EQU 0 NUMBER-FIELD
205  2 *  
206  2 STBN  EQU 0 STB RESOURCE NUMBER
207  2 *  
208  2 STBES EQU 0 ENTRY START
209  2 *  
210  2 STBLU EQU 0 LUCALC
211  2 *  
212  2 STBLA EQU 0 LUCALC
213  2 *  
214  2 STBMA EQU 0 LUCALC
215  2 *  
216  2 STBM EQU 0 LUCALC
217  2 *  
218  2 STBEX EQU 0 LUCALC
219  2 *  
220  2 STBEX EQU 0 LUCALC
221  2 *  
222  2 STBEX EQU 0 LUCALC
223  2 *  
224  2 STBEX EQU 0 LUCALC
225  2 *  
226  2 STBEX EQU 0 LUCALC
227  2 *  
228  2 STBEX EQU 0 LUCALC
229  2 *  
230  2 STBEX EQU 0 LUCALC
231  2 *  
232  2 STBEX EQU 0 LUCALC
233  2 *  
234  2 STBEX EQU 0 LUCALC
235  2 *  
236  2 STBEX EQU 0 LUCALC
237  2 *  
238  2 STBEX EQU 0 LUCALC
239  2 *  
240  2 STBEX EQU 0 LUCALC
241  2 *  
242  2 STBEX EQU 0 LUCALC
243  2 *  
244  2 STBEX EQU 0 LUCALC
245  2 *  
246  2 STBEX EQU 0 LUCALC
247  2 *  
248  2 STBEX EQU 0 LUCALC
249  2 *  
250  2 STBEX EQU 0 LUCALC
251  2 *  
252  2 STBEX EQU 0 LUCALC
253  2 *  
254  2 STBEX EQU 0 LUCALC
255  2 *  
256  2 STBEX EQU 0 LUCALC
257  2 *  
258  2 STBEX EQU 0 LUCALC
259  2 *  
260  2 STBEX EQU 0 LUCALC
261  2 *  
262  2 STBEX EQU 0 LUCALC
263  2 *  
264  2 STBEX EQU 0 LUCALC
265  2 *  
266  2 STBEX EQU 0 LUCALC
267  2 *  
268  2 STBEX EQU 0 LUCALC
269  2 *  
270  2 STBEX EQU 0 LUCALC
271  2 *  
272  2 STBEX EQU 0 LUCALC
273  2 *  
274  2 STBEX EQU 0 LUCALC
275  2 *  
276  2 STBEX EQU 0 LUCALC
277  2 *  
278  2 STBEX EQU 0 LUCALC
279  2 *  
280  2 STBEX EQU 0 LUCALC
281  2 *  
282  2 STBEX EQU 0 LUCALC
283  2 *  
284  2 STBEX EQU 0 LUCALC
285  2 *  
286  2 STBEX EQU 0 LUCALC
287  2 *  
288  2 STBEX EQU 0 LUCALC
289  2 *  
290  2 STBEX EQU 0 LUCALC
291  2 *  
292  2 STBEX EQU 0 LUCALC
293  2 *  
294  2 STBEX EQU 0 LUCALC
295  2 *  
296  2 STBEX EQU 0 LUCALC
297  2 *  
298  2 STBEX EQU 0 LUCALC
299  2 *  
300  2 STBEX EQU 0 LUCALC
301  2 *  
302  2 STBEX EQU 0 LUCALC
303  2 *  
304  2 STBEX EQU 0 LUCALC
305  2 *  
306  2 STBEX EQU 0 LUCALC
307  2 *  
308  2 STBEX EQU 0 LUCALC
309  2 *  
310  2 STBEX EQU 0 LUCALC
311  2 *  
312  2 STBEX EQU 0 LUCALC
313  2 *  
314  2 STBEX EQU 0 LUCALC
315  2 *  
316  2 ORG STBES
317  2 *  
318  2 REP N DO FOR N ENTRIES
319  2 *  
320  2 REP STBEL DEFINE STORAGE WORDS FOR AN ENTRY
321  2 *  
322  2 DEF 0 
323  1 END XVSTB
CD************
CD0 FORTRAN CALLING PROCEDURE:
CD0 CALL XXAUT
CD0
CD0************
CD1 XXAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE
CD1
CD1************
CD2 INPUTS IN COMMON:
CD2
CD2 XE(5) MASSTA, XE(10) SERSRA, XE(11) SESEND, XE(12) SEEKTR,
CD2 XE(140) TABEND, XB(1) IPOP, XB(2) LIBD,
CD2 XB(249) SERN0, XB(250) SLEN, XB(251) SETAB
CD2
CD2************
CD3 OUTPUTS IN COMMON:
CD3
CD3 XE(5) MASSTA, XS(1) FLGTAB
CD3
CD3************
CD4 COMMON USED:
CD4
CD4 EQUIVALENCE (XE(5), MASSTA)
CD4 + (XE(10), SERSRA), (XE(11), SESEND)
CD4 + (XE(12), SEEKTR), (XE(140), TABEND),
CD4 + (XB(1), IPOP), (XB(2), LIBD),
CD4 + (XB(235), SERN0), (XB(245), SLEN),
CD4 + (XB(250), SETAB),
CD4 + (XS(13), FLGTAB)
CD4
CD4************
CD5 FDS ROUTINES CALLED:
CD5
CD5 XCPFR, XREX, XRMV, XMSG,
CD5
CD5************
CD6 RTE ROUTINES CALLED:
CD6
CD6 I08
CD6
BEGIN XSAUT

IF ENTRY IS FROM A DIRECTIVE THEN
SET MASTER STATE TO INDICATE REENTRY
DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
EXIT IF PROCESSOR NOT FOUND TO :ERR1;
STUFF INTERFACE TABLE & IT AND VERSION INTO SEQUENCE TABLE ENTRY
ENDIF

CALL XSTO TO STORE REVISED SEQUENCE TABLE IN XWA AS &XERTB
ELSE IF AM REENTERED FROM INT.
CALL XMTX TO SET UP TEMPORARY EN-ENTRY XMTAB
CALL XXEXE TO EXECUTE FROM TEMPORARY ENTRY
IF RESET SEQUENCE NUMBER IS NO: REQUESTED THEN
EXIT XSAUT IF TERMINAL ENTRY WAS JUST EXECUTED
SET STARTING ENTRY TO NEXT ENTRY
ENDIF

DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
CALL XXEXE TO EXECUTE REMAINDER OF TABLE
ENDIF

EXIT XSAUT

:ERR1:
CALL XMSG TO DISPLAY INVALID PROCESSOR NAME
END XSAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELDS (XICNT)

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

INPUTS FROM CALLING SEQUENCE:

XICNT - (INTEGER, 3 WORDS) ARRAY CONTAINING THE NAME "XICNT" USED BY XELDS TO CALL EXEC TO LOAD THE EXECUTION CONTROLLER SEGMENT.

INTERNAL VARIABLES:

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

COMMON USED:

EQUIVALENCE (XICNT, XASSTA)

FBS ROUTINES USED

XETR, XEXT, XRMSE, XIMAN

NOTE: CONTAINS DUMMY CALL TO XEXEC
119 1 BEGIN XICNT
120 2 SET MODE TO XREXT OF BITS 12 AND 13 OF WASITA
121 2 CASE MODE (:MANU:, :SENI:, :AUTT:, :AUTO:)
122 3 :MANU: CALL XIXMAN
123 3 :SENI: CALL XIXSEN
124 3 :AUTT: CALL XIXMSG TO DISPLAY INVALID OPTION
125 3 :AUTO: CALL XIXAUT
126 2 ENDCASE
127 2 SET MASTER STATE TO DIRECTIVE LEVEL
128 2 CALL XERTH TO RETURN TO XEXEC **END RETURN**
129 2 DUMMY CALL XEXEC
130 1 END XICNT
132 1 C0************
134 1 C0          FORTRAN CALLING SEQUENCE:
135 1 C0          CALL XXDEC (RETC)
137 1 C0************
138 1 C0          XXDEC DECODES A RESPONSE OF PROCESSOR NAME (,INT TABLE NAME)
139 1 C0          INTO A SEQUENCE TABLE ENTRY.
141 1 C0          INPUTS IN COMMON:
144 1 C0          XE(85) TOKENS, XE(145) COMBUF, XB(1) NOPROC, XB(2) LIBD
149 1 C0************
150 1 C0          OUTPUTS IN CALLING SEQUENCE:
152 1 C0          RETC - RETURN CODE (O IS NORMAL RETURN)
155 1 C0          OUTPUTS IN COMMON:
157 1 C0          XE(16) PRCNAM, XB(251) SEQTAB
159 1 C0************
160 1 C0          COMMON USED:
162 1 C0 OS     EQUIVALENCE  (XE(16), PRCNAM),
164 1 C0 OS     + (XE(85), TOKEN1), (XE(144),TOKPTR),
165 1 C0 OS     + (XE(145), COMBUF),
166 1 C0 OS     + (XB(1), NOPROC), (XB(2), LIBD ),
167 1 C0 OS     + (XB(145), SEQTAB)
168 1 C0 OS     FDS ROUTINES USED:
170 1 C0 OS     XRCPR, XREXT, XRNOS, XRNMS
172 1 C0 OS     RTE ROUTINES USED:
174 1 C0 OS     IANB
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 3 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
188 2 ENDF
189 2 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
190 2 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
191 2 IF AN INTERFACE TABLE IS REQUIRED BUT NOT SPECIFIED THEN
192 3 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'EINTAB'
193 2 ENDF
194 1 EXIT XXDEC
195 2 :ERR1:
196 2 CALL XRNUG TO DISPLAY ERROR
197 2 SET RETURN CODE TO SAY ERROR
198 1 END XXDEC
**FORTRAN CALLING PROCEDURE:**

**CALL XXDEF**

**INPUTS FROM COMMON:**

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

**INTERNAL VARIABLES:**

- DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE
- DEFNAM - (INTEGER, 3 WORDS) ARRAY WHERE INTERFACE TABLE NAME IS CREATED FROM PROCESSOR NAME

**COMMON USED:**

- EQUIVALENCE (XE(5), MASSTA), (XE(6), SUBSTA)
- (XE(13), INTMAN), (XE(16), PRCHAM), (XE(19), REPTRY), (XE(20), REQBUF)
- (XE(142), IC )
- (XS(6), TMPTAB), (XS(14), DEFNAM), (XS(18), LEN1 ), (XS(19), LEN2 )
- (XS(20), IDC ), (XS(20), RETC )

**RTE ROUTINES USED:**

- CLOSE, EXEC, KCVT, OPEN, READF

**FDS ROUTINES USED:**

- XERTH, XREQ, XREXT, XRMV, XRMG, XRPCK, XRPK
249 1 BEGIN XDEF
250 2 CREATE DEFAULT INTERFACE TABLE NAME FROM PROCESSOR NAME
251 2 CALL OPEN TO OPEN FILE
252 2 ERREXIT IF OPEN ERROR TO :FILERR:
253 2 CALL READ TO READ RECORD 1
254 2 ERREXIT IF READ ERROR TO :FILERR:
255 2 IF THERE IS LITERAL DATA THEN
256 3 CALL READ TO READ RECORD 2
257 3 ERREXIT IF READ ERROR TO :FILERR:
258 2 ENDIF
259 2 CALL CLOSE TO CLOSE FILE
260 2 ERREXIT IF CLOSE ERROR TO :FILERR:
261 2 CALL XREG TO ALLOCATE & STORE XINTAB
262 2 ERREXIT IF RETURN CODE IS NOT ZERO TO :MGERR:
263 2 IF DEFAULT TABLE IS INCOMPLETE THEN
264 3 SET INTERFACE TABLE NAME IN XE TO XINTAB
265 3 SET SUBSTATE TO INTERFACE TABLE EDITOR
266 3 CALL XERTN TO IMPLICITLY CALL INT EDITOR **NO RETURN**
267 2 ENDIF
268 1 EXIT XDEF

269 2 :FILERR:
270 2 CALL CLOSE TO CLOSE FILE
271 2 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
272 1 EXIT XDEF

273 2 :MGERR:
274 2 CALL XRMSG TO DISPLAY SPACE ERROR
275 2 CALL EXEC TO FREE CLASS NUMBER
276 2 SET MASTER STATE TO DIRECTIVE LEVEL
277 2 CALL XRTH TO RETURN TO EXEC **NO RETURN**
278 1 END XDEF
procedure for calling xxman:
call xxman

xxman is the manual execution controller

inputs from common:
xe(5) massta, xb(249) segno

outputs to common:
xe(10) sestr, xe(11) segend, xb(249) segno,
xb(250) serlen, xb(251) sequab, xs(13) flgtab

common used:
equivalence (xe(5), massta), (xe(10), sestr),
+ (xe(11), sereno),
+ (xb(249), segno), (xb(250), serlen),
+ (xb(251), sequab), (xs(13), flgtab)

fds routines used:
xext, xmsg, xcom, xxexe,
xdec, xxdef, xxsto, xxtmp

rte routines used:
ior
BEGIN

IF ENTRY IS FROM A DIRECTIVE THEN

SET HASITA TO INDICATE RE-ENTRY

DO UNTIL PERCENT IS ENTERED

: PROMPT: CALL XITCOM TO PROMPT FOR PRNAME, "ITNAME"

IF PERCENT IS NOT ENTERED THEN

ERROR IF CR ENTERED TO :PROMPT:

CALL XIDEC TO DECODE RESPONSE

ERROR IF INVALID RESPONSE TO :PROMPT:

SET SER #5 IN ZE : Zeros

SET #ENTRIES IN SERTAB IN XE TO 1

CALL XISTO TO STORE SEQUENCE TABLE

IF IT NAME IN SERTAB IS &INTAB THEN

PROCEDURE REQUIRES AN IT THEN

CALL XIDF TO READ UP DEFAULT INTERFACE TABLE

ENDIF

CALL XXEKE TO EXECUTE SERTAB

ENDIF

ENDDO

ELSE - AM BEING REENTERED FROM INTERFACE TABLE EDITOR

CALL XXMTP TO SET UP TO EXECUTE A TEMPORARY TABLE

PERFORM XXMAN **NO RETURN**

ENDIF

RETURN

: PRMERR: CALL XRMSGR TO DISPLAY ERROR

PERFORM XXMAN **NO RETURN**

END XXMAN
**FORTRAN CALLING SEQUENCE:**

```plaintext
CALL XXSEM
```

**INPUTS IN COMMON:**

```plaintext
XE(5) MASSTA, XE(10) SERSTA, XE(11) SEREND, XE(12) SEQ'TR,
XE(14) TABEND, XB(1) NOPROC, XB(2) LIBD,
XB(245) SERNO, XB(250) SELEN, XB(251) SERTAB
```

**OUTPUTS IN COMMON:**

```plaintext
XE(5) MASSTA, XE(13) FLGTAB
```

**INTERNAL COMMON USED:**

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

**COMMON USED:**

```plaintext
+ (XE(10), SERSTA), (XE(11), SEREND),
+ (XE(12), SERSTA), (XE(85), EOS ),
+ (XE(86), INTGR ), (XE(100), AMPER ),
+ (XE(139), EXEND ), (XE(140), TABEND),
+ (XE(141), CURIND ), (XE(145), COMBUS),
+ (XB(1), NOPROC), (XB(22), LIBD ),
+ (XB(235), RESIND),
+ (XB(236), ASCENT), (XB(246), OLDIND),
+ (XB(249), SERNO ), (X(125), SELNEN).
```

**FDS ROUTINES USED:**

```plaintext
SRCPR, XREXT, XRXG, XRMNV,
XXRSG, XRSET, XXDEC, XXDEF, XXEXF,
XXSTO, XXTHP, XXCM
```
BEGIN XXSEM:
IF ENTRY IS FROM A DIRECTIVE THEN
SET MASTER STATE TO SAT REENTRY
DO FOR # ENTRY.. IN SEQUENCE TABLE
SEARCH LIST FOR PROCESSOR NAME
ERROR IF NOT FOUND TO :ERR:
MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
ENDDO
SAVE ORIGINAL ENDING SEQUENCE #
SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
SET FLTAB TO SAT EXECUTE ENTIRE SEQTAB
CALL XISTO TO STORE RESID UE IN AW A
ELSE (I AN BEING REENTERED FROM INTE)
CALL XETMP TO SET UP TEMPORARY EXECUTI\N
CALL XXERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQTAB
IF RESET WAS REQUESTED THEN
RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
ELSE
EXIT XEUT IF TERMINAL ENTRY WAS JUST EXECUTED
SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
ENDIF
DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
CALL XTCOM TO PROMPT USER WITH CURRENT ENTRY
EXIT XXSEM IF RESPONSE IS X
IF RESPONSE IS CR THEN
IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
CALL XIDF TO READ UP DEFAULT TABLE
ENDIF
SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
CALL XXERE TO EXECUTE
SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
ELSE
IF RESPONSE WAS A SEQUENCE # THEN
ERROR IF NUMBER IS ZERO TO :ERR:
SEARCH SEQUENCE TABLE FOR SEQUENCE #
ERROR IF NUMBER IS NOT FOUND TO :ERR:
SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
ELSE
IF RESPONSE IS AN AMPSERAND THEN
SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
CALL XEDC TO DECODE PROCESSOR NAME, IT NAME RESPONSE
IF RSP,ONSE IS VALID THEN
CALL XISTO TO STORE OVERriding ENTRY
SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
SET CURRENT SEQUENCE # TO FIRST ENTRY
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
EXIT XXSEM
:ERR:
CALL XEDTSG TO DISPLAY ERROR MESSAGE
EXIT XXSEM
CALLING PROCEDURE FOR XXSTO:

CALL XXSTO

XXSTO SETS UP TO REQUEST MANAGER TO STORE 6SET0 AND HANDLE ANY ERRORS RETURNED BY THE MANAGER.

INPUTS FROM COMMON:

X8(250) SELEN, X8(251) SENTAB, X8(13) FLGTAB

OUTPUTS TO COMMON:

X8(5) MASSTA

COMMON USED:

EQUIVALENCE (X8(5), MASSTA),

(X8(19), REG1TB), (X8(20), REG2TB), (X8(250), SELEN), (X8(251), SENTAB), (X8(13), FLGTAB)

FDS ROUTINES USED:

XERR, XREMOV, XRSIG, XERTM

RTE ROUTINES USED:

EXEC
1 BEGIN XXST0
2   SET CLASS NUMBER TO ZERO
3 IF TABLE FLAG SAYS STORE ENTIRE 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  ENDIF
12  CALL XRER TO REQUEST MANAGER TO STORE -SEGTO
13 IF RETURN CODE IS NOT ZERO THEN
14   CALL XERROR TO WRITE SPACE ERROR
15  CALL EXEC TO RELEASE CLASS 0
16  SET MASTER STATE TO ZERO
17  CALL XERTH TO RETURN TO EXEC **NO RETURN**
18  ENDIF
19  RETURN
20 1 END XXST0
630 1 C8**********
631 1 C80 FORTRAN CALLING SEQUENCE:
632 1 C80 CALL XITHP
633 1 C80 C8**********
634 1 C80 XITHP SETS UP A ONE ENTRY SEQUENCE TABLE USING SINTAB AND
635 1 C80 STORES IT IN THE AMA TO EXECUTE WHEN EXECUTION WITH A
636 1 C80 TEMPORARY ENTRY IS NECESSARY
637 1 C80 C8**********
638 1 C80 INPUTS FROM COMMON:
639 1 C80 XE(12) SERPTR, XB(250) SEQLEN, XB(251) SERTAB
640 1 C80 C8**********
641 1 C80 OUTPUTS TO COMMON:
642 1 C80 XE(6) SUBSTA, XB(249) SEQNO, XB(250) SEQLEN,
643 1 C80 XB(251) SERTAB, XS(13), FLGETAB
644 1 C80 C8**********
645 1 C80 COMMON USED:
646 1 C80 EQUIVALENCE (XE(6), SUBSTA), (XE(12), SERPTR),
647 1 C80 + (XE(19), SERPTR), (XE(20), REBUF),
648 1 C80 + (XB(249), SEQNO), (XB(250), SEQLEN),
649 1 C80 + (XB(251), SERTAB), (XS(13), TMPTAB),
650 1 C80 C8**********
651 1 C80 FD5 ROUTINES USED:
652 1 C80 XRED, XMNOV, XGSTO
653 1 C80 C8**********
654 1 C80 RTE ROUTINES USED:
655 1 C80 EXEC
656 1 C80
ASSGN - DATA ASSIGNMENT PROCESSOR

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-NODE EXPRESSIONS AND FUNCTIONS AND ALLOWS REPEETITIVE EVALUATIONS IN ORDER TO COMPUTE AND STORE MULTIPLE VALUES

INPUTS FROM THE MANAGER:

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

INPUTS FROM THE INTERFACE TABLE:

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

OUTPUTS TO THE ANA:

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

INTERNAL VARIABLES:

BLANK COMMON - ASGCOM 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(1) = LU, PARMS(?) = DEBUG FLAGS</td>
<td>5</td>
<td>1</td>
<td>PARMS</td>
</tr>
<tr>
<td>STWIDTH</td>
<td>1</td>
<td>38</td>
<td>SYMBOL TABLE WIDTH</td>
</tr>
<tr>
<td>STLONG</td>
<td>1</td>
<td>39</td>
<td>SYMBOL TABLE LENGTH</td>
</tr>
<tr>
<td>LASTSY</td>
<td>1</td>
<td>40</td>
<td>LAST SYMBOL TABLE ENTRY DEFINED</td>
</tr>
<tr>
<td>SYNTAB</td>
<td>12,81</td>
<td>41</td>
<td>SYMBOL TABLE (WORDS 1-8 = TOC ENTRY OR APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR SUBSCRIPTED DATA ELEMENT)</td>
</tr>
<tr>
<td>SSTING</td>
<td>247</td>
<td>1013</td>
<td>RESULT STACK USED DURING POST-FIX STRING EVALUATION (EACH ELEMENT IS A STRING EXPRESSION)</td>
</tr>
<tr>
<td>RESULT</td>
<td>4,35</td>
<td>1260</td>
<td>FIXED DATA</td>
</tr>
<tr>
<td>WB</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>12</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>13</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>14</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>15</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>16</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>17</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>18</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>19</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>20</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>21</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>22</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>23</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>24</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>25</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>26</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>27</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>28</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>29</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>30</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>31</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>32</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>33</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>34</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>35</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>36</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>37</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>38</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>39</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>40</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>41</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>42</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>43</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>44</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>45</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>46</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>47</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>48</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>49</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>50</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>51</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>52</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>53</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>54</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>55</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>56</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>57</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>58</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>59</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WB</td>
<td>60</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TOKEN</td>
<td>PRIORITY</td>
<td>INPUT PRIORITY</td>
<td>OUTPUT PRIORITY</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>CD4</td>
<td>107</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>108</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>109</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>110</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>111</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>112</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>113</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>114</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>115</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>116</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>118</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>119</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>120</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>121</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>122</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>123</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>124</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>125</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>126</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>127</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>128</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>129</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>130</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>131</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>132</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>133</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>134</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>135</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>136</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>137</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>138</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>139</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>140</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>141</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>142</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>143</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>144</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>145</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>146</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>147</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>148</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>149</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>150</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>151</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>152</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>153</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>154</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>155</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>260</td>
<td>1 CD5</td>
<td>\texttt{&lt;CONSTANT&gt;} ::= \texttt{INTEGER} //</td>
</tr>
<tr>
<td>261</td>
<td>1 CD5</td>
<td>\texttt{SINGLE PRECISION REAL} //</td>
</tr>
<tr>
<td>262</td>
<td>1 CD5</td>
<td>\texttt{DOUBLE PRECISION REAL}</td>
</tr>
<tr>
<td>263</td>
<td>1 CD5</td>
<td>\texttt{&lt;RANGE&gt;} ::= \texttt{&lt;RANGE&gt;} \texttt{&lt;LIMITS&gt;} //</td>
</tr>
<tr>
<td>264</td>
<td>1 CD5</td>
<td>\texttt{&lt;RANGE&gt;} \texttt{&lt;LIMITS&gt;} //</td>
</tr>
<tr>
<td>265</td>
<td>1 CD5</td>
<td>\texttt{&lt;RANGE&gt;} \texttt{&lt;LIMITS&gt;} //</td>
</tr>
<tr>
<td>266</td>
<td>1 CD5</td>
<td>\texttt{&lt;LIMITS&gt;} ::= \texttt{INDEX=INTEGER,INTEGER}</td>
</tr>
<tr>
<td>267</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>268</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>269</td>
<td>1 CD5</td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>1 CD5</td>
<td></td>
</tr>
</tbody>
</table>
279 1 CD 6***
280 1 CD 0         DDRSP - DATA BOX DISPLAY PROCESSOR
281 1 CD 0
282 1 CD 0
283 1 CD 0
284 1 CD 0
285 1 CD 0
286 1 CD 0
287 1 CD 0
288 1 CD 0
289 1 CD 0
290 1 CD 0
291 1 CD 0
292 1 CD 0
293 1 CD 0
294 1 CD 0
295 1 CD 0
296 1 CD 0
297 1 CD 0
298 1 CD 0
299 1 CD 0
300 1 CD 0
301 1 CD 0
302 1 CD 0
303 1 CD 0
304 1 CD 0
305 1 CD 0
306 1 CD 0
307 1 CD 0
308 1 CD 0
309 1 CD 0
310 1 CD 0
311 1 CD 0
312 1 CD 0
313 1 CD 0
314 1 CD 0
315 1 CD 0
316 1 CD 0
317 1 CD 0
318 1 CD 0
319 1 CD 0
320 1 CD 0
321 1 CD 0
322 1 CD 0
323 1 CD 0
324 1 CD 0
325 1 CD 0
326 1 CD 0
327 1 CD 0
328 1 CD 0
329 1 CD 0
330 1 CD 0
331 1 CD 0
332 1 CD 0
333 1 CD 0
334 1 CD 0
335 1 CD 0
336 1 CD 0
337 1 CD 0

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

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

<table>
<thead>
<tr>
<th>OVERLAYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIDIM</td>
</tr>
<tr>
<td>XIDP1</td>
</tr>
<tr>
<td>XIDP2</td>
</tr>
</tbody>
</table>

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

INPUTS TO DBDSP FROM INTERFACE TABLE

<table>
<thead>
<tr>
<th>INPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATBOX - DATA BOX FILE NAME</td>
</tr>
<tr>
<td>NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER</td>
</tr>
<tr>
<td>KEP - DISPLAY VARIABLE SCALE LIST SET UP BY USER</td>
</tr>
<tr>
<td>VIODEF - CONSTRAINT VARIABLE DEFINITION LIST</td>
</tr>
</tbody>
</table>

INPUTS TO DBDSP FROM ORDE FILE

<table>
<thead>
<tr>
<th>RECORD 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) - NAME OF FDS PROCESSOR CREATION FILE</td>
</tr>
<tr>
<td>(4) - INTERFACE TABLE VARIABLE NAME FOR THIS FILE</td>
</tr>
<tr>
<td>(7) - NAME OF FDS PROCESSOR UPDATING FILE</td>
</tr>
<tr>
<td>(10) - INTERFACE TABLE VARIABLE NAME FOR THIS U-DATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1M) - NO OF ENTRIES IN SUMMARY TABLE</td>
</tr>
<tr>
<td>(2F) - X SCAN VARIABLE (6 CHAR)</td>
</tr>
<tr>
<td>(5F) - X FIRST SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>(6F) - X SECOND SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>(7F) - X UNITS (6 CHAR)</td>
</tr>
<tr>
<td>(10F) - X CENTROID (REAL)</td>
</tr>
<tr>
<td>Line</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>138</td>
</tr>
<tr>
<td>139</td>
</tr>
<tr>
<td>140</td>
</tr>
<tr>
<td>141</td>
</tr>
<tr>
<td>142</td>
</tr>
<tr>
<td>143</td>
</tr>
<tr>
<td>144</td>
</tr>
<tr>
<td>145</td>
</tr>
<tr>
<td>146</td>
</tr>
<tr>
<td>147</td>
</tr>
<tr>
<td>148</td>
</tr>
<tr>
<td>149</td>
</tr>
<tr>
<td>150</td>
</tr>
<tr>
<td>151</td>
</tr>
<tr>
<td>152</td>
</tr>
<tr>
<td>153</td>
</tr>
<tr>
<td>154</td>
</tr>
<tr>
<td>155</td>
</tr>
<tr>
<td>156</td>
</tr>
<tr>
<td>157</td>
</tr>
<tr>
<td>158</td>
</tr>
<tr>
<td>159</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>161</td>
</tr>
<tr>
<td>162</td>
</tr>
<tr>
<td>163</td>
</tr>
<tr>
<td>164</td>
</tr>
<tr>
<td>165</td>
</tr>
<tr>
<td>166</td>
</tr>
<tr>
<td>167</td>
</tr>
<tr>
<td>168</td>
</tr>
<tr>
<td>169</td>
</tr>
<tr>
<td>170</td>
</tr>
<tr>
<td>171</td>
</tr>
<tr>
<td>172</td>
</tr>
<tr>
<td>173</td>
</tr>
<tr>
<td>174</td>
</tr>
<tr>
<td>175</td>
</tr>
<tr>
<td>176</td>
</tr>
<tr>
<td>177</td>
</tr>
<tr>
<td>178</td>
</tr>
<tr>
<td>179</td>
</tr>
<tr>
<td>180</td>
</tr>
<tr>
<td>181</td>
</tr>
<tr>
<td>182</td>
</tr>
<tr>
<td>183</td>
</tr>
<tr>
<td>184</td>
</tr>
<tr>
<td>185</td>
</tr>
<tr>
<td>186</td>
</tr>
<tr>
<td>187</td>
</tr>
<tr>
<td>188</td>
</tr>
<tr>
<td>189</td>
</tr>
<tr>
<td>190</td>
</tr>
<tr>
<td>191</td>
</tr>
<tr>
<td>192</td>
</tr>
<tr>
<td>193</td>
</tr>
<tr>
<td>194</td>
</tr>
<tr>
<td>195</td>
</tr>
<tr>
<td>196</td>
</tr>
</tbody>
</table>

(12) - X INCREMENT (REAL)
(14) - X NUMBER OF STEPS (INTEGER 1-5)
(15) - YSCAN VARIABLE (6 CHAR)
(18) - Y FIRST SUBSCRIPT (INT OR ZERO)
(19) - Y SECOND SUBSCRIPT (INT OR ZERO)
(20) - Y UNITS (6 CHAR)
(22) - Y CENTROID (REAL)
(23) - YX INCREMENT (REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPANDBABLE VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

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

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

CD A COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPRM
CD A DKEY - NAME DATA BOX TO BE DISPLAYED BY DBDSP
CD A DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
CD A JCVAR - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)
CD A JCVARS - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD A JCVARS2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD A JFDAT - NAME OF COMMON AREA USED FOR INTERFACE TABLE
CD A JFDAT2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
CD A JFDAT3 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
CD A JFDAT4 - LENGTH OF COMBUF IN 16-BIT WORDS - XPRM
CD A JFDAT5 - LENGTH IN WORDS OF CHAR STRING USED FOR USER PROMPT - XPRM
CD A JFDAT6 - ARRAY CONTAINING INCLUDES FOR 8 CONSTR (4 X 121 SIZE)
CD A JFDAT7 - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCM
CD A JFDAT8 - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
CD A JFDAT9 - LIST OF CONSTRAINT RELATION INPUT BY USER
CD A JFDAT10 - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
CD A JFDAT11 - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
CD A JFDAT12 - NUMBER OF DEP DISP VAR INPUT NUMBER LIST (INTEGER)
CD A JFDAT13 - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR 0/P
CD A JFDAT14 - SET OF INDICATORS FOR CONSTRAINTS VIOLATED-0/NOT 0
CD A JFDAT15 - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
CD A JFDAT16 - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
CD A JFDAT17 - STRING CONTAINING USER PROMPT MESSAGE
CD A JFDAT18 - LIST OF XVAR VALUES FOR X COORDINATES (1 - 11 REAL)
CD A JFDAT19 - NAME OF XVAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
397 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
398 1 CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)
399 1 CD 4 YSCANN - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
400 1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
401 1 CD 4 ZTABLE - TABLE IN COMMON FOR SUNTAB VARIABLE NAMES AND UNITS
402 1 CD 4 NAMVUL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCH
403 1 CD 4 SUNTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
404 1 CD 4 PARMS - COMMUNICATION BUFFER FOR BMPAR - LU, USER ID, FLAGS
405 1 CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATM
406 1 CD 4 LUSDSP - BMPAR 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 NO 0 RUN ALL DISPLAYS TO O/P
410 1 CD 4 WITHOUT PROMPT
411 1 CD 4 CARTAG - CARTAGE USED TO LOCATE DATA BOX
412 1 CD 4 CVALUE -
413 1 CD 4 ********
414 1 CD 5
415 1 CD 5
416 1 CD 5
417 1 CD 5 XPRM, XELBS, XPXIT, BMPAR
418 1 CD 5
419 1 CD 5
420 1 CD 5
421 1 CD 5********
**C**

- **DEFIN** IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

**C**

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

**C**

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

**C**

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

**C**

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

**C**

- **INTERNAL VARIABLES:**
  - **INBUF** - INTERFACE TABLE HEADER
  - **ILENG** - LENGTH OF SYMBOLIC STRING
  - **ITOKEN** - POSITION WITHIN THE SYMBOLIC STRING
  - **NAME** - BUFFER FOR VALID NAMES TO BE ALLOCATED
  - **NBUF** - BUFFER AREA FOR XGET AND XREP USE
  - **NEXT** - NUMBER OF ENTRIES IN THIS AWA REQUEST
  - **NAME** - NUMBER OF NAMES IN SYMBOLIC STRING
  - **STRING** - SYMBOLIC STRING INPUT TO **DEFIN**

**C**

- **EXTERNAL ROUTINES USED:**
  - **EXEC, IAMP, KCVT, XMPAR, XPIET**
  - **XPREQ, XPRF, XPROP, XIDFT, XIMSG**
BEGIN DEF1N
CALL RMPAR TO GET LU AND DEBUG FLAGS
CALL APGET TO RETRIEVE SYMBOLIC STRING
GET STRING LENGTH FROM INTERFACE TABLE HEADER
START SEARCH WHILE TOKEN-POSITION .LT. STRING-LENGTH, OR
WHILE CURRENT-TOKEN .NE. END-OF-STRING
PERFORM ZDPM TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
EXIT IF THERE WAS AN ERROR RETURN FROM ZDPM
CALL XIMP6 TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
SET PROCESSOR RETURN CODE TO ABEND
INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
END LOOP
SET PROCESSOR RETURN FOR NORMAL EXIT
END SEARCH
SET OPTION SO XPR6 WILL DO A QUEUE REQUEST
DO FOR NUMBER OF REQUESTS IN REQUEST BUFFER (MAXARR)
IF THIS IS LAST REQUEST, THEN
SET OPTION TO CLOSE REQUEST BUFFER
ENDIF
COMPUTE INDEX TO THIS REQUEST
CALL XPREQ TO QUEUE THIS REQUEST
ENDIF
CALL XPR6 TO RETURN TO THE MANAGER
END DEF1N
BEGIN XIDPM
SET IDIM AND JDIM TO 1
IF TOKEN IS NOT A NAME THEN
SET ERROR CODE
ELSE
MOVE NAME INTO REQUEST
INCREMENT TO NEXT TOKEN
IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREN) THEN
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT AN INTEGER OR
TOKEN IS NOT ZERO THEN
CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
EXIT TO :PNERR:
ENDIF
SET IDIM TO THIS TOKEN
INCREMENT TO NEXT TOKEN
IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT AN INTEGER OR
TOKEN IS NOT ZERO THEN
CALL XIMSG TO DISPLAY ERROR "INVALID JOIN"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
ENDIF
IF TOKEN IS NOT A BEGIN TYPE FIELD SLASH THEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
IF TOKEN IS NOT A NAME THEN
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
STARTSEARCH FOR ALL VALID TYPES
EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
SET TYPE AND CLASS IN REQUEST
COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
IF SIZE IS TOO LARGE (>1200 WORDS) THEN
CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
STARTSEARCH FOR ALL VALID TYPES
EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
SET TYPE AND CLASS IN REQUEST
COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
IF SIZE IS TOO LARGE (>1200 WORDS) THEN
CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
ENDFOR
EXIT TO :PNERR:
INCREMENT TO NEXT TOKEN
ENDFOR
CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
STARTSEARCH FOR ALL VALID TYPES
EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
SET TYPE AND CLASS IN REQUEST
COMPUTE SIZE AS IDIM * JDIM * LENGTH OF TYPE
IF SIZE IS TOO LARGE (>1200 WORDS) THEN
CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
EXIT TO :PNERR:
ENDIF
INCREMENT TO NEXT TOKEN
CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
3  BUILD REQUEST TO DELETE/VERIFY ABSENT THIS DATA ELEMENT
4  BUILD REQUEST TO ALLOCATE THIS DATA ELEMENT
5  ENDIF
6  1 EXIT X10PN

$PHEPR1:
2  CALL X10MSG TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
3  1 END X10PN
ENSC IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

ENSC 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:

XZRET - RETURN CODE TO MANAGER

OUTPUTS TO AWA:

XSCAN - NEW VALUE FOR X SCAN VARIABLE
YSCAN - NEW VALUE FOR Y SCAN VARIABLE

RTE ROUTINES USED:

CLOSE, EXEC, FLOAT, KCVT, POIN
READ, RMPAR, WRIT

FDS ROUTINES USED:

XPXIT, XREXT, XMMOV, XUDBG, XVPAM
XIFCL, XMSB
1 BEGIN SETXY
2 IF XCUR IS END STEP THEN
3 IF THERE IS 1 VARIABLE THEN
4 SET X TO CENTROID
5 CALL POSN TO POSITION FILE TO CENTROID RECORD
6 ERREXIT IF POSN ERROR TO :ERR4;
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 :ERR4:
13 ELSE
14 SET YCUR TO (-XSTEP)
15 IF XSTEP IS ZERO AND Y IS CENTROID THEN
16 INCREMENT YCUR BY 1
17 CALL WRITF TO WRITE DUMMY AS CENTROID RECORD
18 ERREXIT IF WRITF ERROR TO :ERR4:
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 WRITF TO WRITE DUMMY AS CENTROID RECORD
29 ERREXIT IF WRITF ERROR TO :ERR4:
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 REBUF
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 TSCAN (IF ENTERED) VALUES COMPUTED USING THE CENTROID, THE INCREMENT, AND THE CURRENT STEP NUMBER, CREATING A DATA BOX FILE.

INPUTS FROM MANAGER:

- Logical unit of the user's terminal
- Use ID - user ID code
- Flags - flags for debug
- Entsub - displacement of this scan entry in bsegtb

INPUTS FROM INTERFACE TABLE:

- Procon - cartridge # for data box file
- Sumtab - summary table
- *datbox-name of data box entered
- Nvvar - number of scan variables
- *scan-name of x scan variable
- Xunit - unit of x variable
- Xcent - centroid of x variable
- Xincr - increment for x
- Xsteps - number of steps for x
- *tscan - name of y scan variable
- Yunit - unit of y variable
- Ycent - centroid of y variable
- Yincr - 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
- Yscan - y scan variable
- Bscntb - scan control table (173 words per active scan)

1. Sumtab & char name of summary table
2. Displ displacement for summary table
3. Datbox & char qualified name of data box file
4. Reset reset sequence number
5. Centred centroid record number
6. Xscan name of x variable
7. Xdispl displacement for x
8. Xcent centroid for x
9. Xincr increment for x
789 1 CD3
(18) XSTEP # STEPS OF X
790 1 CD3
(19) XCUR CURRENT X STEP NUMBER
791 1 CD3
(20) YSCAN NAME OF Y VARIABLE
792 1 CD3
(21) STEP @ STEPS FOR Y
793 1 CD3
(24) YCENT CENTROID OF Y
794 1 CD3
(26) YINCR INCREMENT FOR Y
795 1 CD3
(27) YSTEP # STEPS OF Y
796 1 CD3
(29) YCUR CURRENT Y STEP NUMBER
797 1 CD3
(30) IDCB 144 WORD DCB FOR DATBOX
798 1 CD************
799 1 CD5
RTE SUBROUTINES USED:
800 1 CD5
CLOSE, CREAT, EXEC, FLOAT, IAND,
801 1 CD5
KCVT, MOD, POSN, PURGE, RMFAR,
802 1 CD5
WRITE
803 1 CD5
FDC SUBROUTINES USED:
804 1 CD5
XPATR, XPGET, XPPUT, XPYIT, XRCPR,
805 1 CD5
XREXT, XRMOV, XRRFM, XRUNG, XUDBG,
806 1 CD5
XVPAM, XIMSG
807 1 CD5
808 1 CD5
809 1 CD5
810 1 CD5
811 1 CD5
812 1 CD************
BEGIN SCAN
1 CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
2 CALL XPGET TO GET PROCID AND # SCAN VARIABLES
3 ERREXIT IF # SCAN VARIABLES < 1 OR > 2 TO :ERR3:
4 GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
5 ERREXIT IF SUMTAB IS A LITERAL TO :ERR3:
6 ERREXIT IF DISPLACEMENT IS NOT AN ELEMENT BOUNDARY TO :ERR3:
7 GET DBOX NAME FROM INTERFACE TABLE
8 DO FOR # SCAN VARIABLES
9 CALL XPGET TO GET NAME AND DISPLACEMENT
10 COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND IDIN
11 CALL XPGET TO GET UNITS, CENTROID, INCR, # STEPS
12 ERREXIT IT # STEPS < 0 OR > 5 TO :ERR3:
13 ENDDO
14 SAVE REQUEST TO RETRIEVE VALUES FOR $SETB AND $SCNTB
15 CALL XVPM TO REQUEST OWN MANAGEMENT
16 CALL EXEC TO READ IN $SETB
17 IF $SCNTB NOT FOUND THEN
18 SET # SCANS TO ZERO
19 ELSE
20 SET # SCANS TO (TOTAL SIZE OF $SCNTB / SIZE OF ONE SCAN ENTRY)
21 CALL EXEC TO READ IN $SCNTB
22 ERREXIT IF # SCANS = MAXIMUM ALLOWED (4) TO :ERR3:
23 ERREXIT IF THIS DATABASE NAME IS ALREADY IN USE TO :ERR3:
24 ENDF
25 ERREXIT IF THIS IS THE LAST ENTRY IN $SETB TO :ERR3:
26 GET THE SEQUENCE NUMBER OF THIS SCAN FROM $SETB
27 IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
28 SEARCH $SETB FOR THE SEQUENCE NUMBER
29 ERREXIT IF THIS SCAN IS THE LAST ENTRY IN $SETB TO :ERR3:
30 IF THIS IS A SEMI-OVERRIDE (2 PROCESSOR NAMES NOT EQUAL) THEN
31 SET RESET NUMBER TO THIS ENTRY SEQUENCE NUMBER
32 ELSE
33 SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
34 ENDF
35 ELSE
36 SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
37 ENDF
38 COMPUTE SIZE OF DATBOX FILE = ((2 * XSTEPS + 1) * (2 * YSTEPS + 1) + 1) / 2
39 COMPUTE CENTROID RECORD NUMBER = SIZE + 3
40 DO FOR # SCAN VARIABLES
41 COMPUTE BEGINNING VALUE = (CENT + INCR * FLOAT (CUR STEP))
42 ENDB
43 CALL XPPUT TO STORE DATBOX AND SCAN VARIABLES
44 CALL CREAT TO CREATE DATBOX FILE
45 IF FILE ALREADY EXISTS THEN
46 CALL PURGE TO PURGE FILE
47 ERREXIT IF PURGE ERROR TO :ERR2:
48 CALL CREAT TO CREATE FILE
49 ENDF
50 CALL WRIT TO WRITE HEADER RECORD TO DATBOX
51 ERREXIT IF WRITE ERROR TO :ERR2:
52 CALL POSMT TO POSITION FILE TO FIRST DATA RECORD
53 ERREXIT IF POSMT ERROR TO :ERR2:
54 CALL EXEC TO WRITE $SCNTB
55 SAVE REQUEST TO DELETE/VERIFY ABSENT $SCNTB IN REQBUF
56 SAVE REQUESTS TO ALLOCATE AND STORE VALUES FOR NEW $SCNTB
57 CALL XVPM TO REQUEST OWN MANAGEMENT
58 SCAN
59 SCAN
60 SCAN
61 SCAN
62 SCAN
63 SCAN
64 SCAN
65 SCAN
66 SCAN
67 SCAN
68 SCAN
69 SCAN
70 SCAN
71 SCAN
72 SCAN
ERREXIT IF NO AWA SPACE TO :ERR1:
CALL EXIT TO EXIT NORMALLY
EXIT SCAN

IF THERE ARE MORE ACTIVE SCANS (# SCANS > 0) THEN
CALL EXEC TO READ IN NEW BSCNTB
CALL EXEC TO WRITE OUT ORIGINAL BSCNTB
SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL BSCNTB
ENDIF

:ERR2:
CALL CLOSE TO CLOSE DATBOX
CALL PURGE TO PURGE DATBOX
SET VALUE FOR XPPUT
SAVE REQUEST TO DELETE DATBOX FROM AWA
CALL XPWAN TO REQUEST AWA MANAGEMENT

:ERR3:
CALL ZIMSG TO DISPLAY ERROR
CALL XPXIT TO ABEND SCAN
END SCAN
1 BEGIN XICHR
2    SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
3      -3,-2,-1,1
4 CASE (RESULT DATA TYPE) :CHRSTR,:CHRREF,:CHRREF,:FIXERR:
5     2 3
6     :FIXERR,:FIXERR:
7 BEGIN:
8    DETERMINE # WORDS IN CHARACTER STRING
9 MOVE CHARACTER STRING TO RESULT LOCATION
10 IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
11 BLANK FILL AFTER CHARACTER STRING
12 ENDIF
13 BEGIN:
14 IF TOP ENTITY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
15 POP DISPLACEMENT FROM RESULT STACK
16 ELSE
17 SET DISPLACEMENT = 0
18 ENDIF
19 BEGIN:
20 IF RESULT OPERAND IS FREE, THEN
21 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
22 ELSE
23 CHARACTER = CHARACTER
24 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
25 ENDIF
26 CALL XPRET TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
27 IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
28 BLANK FILL REMAINING DATA
29 ENDIF
30 BEGIN:
31 IF FIXERR:
32 SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO"
33 NUMERICAL DATA"
34 CALL XKING TO OUTPUT MESSAGE TO USER
35 CALL XLIST TO LIST SYMBOLIC STRING
36 CALL XEXIT TO EXIT PROCESSOR
37 END XICHR
1053  1
1054  1
1055  1
1056  1
1057  1
1058  1
1059  1
1060  1
1061  1
1062  1
1063  1
1064  1
1065  1
1066  1
1067  1
1068  1
1069  1
1070  1
1071  1
1072  1
1073  1
1074  1
1075  1
1076  1
1077  1
1078  1
1079  1
1080  1
1081  1

FORTRAN CALLING SEQUENCE:

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

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

INPUTS FROM CALLING SEQUENCE:

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

OUTPUTS TO CALLING SEQUENCE:

INDEX - POSITION OF 'TOKEN' IN SYMBOLIC STRING IF FOUND,
OR 'END' IF NOT FOUND
1083  1 BEGIN  XZDFT
1084  2   SET INDEX TO START
1085  3   DO UNTIL INDEX \ Activity END
1086  4   IF ARRAY(INDEX) = EQ. TOKEN, THEN
1087  5   SET END TO INDEX
1088  6   ELSE
1089  7   CASE ARRAY(INDEX)+1:
1090  8   :ADD1:  0 END OF STRING
1091  9   :ADD2:  1 INTEGER
1092 10   :ADD3:  2 REAL
1093 11   :ADD4:  3 DOUBLE
1094 12   :ADD1:  4 UNKNOWN
1095 13   :ADD4:  5 NAME
1096 14   :ADD1:  6 UNKNOWN
1097 15   :ADD1:  7 UNKNOWN
1098 16   :CALST:  8 CHARACTER STRING
1099 17   :ADD1:  9 UNKNOWN
1100 18   :ADD1:  10 
1101 19   :ADD1:  11 -
1102 20   :ADD1:  12 *
1103 21   :ADD1:  13 /
1104 22   :ADD1:  14 <
1105 23   :ADD1:  15 >
1106 24   :ADD1:  16 #
1107 25   :ADD1:  17 @
1108 26   :ADD1:  18 =
1109 27   :ADD1:  19 &
1110 28   :ADD1:  20 ?
1111 29   :ADD1:  21 @
1112 30   :ADD1:  22 )
1113 31   :ADD2:  23 START OF SYMBOLIC STRING
1114 32   :ADD1:  24 END OF STRING
1115 33   :ADD1:  25 %
1139 1 CD ****
1140   CD 0
1141   CD 0
1142   CD 0
1143   CD 0
1144   CD 0
1145   CD 0
1146   CD 0
1147   CD 0
1148   CD 0
1149   CD 0
1150   CD 0
1151   CD 0
1152   CD 0
1153   CD 0
1154   CD 0
1155   CD 0
1156   CD 0
1157   CD 0
1158   CD 0
1159   CD 0
1160   CD 0
1161   CD 0
1162   CD 0
1163   CD 0
1164   CD 0
1165   CD 0
1166   CD 0
1167   CD 0
1168   CD 0
1169   CD 0
1170   CD 0
1171   CD 0
1172   CD 0
1173   CD 0
1174   CD 0
1175   CD 0
1176   CD 0
1177   CD 0
1178   CD 0
1179   CD 0
1180   CD 0
1181   CD 0
1182   CD 0
1183   CD 0
1184   CD 0
1185   CD 0
1186   CD 0
1187   CD 0
1188   CD 0
1189   CD 0
1190   CD 0
1191   CD 0
1192   CD 0
1193   CD 0
1194   CD 0
1195   CD 0
1196   CD 0
1197   CD 0

<table>
<thead>
<tr>
<th>Record 1</th>
<th>Record 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) - Name of FDS processor creating file</td>
<td>(1) - No of entries in summary table</td>
</tr>
<tr>
<td>(2) - X scan variable (6 char)</td>
<td>(2) - X scan variable (6 char)</td>
</tr>
<tr>
<td>(3) - X first subscript (INT or ZERO)</td>
<td>(3) - X first subscript (INT or ZERO)</td>
</tr>
<tr>
<td>(4) - Interface table variable name key for this file</td>
<td>(4) - Interface table variable name key for this file</td>
</tr>
<tr>
<td>(5) - X second subscript (INT or ZERO)</td>
<td>(5) - X second subscript (INT or ZERO)</td>
</tr>
<tr>
<td>(6) - X units (6 char)</td>
<td>(6) - X units (6 char)</td>
</tr>
<tr>
<td>(7) - X centroid (REAL)</td>
<td>(7) - X centroid (REAL)</td>
</tr>
<tr>
<td>(8) - X increment (REAL)</td>
<td>(8) - X increment (REAL)</td>
</tr>
<tr>
<td>(9) - X number of steps (INTEGER 1-5)</td>
<td>(9) - Y number of steps (INTEGER 1-5)</td>
</tr>
<tr>
<td>(10) - Y scan variable (6 char)</td>
<td>(10) - Y scan variable (6 char)</td>
</tr>
<tr>
<td>(11) - Y first subscript (INT or ZERO)</td>
<td>(11) - Y first subscript (INT or ZERO)</td>
</tr>
<tr>
<td>(12) - Y second subscript (INT or ZERO)</td>
<td>(12) - Y second subscript (INT or ZERO)</td>
</tr>
<tr>
<td>(13) - Y units (6 char)</td>
<td>(13) - Y units (6 char)</td>
</tr>
<tr>
<td>(14) - Y centroid (REAL)</td>
<td>(14) - Y centroid (REAL)</td>
</tr>
<tr>
<td>(15) - Y increment (REAL)</td>
<td>(15) - Y increment (REAL)</td>
</tr>
<tr>
<td>(16) - Y number of steps (INTEGER 1-5)</td>
<td>(16) - Y number of steps (INTEGER 1-5)</td>
</tr>
</tbody>
</table>

Dependent variable names and units in records 3, 4, & 5

Summary table records
- Each summary table contain value for each
1198 1 CD 2 DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES
1199 1 CD 2 INCLUDING ERROR FLAG WHICH IS FIRST VALUE
1200 1 CD 2 IN SUMMARY TABLE)
1201 1 CD 2
1202 1 CD 3 "OUTPUT FROM XZDP1
1203 1 CD 3
1204 1 CD 3 MASK TABLES CONTAINING THE CONSTRAINT MASKS FOR EACH
1205 1 CD 3 CONSTRAINT WHICH WAS VIOLATED AND INDICATION OF
1206 1 CD 3 WHETHER OR NOT ANY CONSTRAINT WAS VIOLATED FOR EACH
1207 1 CD 3 ARRAY COORDINATE
1208 1 CD 3
1209 1 CD 3 LIST OF VALID DISPLAY DEPENDANT VARIABLES FOR EACH
1210 1 CD 3 PAGE (UP TO 16 PAGES)
1211 1 CD 3 LIST OF VALID CONSTRAINT VARIABLES WHICH WERE VIOLATED
1212 1 CD 3 (UP TO 8 CONSTRAINTS)
1213 1 CD 3
1214 1 CD 3 DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES
1215 1 CD 3
1216 1 CD 3 "OUTPUT FROM XZDP2
1217 1 CD 3
1218 1 CD 3 DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
1219 1 CD 4 DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIP
1220 1 CD 4 TORS
1221 1 CD 4 IDY1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
1222 1 CD 4 IDY2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR
1223 1 CD 4 ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)
1224 1 CD 4 IXSCM1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
1225 1 CD 4 IXSCM2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
1226 1 CD 4 IYSCM - NAME OF COMMON AREA USCD FOR INTERFACE TABLE
1227 1 CD 4 IYSCN1 - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
1228 1 CD 4 IYSCN2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
1229 1 CD 4 MASK1 - ARRAY CONTAINING CONSTRAINTS A THRU D
1230 1 CD 4 "MASK2 - ARRAY CONTAINING CONSTRAINTS E THRU G
1231 1 CD 4 NAMVAR - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
1232 1 CD 4 NC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
1233 1 CD 4 NCREL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
1234 1 CD 4 NVAV - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
1235 1 CD 4 NVAR - LIST OF DEP DISP VARIABLE PAIRS PAGED OUT PUT (1-16PR)
1236 1 CD 4 NDVRC - NUMBER OF DEP DISP VAR IN NDVAR LIST (INTEGER)
1237 1 CD 4 NDVR - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR O/P
1238 1 CD 4 MKERR - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/MOT
1239 1 CD 4 NXSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1240 1 CD 4 NYSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1241 1 CD 4 XCOR0 - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
1242 1 CD 4 XSCNNM - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1243 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1244 1 CD 4 YCOR0 - LIST OF X VAR VALUES FOR Y COORDINATES (1 - 11 REAL)
1245 1 CD 4 YSCNNM - NAME OF Y VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1246 1 CD 4 YUNITS - NAME OF Y VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1247 1 CD 4 ITABLE - TABLE IN COMMON FOR SUMTAB VARIABLE NAMES AND UNITS
1248 1 CD 4 NAMVU - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
1249 1 CD 4 SINTAB - VALUES FOR SCAN VARIABLES(S) - 1 TO 32 VALUES/RECORD
1250 1 CD 4 PPARS - COMMUNICATION BUFFER FOR RPAR = LU, USER ID, FLAG
1251 1 CD 4 LU - LOGICAL UNIT # FOR XROM CALLING SEQUENCE - USER LOCATM
1252 1 CD 4 LUDSP - DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED LU
1253 1 CD 4 PROMPT - TABLE IN COMMON TO COMMUNICATE WITH XROM
1254 1 CD 4 DEBUG
1255 1 CD 4 SELECT = O PROMPT ; SELECT NOT O RUN ALL DISPLAYS TO O/P
XZDP2

XZDP2 - DATA BOX DISPLAY OVERLAY - PROMPTS USER FOR DISPLAY DESIRED. THEN FORMATS AND DISPAYS DATA ACCORDINGLY

- SCHEDULED BY DDBSP

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

INPUTS TO XZDP2 FROM XZDP1

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

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

DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES

OUTPUT FROM XZDP2

DISPLAY FORMAT SHOWN IN DOCUMENTATION IS SENT TO
NAME LU DEVICE

XZDP2

COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XROM

DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DDBSP

DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS

IDV1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR

IDV2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.

ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XZMSK (MAX OF 8)

IXSN1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)

IXSN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)

IPBUF - NAME OF COMMON AREA USED FOR INTERFACE TABLE

IVSN1 - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)

IVSN2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)

LENBF - LENGTH OF COMBUF IN 16-BIT WORDS - XROM

LENST - LENGTH IN WORDS OF CHAR STRING USED FOR USER PROMPT - XROM

MASK - ARRAY CONTAINING MASKS FOR UP TO 8 CONSTRAINTS (4 X 121 SIZE)

NANVRL - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN

NC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)

NCRELL - LIST OF CONSTRAINT RELATIONS INPUT BY USER

NCVARL - LIST OF CONSTRAINT VARIABLE NAMES INPUT B: USER (32 MAX)
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1591 C0********
1592 C00      FORTRAN CALLING PROCEDURE
1593 C00      CALL XZ1SP (STRING, LEN)
1594 C00      CD1
1595 C00      XZ1SP REMOVED DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1596 C00      CD1  A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1597 CD1
1598 CD1
1599 CD1
1600 CD1
1601 CD1
1602 CD1
1603 CD1
1604 CD2
1605 CD2
1606 CD2
1607 CD2
1608 CD2
1609 CD2
1610 CD2
1611 CD1
1612 CD1
1613 CD1
1614 CD1
1615 CD1
1616 CD1
1617 CD1
1618 CD1
1619 CD1
1620 CD1
1621 CD1
1622 CD4
1623 CD4
1624 CD4
1625 CD4
1626 CD4
1627 CD4
1628 CD4
1629 CD4
1630 BEGIN XZ1SP
1631 CD4  CALL XZ1SP TO REMOVE DUPLICATE BLANKS FROM STRING
1632 CD4  DO WHILE THERE ARE TRAILING WORDS IN STRING
1633 CD4  SET THIS TRAILING WORD TO BLANKS
1634 CD4  EMD000
1635 END XZ1SP
CD**********
CD          FORTRAN CALLING SEQUENCE:
CD          CALL XIFCL (LU)
CD**********
CD1          XIFCL SEARCHES XVSTB LOOKING FOR LU SO THAT THE
CD1          PROCESSOR'S CLASS NUMBER CAN BE EXTRACTED. IT
CD1          SAVES THE CLASS NUMBER IN EXTERNAL XPCLS.
CD**********
CD2          INPUTS FROM CALLING SEQUENCE:
CD2          LU - LOGICAL UNIT OF TERMINAL BEING USED.
CD**********
CD3          NOTE: CALLING PROGRAM MUST HAVE XPCLS DEFINED
CD3          AS AN EXTERNAL REFERENCE.
CD**********
CD4          *
CD5          *
BEGIN XIFCL
CALL .ENTR TO RESOLVE PARAMETER ADDRESS
STARTSEARCH FOR ALL XVSTB ENTRIES
EXIT IF THIS ENTRY'S LU IS MINE
ENDSEARCH
GET CLASS NUMBER FROM XVSTB
SAVE CLASS NUMBER IN XPCLS
END 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 ASGCOM - LUSSTRNG,OPRND$,OPINFO

OUTPUTS TO ASGCOM

RESULT,RLTPT,OPRND$

EXTERNAL REFERENCES

FDS - XPXIT, XRMOV, XILSS, XIZSG

RTE - ABS, ANMT, ALOG, ALGOT, AMOD, ATAN, ATAND2, COS, DABS, DATAN,

DATN2, DBLE, DCOS, DQINT, DEEP, DLOG, DLGOT, DMOD, DS1'W, DSIN, DSQRT,

EIP, FLOAT, IABS, IDINT, IFIX, ISIGN, MOD, OVF, SIGN, SIN, SMGLE, SQRT,

TAN, TANH
1755 1 BEGIN XZFE
1756 2  CASE (RESULT DATA TYPE ) :FRESTR:, :FREFRE:, :FREFFE:, :FREFIX:, 
1757 3 2 3 1
1759 3  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 :FREFRE:
1764 3 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1765 4 POP DISPLACEMENT FROM RESULT STACK
1766 3 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPTED
1767 4 SET DISPLACEMENT = 0
1768 3 ENDIF
1769 3 :FREFFE:
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 XRPC02 TO RETRIEVE 1 WORD FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
1773 4 SET # WORDS TO BE STORED IN OBJECT = 1
1774 3 ELSE FREE = CHARACTER DATA ELEMENT
1775 4 CALL XPR02 TO RETRIEVE LOGICAL ELEMENT OF CHARACTER DATA FROM RESULT OPERAND
1776 4 AT DISPLACEMENT DETERMINED
1777 4 SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR RESULT OPERAND
1778 3 ENDIF
1779 3 :FREFIX:
1780 3 CALL XZPC2 TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781 3 SET # WORDS TO BE STORED = RESULT DATA TYPE
1782 2 END CASE
1783 1 END XZFE
1785 1 CD0  FORTRAN CALLING PROCEDURE:
1786 1 CD0  CALL XZFXD
1787 1 CD0  
1788 1 CD0  *******************
1789 1 CD0  *******************
1790 1 CD0  *******************
1791 1 CD0  XZFXD IS USED BY THE ASGN ROUTINE XZPS2 TO PROCESS DATA ASSIGNMENTS
1792 1 CD0  FOR FIXED-TYPE OBJECT DATA ELEMENTS
1793 1 CD0  *******************
1794 1 CD0  *******************
1795 1 CD0  *******************
1796 1 CD0  *******************
1797 1 CD0  *******************
1798 1 CD0  *******************
1799 1 CD0  *******************
1800 1 CD0  *******************
1801 1 CD0  *******************
1802 1 CD0  *******************
1803 1 CD0  *******************
1804 1 CD0  *******************
1805 1 CD0  *******************
1806 1 CD0  *******************
1807 1 CD0  *******************
1808 1 CD0  *******************
1809 1 CD0  *******************
1810 1 CD0  *******************
1811 1 CD0  *******************
1812 1 CD0  *******************
1813 1 CD0  *******************
1 BEGIN XIXFD
2 SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
3 CASE (RESULT DATA TYPE ) :NOCHAR:, :FIXRE:, :FIXRE:, :FIXFIX:, :FIXFIX:
4 3 :
5 3 :
6 3 :
7 3 :
8 3 :
9 3 :
10 3 :
11 3 :
12 3 :
13 3 :
14 3 :
15 3 :
16 3 :
17 3 :
18 3 :
19 3 :
20 3 :
21 3 :
22 3 :
23 3 :
24 3 :
25 3 :
26 3 :
27 3 :
28 3 :
29 3 :
30 3 :
31 3 :
32 3 :
33 3 :
34 3 :
35 3 :
36 3 :
37 3 :
38 3 :
39 3 :
40 3 :
41 3 :
42 1 END XIXFD
43 XIXFD
44 XIXFD
45 XIXFD
46 XIXFD
47 XIXFD
48 XIXFD
49 XIXFD
50 XIXFD
51 XIXFD
52 XIXFD
53 XIXFD
54 XIXFD
55 XIXFD
56 XIXFD
57 XIXFD
58 XIXFD
59 XIXFD
60 XIXFD
61 XIXFD
62 XIXFD
63 XIXFD
64 XIXFD
65 XIXFD
66 XIXFD
67 XIXFD
68 XIXFD
69 XIXFD
70 XIXFD
71 XIXFD
72 XIXFD
73 XIXFD
74 XIXFD
75 XIXFD
76 XIXFD
77 XIXFD
78 XIXFD
79 XIXFD
80 XIXFD
81 XIXFD
82 XIXFD
83 XIXFD
84 XIXFD
85 XIXFD
86 XIXFD
87 XIXFD
88 XIXFD
89 XIXFD
90 XIXFD
91 XIXFD
92 XIXFD
93 XIXFD
94 XIXFD
95 XIXFD
96 XIXFD
97 XIXFD
98 XIXFD
99 XIXFD
100 XIXFD
101 XIXFD
102 XIXFD
103 XIXFD
104 XIXFD
105 XIXFD
106 XIXFD
107 XIXFD
108 XIXFD
109 XIXFD
110 XIXFD
111 XIXFD
112 XIXFD
113 XIXFD
114 XIXFD
115 XIXFD
116 XIXFD
117 XIXFD
118 XIXFD
119 XIXFD
120 XIXFD
121 XIXFD
122 XIXFD
123 XIXFD
124 XIXFD
125 XIXFD
126 XIXFD
127 XIXFD
128 XIXFD
129 XIXFD
130 XIXFD
131 XIXFD
132 XIXFD
133 XIXFD
134 XIXFD
135 XIXFD
136 XIXFD
137 XIXFD
138 XIXFD
139 XIXFD
140 XIXFD
141 XIXFD
142 XIXFD
1844 1 CD0  FORTRAN CALLING PROCEDURE
1845 1 CDG
1846 1 CD0  CALL XILSS (LU, STRING, INDEX)
1847 1 CD0
1848 1 CD0
1849 1 CD0  ********
1850 1 CD0
1851 1 CD0  XILSS IS CALLED TO LIST A SYMBOLIC STRING AND AN INDICATOR TO A
1852 1 CD0  PARTICULAR TOKEN IN THAT STRING
1853 1 CD0
1854 1 CD0  ********
1855 1 CD0
1856 1 CD2  INPUT
1857 1 CD0  LU - LOGICAL UNIT NO. FOR OUTPUT OF STRING
1858 1 CD0  STRING - SYMBOLIC STRING TO BE LISTED
1859 1 CD0  INDEX - SUBSCRIPT INTO STRING OF THE TOKEN TO BE INDICATED
1860 1 CD0
1861 1 CD2
1862 1 CD0  ********
1863 1 CD3
1864 1 CD3  OUTPUT
1865 1 CD3
1866 1 CD3  THE SYMBOLIC STRING IS OUTPUT TO THE LU FOLLOWED BY A LINE CONTAINING
1867 1 CD3  AN INDICATOR (UP ARROW) TO THE DESIGNATED TOKEN.
1 ******
2 NOTES
3 ******
4 USES ROUTINES
5 EXEC
6 XRI6
7 XRE14
8 XRD18
9 XRMG
10 XRMV

10 ******
11 BEGIN XILS
12 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
13 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
14 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
15 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSZ)
16 EXIT TO ERROR 2 IF FIELD = 0
17 IF SIZE < 0, THEN
18 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMBOLIC STRING
19 ENDIF
20 IF TOKSZ < 0, THEN
21 SET TOKSZ TO SIZE + 2
22 ENDIF
23 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
24 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
25 SET BUFFER POINTER TO 1ST POSITION FOR DATA
26 IF INDICATED TOKEN PROCESSED, THEN
27 OUTPUT LINE WITH INDICATOR
28 ENDIF
29 IF FIELD > 0, THEN
30 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
31 ELSE
32 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
33 :SOME: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING
34 AND PUT RESULTS INTO PRINT BUFFER
35 :TWO: CALL XRE14 WITH VALUE IN NEXT 2 WORDS OF SYMBOLIC STRING
36 AND PUT RESULTS INTO PRINT BUFFER
37 :THREE: CALL XRD18 WITH VALUE IN NEXT 3 WORDS OF SYMBOLIC STRING
38 AND PUT RESULTS INTO PRINT BUFFER
39 :FOUR: MOVE THE NEXT 3 WORDS OF SYMBOLIC STRING INTO PRINT BUFFER
40 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMBOLIC STRING
41 :SIX: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMBOLIC STRING AND PUT RESULTS
42 INTO PRINT BUFFER FOLLOWED BY AN "R"
43 :EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
44 INDEX BY 1
45 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
46 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
47 INDICATOR LINE NOT YET OUTPUT, THEN
48
FORTRAN CALLING PROCEDURE

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

1-CONT
2-THREE CONTROL WORDS CONTAINING THE TERMINAL LU AND THE DEBUG CONTROL FLAG. IF CONT(3) BIT 12 IS ON, XUDBG WILL BE CALLED AFTER THE MESSAGE IS OUTPUT
3-NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE A - AREA INDICATOR AS FOLLOWS
4-XX
5-XE
6-XI
7-XX
8-XL
9-DF
10-SC
11-NN - MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'
12-WORKS OF 'SOURCE' TO BE OUTPUT
13-LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED
14-SOURCE (NOT USED IF 'NN' OF 'NUMBER' IS ZERO)
15-LENGTH - INTEGER NUMBER OF WORDS OF 'SOURCE' TO BE INSERTED INTO MESSAGE. ZERO INDICATES NO INSERTION
16-SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED IF 'LENGTH' IS ZERO)
17-OUTPUT
18-UP TO EIGHTY CHARACTER LINE OF TEXT TO UNIT 'LU' OF THE FORM *AANN* MESSAGE(1-LOCATE) SOURCE(1-LENGTH) REMAINDER OF MESSAGE
19-NOTES
20-USES FOR SYSTEM MESSAGE FILE JXIMSG
21-USES CLOSE, EXEC, IAND, KCVT, OPEN, READ, XRMOV, XUDBG
BEGIN XZMSG
SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
SET NUMBER IN PREFIX
READ MESSAGE DIRECTORY RECORD
IF AREA VALID
THEN
SET AREA CODE IN PREFIX
IF MESSAGE NUMBER > 0
THEN
IF VALID MESSAGE NUMBER
THEN
COMPUTE MESSAGE RECORD NUMBER
READ RECORD
CALL XRMOV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
ELSE
EXIT TO :ERROR:
ENDIF
ENDIF
ENDIF
CALL XRMOV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
IF MESSAGE NUMBER > 0
THEN
CALL XRMOV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
ENDIF
ELSE
SET AREA IN PREFIX
ERROR: CALL XRMOV TO MOVE 'XZMSG ERROR' INTO BUFFER
ENDIF
OUTPUT BUFFER TO USER'S TERMINAL
THEN
CALL XUDOG
ENDIF
ENDIF
END XZMSG
BEGIN XOPR
CASE OPERATOR: ADD:, SUBTR:, MUL:, DIVIDE:, INDEX:, SUBSCR:,
UNARY =
EXPO:, MINUS:

PERFORM ADDITION
SET RESULT'S DATA TYPE AS APPROPRIATE

PERFORM SUBTRACTION
SET RESULT'S DATA TYPE AS APPROPRIATE

PERFORM MULTIPLICATION
SET RESULT'S DATA TYPE AS APPROPRIATE

PERFORM DIVISION
SET RESULT'S DATA TYPE AS APPROPRIATE

IF RESULT STACK IS NOT EMPTY, THEN
GET 1-DIM FOR TOP OPERAND IN RESULT STACK
SET # WORDS PER ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK
ELSE
EGGLE SUBSCRIPT FOR OBJECT IS BEING EVALUATED
GET 1-DIM FOR FIRST ENTRY IN SYMBOL TABLE
SET # WORDS PER ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
ENDIF

CALCULATE RESULT (i.e., INDEX = IDIM*(SECOND OPERAND-1)+FIRST OPERAND)

IF RESULT STACK IS NOT EMPTY, THEN
SET # WORDS/ELEMENT BASED ON TOP ENTRY OF RESULT STACK ENTRY
ELSE
OBJECT IS BEING SUBSCRIPTED
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 XPROC TO RETRIEVE A LOGICAL VALUE FOR TOP OPERAND AT DISPLACEMENT
DECREMENT RESULT STACK POINTER TO DISCARD ARRAY NAME
ELSE
OBJECT IS BEING SUBSCRIPTED
SET TYPE = -2
ENDIF

PERFORM EXPONENTIATION
SET RESULT'S DATA TYPE AS APPROPRIATE

CHANGE SIGN OF OPERAND FOR RESULT
ENDCASE

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

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

CALL XPCS(TARGET, OPMUN)

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

INPUTS

TARGET - DESIRED FDS FIXED DATA TYPE
OPMUN - OPERAND NUMBER FOR ENTRY CURRENTLY BEING SET UP
FROM ASGCOM - L,U, SSTRM, RESULT, RSLTP,T, DATYP

OUTPUTS TO ASGCOM
RSLTP,T, OPMUN

EXTERNAL REFERENCES
FDS - XPXIT, XMMOV, X2LSS, X2MS
RTE - COLE, FLOST, IFIX, OVF, SNGL
1 BEGIN XIPCS
2  POP OPERAND AND DATA TYPE FROM TOP ENTRY OF RESULT STACK
3  IF TARGET TYPE AND DATA TYPE ARE NOT EQUAL, THEN
4       CASE (TARGET TYPE) :INTG;, :REAL;, :DBL:
5       :INTG:
6       CONVERT OPERAND TO INTEGER
7       :REAL:
8       CONVERT OPERAND TO SINGLE PRECISION REAL
9       :DBL:
10      CONVERT OPERAND TO DOUBLE PRECISION REAL
11     ENDCASE
12    ENDF
13     IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
14     STORE OPERAND AND CURRENT DATA TYPE FOR USE BY FUNCTION OR ARITHMETIC OPER.
15   EXIT XIPCS
16   2:OVER:
17   2  SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
18   2  CALL XMSG TO OUTPUT MESSAGE TO USER
19   2  CALL XLESS TO LIST SYMBOLIC STRING
20   2  CALL XPXIT TO EXIT PROCESSOR
21   1  END XIPCS
2222 1 CD0  FORTRAN CALLING PROCEDURE:
2224 1 CD0  CALL XIPS1
2225 1 CD0  -----------------------------------------
2227 1 CD1  XIPS1 IS USED BY ASGNC TO TRANSLATE THE SYMBOLIC STRING EXPRESSION
2228 1 CD1  TO A POST-FIX NOTATION STRING
2230 1 CD1  -----------------------------------------
2232 1 CD2  INPUTS FROM ASGNC
2234 1 CD2  LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTBL
2236 1 CD2  -----------------------------------------
2238 1 CD3  OUTPUTS TO ASGNC
2240 1 CD3  EXPTR, RNGSTK, POLISH, SYNTAX
2242 1 CD3  -----------------------------------------
2244 1 CD4  INTERNAL VARIABLES
2246 1 CD4  GRPSK - 2X120 ARRAY USED TO TRACK FUNCTIONS, SUBSCRIPTS, AND
2248 1 CD4  PARENTHETICAL GROUPINGS
2249 1 CD4  OPSTK - 2X120 ARRAY; EACH ENTRY CONTAINS OPERATOR TOKEN AND ITS OUTPUT
2250 1 CD4  PRIORITY
2251 1 CD4  PCRT - PRECEDNT TYPE FOR PRECEDING CHARACTER ; USED FOR SYNTAX CHECK
2252 1 CD4  TKPRT - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING
2253 1 CD4  TOKEN - TOKEN CURRENTLY BEING PROCESSED
2254 1 CD4  -----------------------------------------
2255 1 CD5  --------------------------
2256 1 CD5  --------------------------
1 BEGIN XIPS1
2 DO FOR EACH TOKEN UNTIL ; OR ESS IS REACHED
3 PERFORM SETUP FOR SPECIAL HANDLING FOR NUMBERS, NAMES, +, *, (, AND "
4 PERFORM STRING FOR SYNTAX CHECKING AND POLISH STRING BUILD
5 END DO
6 PERFORM RANGE TO PROCESS RANGE DEFINITION, IF NECESSARY
7 END XIPS1
BEGIN SETUP

CASE TOKEN

:INVLDA: :SYMBOL: :SYMBOL: :SYMBOL:

NAME CHARACTER + -


/ < >


= ( }


:ENDCASE:

:SYMBOL:

IF TOKEN IS A NAME AND NEXT TOKEN IS A LEFT BRKT, THEN

STARTSEARCH UNTIL FUNCTION TABLE IS CHECKED

EXIT IF TOKEN NAME MATCHES FUNCTION NAME

CHANGE TOKEN TO FUNCTION'S INDEX IN TABLE + 128

PUSH COUNT=0=BINARY/UNARY FLAG AND LIMIT=0 ONTO GROUPING STACK

END LOOP

EXIT IF NO MATCH IS FOUND

ELSE TOKEN IS AN OPERAND

IF TOKEN IS A NAME AND NEXT TOKEN IS (, THEN

RET 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

INCUMENT TO NEXT TOKEN AND EXIT TO :END:

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 EXPONENITATION

ENDIF

:LPAREN:

IF PRECEDING TOKEN WAS A VARIABLE (DATA ELEMENT NAME), THEN

PUSH COUNT AND LIMIT ALLOWING DOUBLY SUBSCRIPTING ONTO EXPRESSION STACK

ELSE EXPRESSION IS BEING PROCESSED

ENDIF

ENDIF

ENDCASE
1 BEGIN STRING
2 SET INDEX INTO SYNTAX TABLE TO MIN(TOKEN, 40)
3 ERREXIT TO :SYNTX1: IF THIS TOKEN IS NOT VALID ACCORDING TO SYNTAX TABLE
4 IF TOKEN IS NOT AN OPERAND, THEN
5 DO UNTIL TOKEN IS PUSHED ONTO OPERATOR STACK OR DISCARDED
6 IF INPUT PRIORITY OF THIS TOKEN > OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR
7 STACK, THEN
8 IF TOKEN IS #, THEN
9 INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK
10 ERREXIT TO :SYNTX1: IF COUNT > COMMA LIMIT FOR (TOP ENTRY IN GROUPING STK
11 IF COMMA LIMIT SHOWN SUBSCRIBING IN F...GRESS (LIMIT > 0) OR, THEN
12 PUSH TOKEN AND OUTPUT PRIORITY ONTO OPERATOR STACK
13 ELSE
14 DISCARD > FUNCTION LIST IS BEING PROCESSED
15 ENDF
16 ELSE OPERATOR IS NOT #,
17 PUSH TOKEN AND ITS OUTPUT PRIORITY ONTO OPERATOR STACK
18 ENDF
19 ELSE INPUT PRIORITY IS < OR = OUTPUT PRIORITY
20 IF INPUT PRIORITY < OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR STACK OR
21 INPUT PRIORITY = OUTPUT PRIORITY NOT = 2, THEN
22 ERREXIT TO :SYNTX1: IF INPUT PRIORITY = 0 AND OPERATOR STACK IS EMPTY
23 IF TOP ENTRY OF OPERATOR STACK IS #, THEN
24 ERREXIT IF THIS IS NOT LAST ENTRY ON OPERATOR STACK TO :BADEOL:
25 ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTX2:
26 DISCARD TOKEN (# OR LESS)
27 ENDF
28 POP OPERATOR STACK
29 PUSH OPERATOR ONTO EXPRESSION STACK
30 ELSE BRACKETS OR PARENS HVE BEEN MATCHED
31 IF TOP ENTRY OF OPERATOR STACK IS ( THEN
32 ERREXIT TO :SYNTX2:
33 IF TOP ENTRY OF GROUPING STACK INDICATES SUBSCRIBING(LIMIT>0), THEN
34 PUSH SUBSCRIBING OPERATOR ONTO EXPRESSION ARRAY
35 ENDF
36 ELSE TOP OPERATOR ENTRY IS OPEN BRACKET (FUNCTION LIST)
37 ERREXIT .F CURRENT TOKEN IS NOT CLOSE BRACKET TO :SYNTX2:
38 ERREXIT TO :FLSTK: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GRPING STK COUNT=0)
39 ENDF
40 POP OPERATOR STACK
41 POP GROUPING STACK
42 DISCARD CURRENT TOKEN
43 ENDF
44 ENDIF
45 END
46 ENDF
47 :ENDO:
48 INCREMENT TO NEXT TOKEN USING TOKEN LENGTH FROM SYNTAX TABLE
49 EXIT STRING
50 :SYNTX1:
51 SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"
52 :SYNTX2:
53 SET MESSAGE TO BE OUTPUT TO "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY!"
54 :COMERR:
55 SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
3591  SET MESSAGE TO BE OUTPUT TO "INCOMPLETE FUNCTION LIST"
3592  CALL XNS6 TO OUTPUT MESSAGE TO USER
3595  CALL XNS6 TO LIST PROCESSOR STRING
3598  END STRING
FORTRAN CALLING PROCEDURE:

CALL XIPS2

XIPS2 is used by ASSGN to evaluate the post-fix string generated
by XIPS1 and to store the results into the object data element

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

INPUTS FROM ASGCOM

LU, TOKENS, LASTY, SYMTAB, SSTRNG, RESULT, ORPRND

OUTPUT TO ASGCOM

SYMTAB, RESULT, RE88, RSLTP1, OPINFO

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

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

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

EXTERNAL REFERENCES

FDB - XPREU, XPRIT, XROMV, XZCNR, XZFMU, XZFPD, XZFD, XZLSS, XZMSU,
XZOPU, XZPS, XZRET

RTE - IABS, IAND, MAXB
2465 1 BEGIN XIPS2
2466 2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
2467 2 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
2468 2 INITIALIZE RANGE VALUES FOR ITERATION
2469 3 DO UNTIL ALL RANGES ARE FINISHED
2470 3 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
2471 4 POP ENTRY FROM POLISH STRING
2472 4 IF ENTRY IS AN OPERAND, THEN
2473 4 PERFORM RPUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH INTO RESULT STACK
2474 4 ELSE ENTRY IS AN OPERATOR
2475 5 IF OPERATOR IS NOT "+", THEN
2476 5 PERFORM EEXVAL TO EVALUATE POLISH STRING
2477 5 ELSE OPERATOR IS =
2478 6 PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
2479 6 ENDEF
2480 4 ENDIF
2481 3 END DO
2482 3 PERFORM RNSSET TO DETERMINE CORRECT RANGE VALUES
2483 2 END DO
2484 1 END XIPS2
BEGIN TOC
DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
IF SYMBOL NOT = 0 (SYMBOL IS DE OR INDEX), THEN
IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN
CALL XPREG TO RETRIEVE TOC ENTRY
ELSE SYMBOL IS A RANGE INDEX
SET DATA TYPE TO INTEGER
ENDIF
ENDIF
END DO
END XPREG WITH A CLOSE BUFFER REQUEST
END TOC
BEGIN RPUSH
1. IF OPERAND IS A CHARACTER STRING POINTER (OPERAND < 0), THEN
2. PUSH ABSTRACT VALUE OF OPERAND AND -3 DATA TYPE ONTO RESULT STACK
3. ELSE IF OPERAND IS A SYMBOL TABLE INDEX
4. CALL SIZE TO RETRIEVE DATA AND DATA TYPE FOR OPERAND
5. PUSH RETRIEVED VALUE AND DATA TYPE ONTO RESULT STACK
6. END IF
7. END RPUSH
2517 1 BEGIN EVAL
2518 2 IF OPERATOR IS A FUNCTION, THEN
2519 3 DETERMINE & OPERANDS FROM FUNCTION TABLE
2520 4 ELSE
2521 5 DETERMINE & OPERANDS FROM SYNTAX TABLE
2522 6 ENDSF
2523 7 DO FOR EACH OPERAND
2524 8 IF RESULT STACK CONTAINS A SYMBOL INDEX, THEN
2525 9 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :RXERR:
2526 10 CALL XSPS TO RETRIEVE DATA AND "PE FOR FIX" ELEMENT OF ARRAY
2527 11 STORE DATA AND TYPE IN RESULT STACK AT THIS ENTRY
2528 12 ELSE
2529 13 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :RXERR:
2530 14 ENDSF
2531 15 END DO
2532 16 IF THE OPERATOR IS A FUNCTION, THEN
2533 17 DO FOR EACH FUNCTION OPERAND
2534 18 SET TARGET TYPE FROM FUNCTION TABLE
2535 19 CALL XIPCS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
2536 20 END DO
2537 21 CALL XIFHC TO PERFORM FUNCTION FOR RESULTS
2538 22 IF DATA TYPE FOR RESULTS FROM FUNCTION TABLE
2539 23 ELSE THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "="
2540 24 DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
2541 25 CASE (TYPE REQUIREMENT+1) :SAME:, :INTEGER:
2542 26 4 :SAME:
2543 27 5 IF & OPERANDS > 1, THEN
2544 28 6 SET TARGET TYPE TO MAX OF TWO DATA TYPES
2545 29 ELSE & OPERANDS = 1
2546 30 7 SET TARGET TYPE TO OPERAND'S DATA TYPE
2547 31 ENDSF
2548 32 :INTEGER:
2549 33 :SET TARGET TYPE TO INTEGER
2550 34 ENDCASE
2551 35 DO FOR EACH OPERAND
2552 36 CALL XIPCS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
2553 37 END DO
2554 38 CALL XIPMR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
2555 39 ENDIF
2556 1 EXIT EVAL
2557 1 ERREXIT:
2558 2 SET MESSAGE TO BE OUTPUT TO "FREE OR CHARACTER DATA ELEMENT FOUND IN AN
2559 3 " EXPRESSION"
2560 4 CALL XIPSM TO OUTPUT MESSAGE TO USER
2561 5 CALL XIILST TO LIST SYMBOLIC STRING
2562 6 CALL XPAR TO EXIT PROCESSOR
2563 7 END EVAL
BEGIN REPLAC
IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
ENDIF
CASE (OBJECT DATA TYPE ) :
FREE; :FIXED; :FIXED; :CHAR; :CHAR;
1 2 3 4 5 6 7 8 CHAR; :CHAR; :CHAR;
:FREE; :FIXED; :FIXED; :CHAR; :CHAR;
CALL XPFR TO RETRIEVE DATA AND SET UP FOR STORE
CALL FIXOBJ TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
CHAR;
CHAR;
CHAR;
CHAR;
END CASE
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
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
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 - IAND**

**IAND**

**CALL**
FORTRAN CALLING PROCEDURE

CALL XSYM (TKMPTR, FLAG, SYMIND)

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

INPUT

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

FLAG - SPECIAL PROCESSING FLAG

NORMAL PROCESSING

SYMBOL IS A RANGE

SYMBOL IS SUBSCRIPTED

ASC.COM COMMON - LU, SSTRING, STWIDE, SYMTAB, TOKENS,

OUTPUT

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

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

ASC.COM COMMON - SYMTAB

INTERNAL

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

EXTERNAL REFERENCES

XPIT, XILSS, XIMSG, XISM


<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2687</td>
<td>BEGIN XSYM</td>
</tr>
<tr>
<td>2688</td>
<td>CALL XSYM TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY</td>
</tr>
<tr>
<td>2689</td>
<td>IF TOKEN IS A NAME, THEN</td>
</tr>
<tr>
<td>2690</td>
<td>IF SPECIAL PROCESSING FLAG IS SET (1=RANGE INDEX, 2=SUBSCRIPTED), THEN</td>
</tr>
<tr>
<td>2691</td>
<td>IF FLAG INDICATES RANGE INDEX, THEN</td>
</tr>
<tr>
<td>2692</td>
<td>ENEXIT TO :BADRTG: WITH ERROR AS01 IF OBJECT (FIRST ENTRY IN TABLE)</td>
</tr>
<tr>
<td>2693</td>
<td>ERREXIT TO :BADRTG: WITH ERROR AS02 ENTRY IS ALREADY SUBSCRIBED</td>
</tr>
<tr>
<td>2694</td>
<td>ERREXIT TO :BADRTG: WITH ERROR AS03 ENTRY IS ALREADY A DEFINED RANGE</td>
</tr>
<tr>
<td>2695</td>
<td>ENDEF</td>
</tr>
<tr>
<td>2696</td>
<td>SET ENTRY FLAG WORD TO FLAG VALUE</td>
</tr>
<tr>
<td>2697</td>
<td>ENDF</td>
</tr>
<tr>
<td>2698</td>
<td>ENDF</td>
</tr>
<tr>
<td>2699</td>
<td>ADD BIAS OF 256 TO SYMBOL INDEX</td>
</tr>
<tr>
<td>2700</td>
<td>EXIT XSYM</td>
</tr>
<tr>
<td>2701</td>
<td>:BADRTG:</td>
</tr>
<tr>
<td>2702</td>
<td>CALL XRING TO OUTPUT ERROR DESCRIPTION</td>
</tr>
<tr>
<td>2703</td>
<td>CALL XSS TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR</td>
</tr>
<tr>
<td>2704</td>
<td>CALL EXIT TO TERMINATE PROCESSOR</td>
</tr>
<tr>
<td>2705</td>
<td>END XSYM</td>
</tr>
</tbody>
</table>
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
a symbolic string

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

ASYCMOM - LASTSY, SSTRNG, STMIDE, SYMtab, tokens

OUTPUT
SYMIND - index into symbol table (SYMtab) where symbol is located.
VALUE OF SYMIND INDICATES ENTRY NUMBER, I.e., 1, 2, ...
ASYCMOM - LASTSY, SYMtab

INTERNAL
DISP - DISPLACEMENT INTO SYMBOL TABLE ENTRY FOR FIELD TO BE USED
AS KEY. NAMES USE A VALUE OF 1, CONSTANTS A VALUE OF 0.
EDLOOP - INDEX OF LAST ALLOCATED WORD IN SYMtab. VALUE IS EQUAL
to LASTSY*SIMUM.
IBUF - INTERNAL BUFFER FOR SYMBOL VALUE USED TO ASSURE LAST
WORDS OF INTEGER AND REAL CONSTANTS ARE ZERO.

EXTERNAL REFERENCES
XRCPR, XXNOW

SPECIAL REMARKS
THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS

I TYPE (1 WD) I NAME (4 WD) I RESERVED FOR SYMIND I VALUE (1 WD) I OTHER I
BEGIN XISTT

CLEAR BUFFER TO BE USED IN MOVING TOKEN

IF TOKEN IS A NAME, THEN
  SET COMPARISON DISPLACEMENT IN TABLE TO 1 (NAME FIELD)
ELSE
  SET COMPARISON DISPLACEMENT IN TABLE TO 2 (VALUE FIELD)
ENDIF

MOVE TOKEN INTO BUFFER

START SEARCH UNTIL ALL ALLOCATED SYMBOL TABLE ENTRIES EXAMINED

EXIT IF ENTRY MATCHES BUFFER CONTENTS AND TYPE FIELD MATCHES TOKEN CODE

END LOOP

STORE TOKEN CODE IN TYPE FIELD OF NEXT ENTRY

STORE BUFFER CONTENTS INTO APPROPRIATE FIELD OF ENTRY (NAME OR VALUE)

INCREMENT NUMBER OF ALLOCATED ENTRIES

END SEARCH

SET STRMIND TO ENTRY NUMBER

END XISTT
DO - CONDITIONAL ITERATION (LOOPING) PROCESSOR

SCHEDULED BY FBS

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

INPUT FROM MANAGER VIA SCHEDULING PARAMETERS

LU - LOGICAL UNIT OF USER'S TERMINAL

FLAGS - DEBUG FLAGS FROM USER SIGN-ON

SENDSP - INDEX OF CURRENTLY EXECUTING USER TB ENTRY

INPUT FROM INTERFACE TABLE

DOTYPE - 4CH CHARACTER STRING SPECIFYING LOOP CONDITION TYPE

UNTIL - LOOP UNTIL RELATION BETWEEN OPRND1 AND OPRND2 IS TRUE. TEST IS MADE AT THE END OF EACH LOOP EXECUTION.

WHILE - LOOP WHILE RELATION BETWEEN OPRND1 AND OPRND2 IS TRUE. TEST IS MADE PRIOR TO STARTING EACH LOOP EXECUTION.

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

RELATN - 2CH CHARACTER STRING CONTAINING RELATIONAL SYMBOL

# - NOT EQUAL

< - LESS THAN

<= OR < - LESS THAN OR EQUAL

= - EQUAL

> OR > - GREATER THAN OR EQUAL

> - GREATER THAN

OPRND2 - SECOND REAL QUANTITY TO COMPARE (SEE OPRND1)

INPUT FROM AVA ON REQUEST TO MANAGER

INTBUF - INTERFACE TABLE (LESS LITERAL AREA)

LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE LITERAL AREA

BSSTK - SEE OUTPUT DEFINITION

BSRTB - 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 - $FAMCE RESET NUMBER OF ENDDO IF RETURN = 3

INPUT/OUTPUT FROM/TOP AVA

BSSTK - 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 SRD, VOLUME IV, FIGURE 1.2-17)

WORDS 8-14 = INTERFACE TABLE ENTRY FOR OPM01 (SAME AS

IN ORIGINAL TABLE EXCEPT FOR POSSIBLE NEW

VALUES POINTING INTO LITERAL AREA)

WORDS 15-21 = INTERFACE TABLE ENTRY FOR OPM02 (SEE

ABOVE)

WORDS 22-25 = LITERAL AREA FOR VALUE(S)/SUBSCRIPT(S) OF

OPRN01 & 2

WORD 26 - RELATION ID CODE AS FOLLOWS

0 - <

1 - >

2 - >= OR >=

3 - =

4 - <= OR <=

5 - <

WORD 27 - SEQUENCE NUMBER OF TOP OF LOOP

1 CD3

C04 INTERNAL VARIABLES

CD4 CODE - ARRAY OF EIGHT ACCEPTABLE RELATION PHENOMICS AND

04 CORRESPONDING INTERNAL CODES

07 CD4 ENTRY INTO WWIDFXML FOR NEW 27 WORD ENTRY

08 CD4 PMDFEL - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD

14 ENTRIES ARE OF THE FORM

0 RIST - AMA MANAGEMENT REQUEST CODE

0 CLASS - CLASS AND TYPE OF DATA

0 NAME - SIX CHARACTER DATA NAME

0 SIZE - SIZE OF DATA

0 DISP - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN

0 CLASH - CLASH I/O NUMBER THROUGH WHICH DATA IS

8 TRANSMITTED

97 CD4 TOP - INDEX INTO SJERT OF TOP OF LOOP

98 CD4 XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS (SET BY

99 CD4 XPGET)

100 CD4 101 CD4

02 CD5 REFERENCED ROUTINES

103 CD5 EXEC, IAHZ, MOD, XPRAP, XPRAP, XPGET, XPGET, XPSET, XPRAP, XPRAP, XPRAP

105 CD5 XRSET, XUPAM, XUPAM, XUPAM

106 CD5 107 CD5 NOTES

108 CD5 DO AND ENDDO MUST BE USED IN PAIRS

109 CD5 110 CD5

111 CD5 WWIDXML IS ALLOCATED IN THE AMA FOR INTERPROCESSOR COMMUNICATIONS

112 CD5 EZSCH CLEANS UP ANY RESIDUAL WWIDXML AFTER EXECUTION CONTROLLER

113 CD5 TEMPNATION

114 CD5 115 CD5 THE MAXIMUM NUMBER OF NESTED LOOPS IS 4

116 CD5 117 CD5 ALL NESTING (STRUCTURE) ERRORS ARE LEFT FOR EXECUTION TIME

118 CD5 119 CD5 DETECTION
FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

THE ELSE UTILITY PROCESSOR LOCATES THE ENDBF CORRESPONDING TO
THE ELSE AMD SETS THE SEQUENCE RESET NUMBER VIA XPIIT) SO
THE PROCESSOR FOLLOWING ENDBF IS EXECUTED NEXT IN THE SEQUENCE.

INPUT
SCHEDULING PARAMETERS
L0 U - LOGICAL UNIT NO. OF THIS FDS USER
FLAGS - DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT

OUTPUT
RETURN PARAMETERS SENT TO FDS MANAGER VIA XPIIT
(1) = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE
NO. GIVEN IN RPARMS(2)
(2) = 8 => ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE
(2) = SEQUENCE NO. TO BE EXECUTED NEXT WHEN RPARMS(1) = 3

REFERENCE ROUTINES

CD5 RPAR
CD5 XPIIT
CD5 XREX
CD5 XUDDE
CD5 XIFDL
CD5 XISCM
CD5 CD5
BEGIN ELSE
2 CALL RMPAR TO GET SCHEDULING PARAMETERS
3 CALL XICFL TO ESTABLISH FDS MANAGER'S CLASS NO. (XPCLS)
4 CALL XISCN TO SEARCH FOR MATCHING ENDPF COMMAND
5 CALL XPIT TO RETURN PARAMETERS TO FDS MANAGER
6 END ELSE
ENDDO

DO - LOOP TERMINATION PROCESSOR

SCHEDULED BY FDS

THE DO AND ENDDO UTILITY PROCESSOR PAIR PROVIDE FDS USERS WITH THE
CAPABILITY OF LOPPING 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 ACCESSED FOR THE MANAGER
PROCESSOR CLASS I/O COMMUNICATIONS NUMBER

OUTPUT TO THE MANAGER VIA RETURN PARAMETERS

RETURN - RETURN CODE TO MANAGER

0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL

EXECUTION

3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE

NUMBER FOR CONTINUED EXECUTION

B - 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

REG - AW MANAGEMENT REQUEST CODE

CLAS - CLASS AND TYPE OF DATA

NAME - SIX CHARACTER DATA NAME

SIZE - SIZE OF DATA

disp - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN

CLASS - CLASS I/O NUMBER THROUGH WHICH DATA IS

TRANSMITTED

XPLCS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

REFERENCE ROUTINES

EXEC, IANH, XXRAP, XGET, XPREQ(XPGET), XPXIT, XRLOC, XRMOV,

XUDBG, XVPW, XIMSG

ENDDO

ENDDO
BEGIN

1. FIND CLASS USING LAST STOCK ENTRY AS KEY
2. IF REVERSE HAS NUMBER OF FOUND, THEN
   3. MAKE NUMBER OF STOCK ENTRY REVERSE
5. OTHERWISE
   6. IF LAST STOCK ENTRY FROM FIRST CLASS BUFFER
8. THEN
   9. CALL $XGET TO RESTORE BUFFER, THEN
10. CALL $XGET TO EXAMINE RELATION
11. CALL $XGET TO EVALUATE RELATION
12. IF RELATION $XGET TRUE, THEN
13. WRITE RESULT OF $XGET TO OUTPUT
14. ELSE IF RELATION $XGET FALSE, THEN
15. WRITE Result of $XGET TO OUTPUT
16. ELSE
17. WRITE Result of $XGET TO OUTPUT
18. ENDIF
19. END
20. ENDIF
21. END
22. END
FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

THE ENDF 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

RPARMS - return parameters for FDS Manager via XPXIT

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

ROUTINES USED

RMPAR

XREXT

XPXIT

XUTEB

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

- **CD0**
  - IF UTILITY PROCESSOR COMPARSES TWO INPUT QUANTITIES ACCORDING TO A SPECIFIED RELATION. WHEN THE COMPARISON IS FALSE, THE SEQUENCE RESET NUMBER IS SET (VIA XPXIT) TO THE PROCESSOR FOLLOWING THE CORRESPONDING ELSE (OR ENDIF). IF THE COMPARISON IS TRUE THE NORMAL PROCESSOR EXECUTION SEQUENCE IS FOLLOWED.
  - **CD0**
  - **CD0**
  - **CD0**
  - **CD0**

**INPUT SCHEDULING PARAMETERS**
- **CD0**
  - **CD0**
  - **CD0**
  - **CD0**

**INTERFACE TABLE PARAMETERS**
- **CD0**
  - **CD0**
  - **CD0**
  - **CD0**

**OUTPUT**
- **CD0**
  - **CD0**
  - **CD0**
  - **CD0**

**RPARMS** - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPXIT
- **CD0**
  - **CD0**
  - **CD0**
  - **CD0**

**REFERENCE ROUTINES**
- **CD0**
  - **CD0**
  - **CD0**
  - **CD0**
1 BEGIN IF
2 CALL IMPAR TO GET INPUT (SHEPLING) 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 ERROR IF RELATIONAL OPERATOR INVALID :ERROR:
6 CALL XIEML 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 ERROR: CALL XIMSG 'INVALID RELATIONAL OPERATOR - MUST BE <,>,=,<,<=,>=,OR >'
15 END IF
FORTAN CALLING PROCEDURE

LVALUE = XIEVL (OPRND1, RELTH, OPRND2)

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

INPUT
OPRND1 - FIRST REAL VALUE TO COMPARE
OPRND2 - SECOND REAL VALUE TO COMPARE
RELTH - RELATION CODE AS FOLLOWS
0 - NOT EQUAL
1 - GREATER THAN
2 - GREATER THAN OR EQUAL
3 - EQUAL
4 - LESS THAN OR EQUAL
5 - LESS THAN

OUTPUT
XIEVL - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

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

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

THEREFORE FOR OPRND1 - OPRND2 POSITIVE (BITS 0-2) 1, 1, 1
FOR OPRND1 - OPRND2 ZERO (BITS 3-5) 0, 0, 1
AND FOR OPRND1 - OPRND2 NEGATIVE (BIT 6-8) 1, 0, 0
(BITS 9-15 ARE NOT USED) IF RELTH IS 0, 0, 0 OR THE
COMPLEMENT IF RELTH IS 3, 4, OR 5

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES

REFERENCES
REFERENCES
BEGIN X:EVL
CASE (+, 0, --) DIFFERENCE OF OPRAND1 AND OPRAND2

1: SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)

0: SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)

-: 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 ZX9CH (SRCFG, RPAMS)**

**Z9CH 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**

**SRCFG = FLAG INDICATING ORIGINATION OF THIS CALL**

- 0 => CALLED BY IF TO FIND MATCHING ELSE OR ENDF
- 1 => CALLED BY ELSE TO FIND MATCHING ENDF

**COMMON**

**DEBUG = FLAG INDICATING WHETHER ONLINE DEBUG TO BE OUTPUT**

- 0 => NO DEBUG
- > 0 => DEBUG

**PARMS = SCHEDULING PARAMETERS FROM THE FBS MANAGER**

- 1 = LOGICAL UNIT NO. OF THE FBS USER
- 2 = SEQNO, SEQUENCE OF COMMAND

**OUTPUTS**

**CALLING SEQUENCE**

**RPAMS = PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XPII**

- 1 = CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
- 2 = ABnormally TERMINATE PROCESSOR EXECUTION SEQUENCE

**Routines Used**

**EXEC**

**MPAR**

**XPIA**

**XUDG**

**XRCPB**

**XMPG**
1 BEGIN XZCH
2 RETRIEVE &SECTAB FROM THE AWA USING XVPAM
3 STARTSEARCH UNTIL ALL COMMANDS IN &SECTAB
4 EXIT IF CURRENT COMMAND IS FOUND
5 SET NUMBER OF IF NESTS TO 1
6 STARTSEARCH FROM NEXT COMMAND IN &SECTAB UNTIL ALL FOLLOWING COMMANDS
7 IF COMMAND IS FOR ENDFPROCESSOR, THEN
8 DECREMENT NUMBER OF IF NESTS BY 1
9 ELSE
10 IF COMMAND IS FOR IF PROCESSOR, THEN
11 INCREMENT NUMBER OF IF NESTS BY 1
12 ELSE
13 IF CALLED BY IF PROCESSOR, AND
14 COMMAND IS FOR ELSE PROCESSOR, THEN
15 ERREXIT IF THIS IS THE END OF &SECTAB :ERROR1:
16 IF NUMBER OF IF NESTS IS 1, THEN
17 DECREMENT NUMBER OF IF NESTS TO 0
18 ENDF
19 ENDF
20 EXIT IF NUMBER OF IF NESTS IS 0
21 SET SEQUENCE RESET NUMBER (RPARMS(2)) TO SEQUENCE NUMBER OF THE
22 NEXT COMMAND IN THE TABLE
23 ENDOLOOP
24 ERREXIT :ERROR1:
25 ENDS
26 ENDS
27 EXIT
28 1 EXIT XZCH
29 :ERROR1: CALL XZMSG - 'IF CANNOT BE EXECUTED WITHOUT MATCHING ENDF'
30 ERROR4: CALL XMSG - 'SYSTEM ERROR - NO &SECTAB'
31 END XZCH
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
<table>
<thead>
<tr>
<th>NAME</th>
<th>VERSION</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ</th>
<th>SIZE-PRE-TEXT</th>
<th>(CYCLE WORD)</th>
<th>PSMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>53</td>
<td>3 0 1</td>
<td>1792</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:56:07</td>
<td>2</td>
<td>60</td>
<td>3 0 1</td>
<td>1843</td>
<td></td>
</tr>
<tr>
<td>XUPDL</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:59:32</td>
<td>3</td>
<td>53</td>
<td>3 0 1</td>
<td>1875</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>05:37:22</td>
<td>4</td>
<td>84</td>
<td>3 0 1</td>
<td>1932</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>3 0 2</td>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>3 1 2</td>
<td>2103</td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td>ELT SYMB</td>
<td>10 NOV 77</td>
<td>06:19:22</td>
<td>9</td>
<td>394</td>
<td>3 2 3</td>
<td>2105</td>
<td>SET</td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>23:22:28</td>
<td>10</td>
<td>167</td>
<td>3 6 5</td>
<td>2345</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:00:28</td>
<td>11</td>
<td>373</td>
<td>3 8 5</td>
<td>2923</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>15 FEB 78</td>
<td>19:17:22</td>
<td>12</td>
<td>824</td>
<td>3 9 5</td>
<td>3266</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>12:48:36</td>
<td>13</td>
<td>627</td>
<td>3 14 5</td>
<td>4120</td>
<td></td>
</tr>
<tr>
<td>XV</td>
<td></td>
<td>ELT SYMB</td>
<td>22 FEB 78</td>
<td>23:50:36</td>
<td>14</td>
<td>320</td>
<td>3 20 5</td>
<td>4747</td>
<td></td>
</tr>
<tr>
<td>COMMON</td>
<td></td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>22:20:04</td>
<td>15</td>
<td>200</td>
<td>3 20 5</td>
<td>4607</td>
<td></td>
</tr>
<tr>
<td>POLIST</td>
<td></td>
<td>ELT SYMB</td>
<td>11 MAR 78</td>
<td>03:16:51</td>
<td>16</td>
<td>2</td>
<td>3 19</td>
<td>5067</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>11 MAR 78</td>
<td>03:17:36</td>
<td>17</td>
<td>1</td>
<td>3 10 5</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>18</td>
<td>501</td>
<td>3 10 5</td>
<td>5094</td>
<td></td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:52</td>
<td>19</td>
<td>534</td>
<td>3 22 5</td>
<td>5195</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>11:43:48</td>
<td>20</td>
<td>384</td>
<td>3 6 5</td>
<td>5729</td>
<td></td>
</tr>
<tr>
<td>XE</td>
<td></td>
<td>ELT SYMB</td>
<td>18 MAY 78</td>
<td>09:12:58</td>
<td>21</td>
<td>1548</td>
<td>3 4 5</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>22</td>
<td>403</td>
<td>3 20 5</td>
<td>7661</td>
<td></td>
</tr>
<tr>
<td>XU</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:39</td>
<td>23</td>
<td>504</td>
<td>3 20 5</td>
<td>8064</td>
<td></td>
</tr>
<tr>
<td>ERECF</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>25</td>
<td>118</td>
<td>3 13 5</td>
<td>8723</td>
<td></td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>26</td>
<td>1201</td>
<td>3 21 5</td>
<td>8841</td>
<td></td>
</tr>
<tr>
<td>XN</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:39</td>
<td>27</td>
<td>600</td>
<td>3 29 5</td>
<td>10042</td>
<td></td>
</tr>
<tr>
<td>XZ</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:57</td>
<td>28</td>
<td>326</td>
<td>3 2 3</td>
<td>10642</td>
<td></td>
</tr>
</tbody>
</table>

**NEXT AVAILABLE LOCATION**

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

**EXIT F.POLIST**