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.
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>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2.0</td>
<td>COMMON FOR FDS EXECUTIVE</td>
<td>2-1</td>
</tr>
<tr>
<td>3.0</td>
<td>FDS EXECUTIVE MESSAGES</td>
<td>3-1</td>
</tr>
<tr>
<td>4.0</td>
<td>PDL LISTING PROGRAM</td>
<td>4-1</td>
</tr>
<tr>
<td>5.0</td>
<td>FDS EXECUTIVE DETAILED LOGIC FLOW</td>
<td>5-1</td>
</tr>
<tr>
<td>6.0</td>
<td>DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION</td>
<td>6-1</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

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

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

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

TO FOS Routines EXCEPT XE,...)

******

XE CONTENTS

INTEGER
  # CARTPG
  # CLASMD
  # COMPR
  # FLAGS
  # PCOMNAME(3)
  # REXBUF(64)
  # SEREND
  # SUBSTA
  # TKMLNG
  # TOKENS(32)
  # XE
  # XE

DIMENSION
  # INTNAM(3)

EQUIVALENCE
  # XE(1), ALU
  # XE(3), RUAL
  # XE(5), MASSTA
  # XE(7), SEGNAM(11)
  # XE(11), SEREND
  # XE(13), INTNAM(3)
  # XE(19), REXBUF(1)
  # XE(83), TKMLNG
  # XE(85), TOKENS(11)
  # XE(139), EXTEND
  # XE(140), TABEND
  # XE(141), OLDIN
  # XE(142), CARTPG
  # XE(143), NPROC
  # XE(144), COMPR
  # XE(145), COMBUF(1)

CARTPG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

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

COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER

(1) - NUMBER OF TOKENS IN BUFFER

(2) - NUMBER OF USED WORDS IN BUFFER

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

COMPR - POINTER TO TOKEN CURRENTLY BEING PROCESSED FROM COMBUF

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

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

BITS 0-10 UNUSED

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

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

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

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

BITS 0-9 - NOT USED

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

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

BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR

0 - INITIALIZATION FROM DIRECTIVE
1 - INITIALIZATION FOR REENTRY

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

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

BITS 14-15 - EXECUTIVE STATE

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

WPROC - NUMBER OF PROCESSORS IN LIBRARY

OLIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

PROCAM - NAME OF PROCESS FOR WHICH INTERFACE TABLE EDITOR WAS INVOKE OR BEING EXECUTED IN MANUAL, SEMI OR AUTO-WITH-TRACE MODE

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

REDBU - BUFFER FOR MANAGER WORK AREA REQUESTS (SEE SSD 6.2.7)

RETPH - POINTER TO END OF LAST COMPLETED 8-WORD ENTRY IN REDBU (0 INDICATES REDBU EMPTY) OR RETURN CODE FORM XREC

REDIN - (HIGH BYTE = PARAM), LOW BYTE = PARAM, SEE SSD 6.2.6.3)

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

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

SETPA - POINTER TO LAST SEQUENCE TABLE ENTRY EXECUTED IN SEMI OR AUTO MODE

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

SUBSTA - EXECUTIVE SUB-STATE FLAG (LEVEL IN COMMUNICATION WITH USER TERMINAL). SET TO LEVEL TO BE INITIALIZED NEXT OR ZERO IF LEVEL INITIALIZATION FAILS.
XU CONTENTS (SEQUENCE TABLE EDIT LEVEL)

INTEGER DEBUG
* BEGDO
* ENDO
* PRML
* PRTMB
* PROMPT
* DLAB
* SEQNO
* TABSZ
* XLIBD
* MKBUF
* XLIBD

 equivalence (XI(27), DEBUG)
 (XI(33), NEWTAB)
 (XI(34), PRML)
 (XI(36), DLAB)
 (XI(37), PROMPT)
 (XI(38), SEQNO)
 (XI(39), TABSZ)
 (XI(40), XLIBD)
 (XI(101), MKBUF)

 DIMENSION NEWTAB(3), PRML(3), DLAB(3)

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

 BEGDO = INDEX IN WKBUF TO FIRST TABLE ENTRY TO BE LISTED OR DELETED

 ENDO = INDEX IN WKBUF TO LAST TABLE ENTRY TO BE LISTED OR DELETED

 INSERT = FLAG INDICATING TO INSERT WHAT TYPE OF EDIT TO PERFORM:

 = 0 => REPLACE ENTRY AT TABMD

 = 1 => INSERT A NEW ENTRY IN FRONT OF TABMD

 = 2 => ADD AN ENTRY AT THE BOTTOM OF THE TABLE

 INTHAB = INTERFACE TABLE NAME INPUT OR 0

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

 LIDOSI = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBD

 LNEW = NAME OF SEQUENCE TABLE BEING CREATED

 NUMENT = NUMBER OF ENTRIES (INCLUDING THOSE MARK'S FOR DELETION)

 CURRENTLY IN WKBUF

 OLDBAT = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

 PRML = NAME OF PROMPT CREATED BY ESRM

 PRTMB = CURRENT PROMPTING MODE :

 = 1 => UPDATE MODE ( XP )

 = 2 => CREATE OR REMOVING MODE ( # 20D: )

 = 3 => ALL MODE ( # 200D: PROC, TABLE: )

 PROMPT = PROMPT BUILT BY ESRM

 SEQNO = CURRENT SEQUENCE NUMBER BEING PROMPTED

 TABSZ = SIZE IN WORDS OF TABLE IN WKBUF

 TABMD = INDEX IN WKBUF TO CURRENT TABLE ENTRY

 MAXTAB = MAXIMUM SIZE IN WORDS OF WKBUF

 XLIB = WORKING BUFFER CONTAINING TABLE DURING EDIT SESSION

 XLIB = PROCESSOR LIBRARY DIRECTOY
JSUB F  - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
LEEFF  - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
LISTLU  - LU TO WHICH PRINT SHOULD GO (USED WHEN
LST DIRECTIVE CALLS XILSD OR XICH)
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
LITPTR - INDEX IN WKBUFF TO START OF LITERAL DATA
LITSIZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
LISTLE - FLAG USED TO DETERMINE ORIGIN OF A
CALL TO XILSD OR XICH.
= 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)
MODAV - PREVIOUS VALUE OF PRMTMD WHILE PRMTMD = 4 (CONTINUE)
MARG - INDEX IN WKBUFF TO START OF SHORT PROMPTS
MODB - INDEX IN WKBUFF TO BIT MASK WORD(S) FOR THIS ARGUMENT
NUMTAB - ASCII NAME OF TABLE BEING GENERATED
NUMIM - NO. OF BIT MASKS ASSOCIATED WITH THIS ARGUMENT
NUMARG - NO. OF ARGUMENTS IN THIS INTE. TABLE
NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
PRMLEN - LENGTH (IN WORDS) OF PROMPT BUILT
PRMTMD - CURRENT PROMPTING MODE
= 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
= 3, PROMPT WITH A LIST OF CURRENT VALUES
= 4, CONTINUATION OF SAME ARGUMENT
= 5, PROMPT WITH "\"" (CREATE MODE)
PRM - ASCII ARRAY FOR PROMPT BUILT
SF - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
VERS - VERSION NO. OF THIS INTERFACE TABLE
WKBLNG - 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.
FDS ERROR MESSAGES

ATTENTION FUNCTION

**XAO1** MANAGER H/S TERMINATED - REPLY TO CONTINUE TERMINATION

**XAO2** USER INITIATED INTERRUPT ENTER REQUEST - KILL(S), STATUS(S), RETURN( )

**XAO3** ERROR, LU IS NOT SIGNED ON TO FDS

**XAO4** FDS ATTENTION FUNCTION TERMINATING

**XAO5** FDS MANAGER SIGNaled TO TERMINATE BOTTOM ASSOC TASK

**XAO6** FDS EXECUTIVE ACTIVE - NO ACTION TAKEN

**XAO7** FDS PROCESSOR 'NAME' SCHEDULEd TO ABORT

**XAO8** MANAGER WAITING FOR SYSTEM RESOURCES ... NO ACTION TAKEN

**XAO9** BATCH Job CREATION

**XCO1** CONFfGURATION PROGRAMS

**XCO2** LU 'NN' SIGNED ON TO FDS

**XCO3** LU IS CURRENTLY USING ID 'ID' - SIGN ON REJECTED

**XCO4** 'LU' IS AN INVALID LU

**XCO5** FDS CURRENTLY AT MAX USER'S, CANNOT SIGN ON

**XCO6** 'LU' IS ALREADY SIGNED ON TO FDS

**XCO7** CANNOT FIND 'NAME' ID - SIGN ON TERMINATED

**XCO8** ENTER VALID USER ID (A-Z)

**XCO9** MANAGER

**XCO10** INVALID REQUEST II FROM 'NAME'

**XCO11** FDS SIGN OFF FOR LU 'LU'

**XCO12** INVALID BACK CHAIN FOR CURRENT PROCESSOR

**XCO13** 'N' TRACKS NOT AVAILABLE FOR DWA

**XCO14** NO DWA SPACE FOR 'NAME'

**XCO15** NO AWA FOR DWA DIRECTORY

**XCO16** I/O ERROR FOR DWA, STATUS = 'NNNNNN'

**XCO17** TOC SEARCH ERROR, PHASE 3 COMPRESS

**XCO18** PROCESSOR SERVICES

**XCO19** TYPE OF RESPONSE DOES NOT MATCH TYPE REQUESTED

**XCO20** DATA AREA OVERFLOW

**XCO21** SUBSCRIPT OUT OF RANGE

**XCO22** INVALID ENTRY AFTER SUBSCRIPT

**XCO23** INVALID REPEAT ENTRY

**XCO24** TOO MANY NESTED REPEATS

**XCO25** INVALID SUBSCRIPT

**XCO26** RESPONSE IS TOO LONG FOR BUFFER

**XCO27** INVALID RESPONSE ON OR AFTER COLUMN NN

**XCO28** PROCESSOR INITIALIZATION ERROR

**XCO29** AWA ACCESS FAILURE FOR ...

**XCO30** PARAMETER I/O INCONSISTENCY

**XCO31** INVALID PARAMETER REQUEST

**XCO32** INVALID ORDER ATTRIBUTE(S)

**XCO33** RETRIEVAL OF TOO MUCH DATA REQUESTED

**XCO34** SYSTEM SERVICES

**XCO35** 'ID-NAME' NAMED PROGRAM MADE A PAM REQUEST RESERVED FOR MANAGER

**XCO36** 'ID-NAME' NAMED PROGRAM NOT IN WAIT LIST FOR MANAGER REPLY

**XCO37** 'ID-NAME' NAMLO PROGRAM HAS INVALID BACK CHAIN TO MANAGER
4.0 PDL LISTING PROGRAM

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

1 CD1
2 CD1
3 CD1
4 CD1
5 CD1
6 CD2
7 CD2
8 CD2
9 CD2
10 CD2
11 CD2
12 CD2
13 CD2
14 CD2
15 CD2
16 CD2
17 CD3
18 CD3
19 CD3
20 CD3
21 CD3
22 CD3
23 CD3
24 CD3
25 CD3
26 CD3
27 CD3
28 CD3
29 CD3
30 CD3
31 CD3

INPUT
80 COLUMN PDL IMAGES SUBJECT TO THE FOLLOWING CONVENTIONS:
- IN COLUMN 1 INDICATES PAGE EJECT AND THE FIRST TOKEN (6 OR LESS
CHARACTERS) IS REPRODUCED IN COLUMNS 127-132 OF OUTPUT LISTING
UNTIL NEXT RECORD IS DETECTED. IF RECORD CONTAINS ONLY THE 6-TOKEN FROM THE PREVIOUS RECORD CONTINUES TO APPEAR IN THE
IDENTIFICATION COLUMNS OF THE OUTPUT
- IN COLUMN 1 INDICATES A COMMENT TO BE COPIED TO THE OUTPUT LISTING.
OTHER IN COLUMN 1 INDICATES PDL RECORD TO BE STRUCTURED BASED ON KEY
WORDS APPEARING AS FIRST NON-BLANK CHARACTERS (SEE LOGIC).

OUTPUT
132 COLUMN LISTING AS FOLLOWS
2-6 SEQUENCE NUMBER
8-11 STRUCTURE LEVEL NUMBER
13-125 STRUCTURED LISTING OF 80 COLUMN INPUT RECORDS
127-132 IDENTIFICATION
FOLLOWING THE LISTING IS A SYMBOL DEFINITION TABLE INDICATING THE
SEQUENCE NUMBER OF THE LINE CONTAINING EACH 'BEGIN NAME' AND
"LABEL:"

NOTES
USES FSTWRD & SORT1
89 8
90 8
91 8
92 8
93 8
94 8
95 9
96 8
97 8
98 7
99 6
100 5
101 5
102 4
103 5
104 4
105 4
106 4
107 4
108 4
109 4
110 3
111 2
112 2
113 2
114 1
:SEP: SET LEVEL INCREMENT = 1
DECREMENT LEVEL
:END?: CALL FSTW_RD TO GET NEXT WORD OF PDL
DECREMENT LEVEL
IF WORD = LOOP
THEN
SET LEVEL INCREMENT = 1
ENDIF
:TERM: DECREMENT LEVEL
ENDCASE
ENDIF
ENDIF
COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
ELSE
SET INDENTATION FACTOR = 1
ENDIF
CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
INPUT RECORD AND ID FIELD
OUTPUT IMAGE
APPLY LEVEL INCREMENT
CLEAR PAGE EJECT
ENDIF
ENDDO
CALL SORT1 TO ORDER DEFINITION TABLE
OUTPUT DEFINITION TABLE
END POLIST
116 1 CD1  
117 1 CD2  
118 1 CD3  
119 1 CD4  
120 1 CD5  
121 1 CD6  
122 1 CD7  
123 1 CD8  
124 1 CD9  
125 1 CD10 
126 1 CD11 
127 1 CD12 
128 1 CD13 
129 1 CD14 
130 1 CD15 
131 1 CD16 
132 1 CD17 
133 1 CD18 
134 1 CD19 
135 1 CD20 
136 1 CD21 
137 1 CD22 
138 1 CD23 
139 1 CD24 
140 1 CD25 
141 1 CD26 
142 1 CD27 
143 1 CD28 
144 1 CD29 
145 1 CD30 

EXTRACT THE FIRST TOKEN FROM A PDL RECORD

       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
        ENDOF
        ENDDO
        RETURN LOCATION
        END FSTWRD

REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
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.
| 120 | 1 | XBIT | MULTIPLE WORD BIT STRING BIT CLEAR/SET | DIRECT |
| 121 | 1 |  | COMPARE ARRAYS | DIRECT |
| 122 | 1 | XCP | DOUBLE PRECISION TO ASCII CONVERSION | DIRECT |
| 123 | 1 | XRES | ASCII MANAGEMENT REQUEST ROUTINE | DIRECT |
| 124 | 1 | XREX | EXTRACTS A VARIABLE LENGTH FIELD FROM A WORD | DIRECT |
| 125 | 1 | XRE14 | CONVERT A WORD TO ASCII IN IEE14 FORMAT | DIRECT |
| 126 | 1 | XRI5 | BINARY INTEGER TO ASCII CONVERSION ROUTINE | DIRECT |
| 127 | 1 | XRCK | XSTB BN LOCK | DIRECT |
| 128 | 1 | XRLOC | RETURN 16-BIT ADDRESS OF ARGUMENT | DIRECT |
| 129 | 1 | XN | MOVES WORDS FROM ARRAY1 TO ARRAY2 | DIRECT |
| 130 | 1 | XMCS | FDS EXECUTIVE MESSAGE ROUTINE | DIRECT |
| 131 | 1 | XNB | MULTIPLE WORD BIT STRING BIT SEARCH | DIRECT |
| 132 | 1 | XBD | CONVERT A WORD TO ASCII IN O6 FORMAT | DIRECT |
| 133 | 1 | XRSCN | PACKS CHARACTERS FROM R1 TO A2 FORMAT | DIRECT |
| 134 | 1 | XQ6M | FILE NAME QUALIFICATION | DIRECT |
| 135 | 1 | XRSET | SETS A VARIABLE LENGTH FIELD INTO A WORD | DIRECT |
| 136 | 1 | XRSFL | SHIFT A WORD LEFT LOGICALLY | DIRECT |
| 137 | 1 | XRSFR | SHIFT A WORD RIGHT LOGICALLY (XRSFL EP) | DIRECT |
| 138 | 1 | XBUK | XSTB BN UNLOCK (XRCK EP) | DIRECT |
| 139 | 1 | XQV4M | FILE NAME QUALIFICATION | DIRECT |
| 140 | 1 | XBUK | REMOVES BLANKS AND UNPACKS FROM A2 TO R1 FORMAT | DIRECT |
| 141 | 1 | XRSPT | REMOVE DUPLICATE BLANKS FROM A2 STRING | DIRECT |
| 142 | 1 |  |  |  |
| 143 | 1 | XS | SEQUENCE TABLE EDITOR | DIRECT |
| 144 | 1 | XSCAN | SEQUENCE TABLE EDITOR (STE) DIRECTIVE SCANNER | DIRECT |
| 145 | 1 | XSNEL | STE DELETE DIRECTIVE PROCESSOR | DIRECT |
| 146 | 1 | XSEP | STE ENTITY PROCESSOR | DIRECT |
| 147 | 1 | XSEM | SEQUENCE TABLE EDITOR MAIN ROUTINE | DIRECT |
| 148 | 1 | XSL | STE LIST DIRECTIVE PROCESSOR | DIRECT |
| 149 | 1 | XSTL | SEQUENCE TABLE LIST ROUTINE | DIRECT |
| 150 | 1 | XSNPT | STE INPUT PROCESSOR | DIRECT |
| 151 | 1 | XSNPM | STE NUMBER DIRECTIVE PROCESSOR | DIRECT |
| 152 | 1 | X2PCX | STE TABLE COMPACTER | DIRECT |
| 153 | 1 | X2PMT | STE PROMPT DIRECTIVE PROCESSOR | DIRECT |
| 154 | 1 | X2PRM | STE PROMPT CONSTRUCTOR | DIRECT |
| 155 | 1 |  |  |  |
| 156 | 1 | XT | TERMINAL COMMUNICATIONS | DIRECT |
| 157 | 1 | XCOM | PROMPTS USER, READS RESPONSE, CALLS XTLAN AND XTPRM | DIRECT |
| 158 | 1 | XTLAN | CONVERTS ASCII USER'S RESPONSE TO TOKENS | DIRECT |
| 159 | 1 | XTPRM | HANDLES EXTENDED PROMPTING REQUESTS | DIRECT |
| 160 | 1 |  |  |  |
| 161 | 1 | XU | UTILITY (SOFTWARE AIDS) | DIRECT |
| 162 | 1 | XUDG | ON-LINE SNAP AND MEMORY MODIFICATION ROUTINE | DIRECT |
| 163 | 1 | XUDMF | SYSTEM RESIDENT PARTITION DUMP (XVABN EP) | DIRECT |
| 164 | 1 | XUDPF | FILE MANAGER FILE DUMP PROGRAM | DIRECT |
| 165 | 1 | XUDPL | OCTAL AND ASCII DUMP LINE FORMAT | DIRECT |
| 166 | 1 | XUFMT | DUMP FORMATTER | DIRECT |
| 167 | 1 |  |  |  |
| 168 | 1 | XV | SYSTEM SERVICES | DIRECT |
| 169 | 1 | XVABN | FDS ABEND (SEE XVDFP) | DIRECT |
| 170 | 1 | XVAP | FDS COMMUNICATION SERVICES (POST AND WAIT) | DIRECT |
| 171 | 1 |  |  |  |
| 172 | 1 | VX | EXECUTION CONTROL | DIRECT |
| 173 | 1 | VXAUT | AUTOMATIC MODE | DIRECT |
| 174 | 1 | VXCNT | EXECUTION CONTROL MAIN PROGRAM | DIRECT |
| 175 | 1 | VXDEF | DECODES USER RESPONSE IN MAIN AND SEMI | DIRECT |
| 176 | 1 | VXDEF | READS IN DEFAULT INTERFACE TABLE IF NEEDED | DIRECT |
| 177 | 1 | VXFER | EXECUTES SETTAB AND HANDLES ERROR CONDITIONS | DIRECT |
| 178 | 1 | VXMAN | MANUAL MODE | DIRECT |
FUNCTION
FDS ATTENTION TASK-
USER MAY REQUEST THE CURRENT STATUS OF FDS
FOR THE TERMINAL, TO TERMINATE CURRENT PROCESSOR,
OR (IF FDS HAS TERMINATED) TO TEAR DOWN THE
FDS STRUCTURE FOR THIS TERMINAL.

NOTES
THE ID FOR XATN IS CONNECTED TO THE USER'S
TERMINAL AT FDS SIGN-ON BY USING THE ENT
FOR THE DEVICE. THE FUNCTION IS DISCONNECTED
AT FDS SIGN-OFF.
SAVE EQT ADDRESS(IN BREG ON ENTRY)
CALL ESLU(BREG) GET LU IN ASCII & BINARY
STARTSEARCH UNTIL LAST STATUS TABLE ENTRY
EXITLF STLU EN LU
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
 IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STMB)
IF MANAGER IS DORMANT, THEN
WRITE "**xao? MANAGER HAS TERMINATED;
REPLY TO CONTINUE TERMINATION:
READ(LU) ** WAIT FOR REPLY **
LOCK ON THE FDS TABLE RESOURCE
CALL SLIRI disable
IF STBEXEXECUTIVE ADDRESS .NE. 0, THEN
IF STBAT(CURRENT) .NE. STBEX, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XMCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STMB(MANAGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARN ONE(P)
CALL SLIBX ENABLE
CALL MESSAGE 'OFF, BOTTOM'
CALL SLIRG DISABLE
SET BOTTOM TO NEXT-AT
ENDO
ENDIF
ENDIF
CALL SLIST MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBR
ENDIF
DECREMENT NUMBER ACTIVE(STBA)
GET EQT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBR)
CLEAR STBR
CLEAR MANAGER'S ID, STMB, & STBL
ENABLE... (VIA A JMP TO EXEC(DISPATCHER))
RELEASE EXEC'S AND PROCESSOR'S CLASS NUMBERS
CLEAR LOCK ON FDS TABLE
ELSE ** MANAGER IS STILL ALIVE **
WRITE "**XAO? USER INITIATED INTERRUPT?
WRITE 'ENTER REQUEST=KILL(?) STATUS(?),OR RETURN(BLANK)'
READ (LU) REQUEST
IF REQUEST IS KILL OR X, THEN
PERFORM XAKILL
ELSE
IF REQUEST IS STATUS OR $, THEN
PERFORM XASTAT
ENDIF
ENDIF
ELSE
WRITE "**XAO? ERROR LU IS NOT SIGNED-ON TO FDS"
ENDIF
WRITE "**XAO? FDS ATTENTION FUNCTION TERMINATING"
END XATH
77 1 F:GIN XSTAT    PRODUCE A FDS STATUS REPORT
78 2 CALL SLIBR   DISABLE
79 2 GET CURRENT-TIME FROM TIME
80 2 GET MANAGER'S ADDRESS FROM STBMG
81 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
82 2 GET EXECUTIVE'S ADDRESS FROM STBEK
83 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
84 2 GET CURRENT AT FROM STBAT
85 2 MOVE NAME, STATUS, PARTITION, & PRIORITY
86 2 PERFORM XABTN(CURRENT) FIND BOTTOM AT
87 2 SET BOTTOM TO CURRENT
88 2 GO WHILE FATHER-ID NE ZERO
89 3 SET FATHER-ID FROM CURRENT
90 3 CALCULATE NEXT
91 3 IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
92 4 MOVE NAME, STATUS, PARTITION, & PRIORITY
93 4 ENDF
94 4 IF NEXT IS THE MANAGER, THEN
95 4 SET CURRENT AS TOP
96 4 ENDF
97 3 SET CURRENT TO NEXT
98 2 ENDDO
99 2 IF TOP EQ ZERO, THEN
100 3 USE TOP TO MOVE NAME, PARTITION, & PRIORITY
101 3 SET STATUS TO 'IN USE' OCTAL 17
102 3 ENDF
103 2 CALL SLIBX   ENABLE
104 2 WRITE FIRST SET OF HEADERS
105 2 SET TOP AS REPORT DATA
106 2 WRITE REPORT LINE
107 2 SET MANAGER AS REPORT DATA
108 2 WRITE REPORT LINE
109 2 SET EXECUTIVE AS REPORT DATA
110 2 WRITE REPORT LINE
111 2 SET CURRENT AS REPORT DATA
112 2 WRITE REPORT LINE
113 2 WRITE INTERMEDIATE HEADERS
114 2 DO UNTIL MAX ENTRIES OR NO MORE DATA
115 3 WRITE REPORT LINE
116 3 SET NEXT REPORT DATA
117 3 ENDDO
118 1 END XSTAT
119 1 SAMPLE REPORT  ******************
120 1  FDS STATUS FOR LU 10 HH:MM:SS 360
121 1  # NAME PRIOR PART# STATUS
122 1  # TOP AT-  PROCE 922 5 GENERAL WAIT
123 1  # MANAGER- XGNNH 40 5 GENERAL WAIT
124 1  # EXECUTIVE- XEXXH 60 3 GENERAL WAIT
125 1  # CURRENT AT-  PROCE 1131 4 GENERAL WAIT
126 1  # BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
127 1  # PROCE 32767 6 DISC ALLOCATE SUSPEND
128 1  PROC 1131 4 GENERAL WAIT
129 1  PROCD 2000 3 GENERAL WAIT
130 1  PROCB 845 6 GENERAL WAIT
BEGIN XAKILL  
2  TERMINATE CURRENT FDS FUNCTION
3  IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
4  ON NEXT RETURN VIA A PWR.
5  IF THE EXEC IS ACTIVE-DO NOTHING
6  IF A PROCESSOR IS ACTIVE- USE RTI MESEE TO OFF THE PROCESSOR
7  GET MANAGER'S ID ADDRESS(STBMG)
8  IF STATUS OF MANAGER IS NOT WAIT, THEN
9  SET TERMINATE FLAG IN STD-ENTRY
10  WRITE "***XA05 FDS MANAGER SIGNALS TO TERMINATE SEQUENCE'
11  ELSE
12  IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
13  WRITE "***XA06 FDS EXECUTIVE ACTIVE; NO ACTION TAKEN'
14  ELSE
15  PERFORM XATHM(CURRENT) FIND BOTTOM AT
16  IF BOTTOM AT IS D.RTR OR SHP THEN
17  WRITE "***XA08 MANAGER IS WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN.'
18  EXIT XAKIL
19  ELSE
20  IF RETURNED BOTTOM IS MANAGER THEN
21  IF MANAGER IS NOT WAITING ON A PROGRAM THEN
22  WRITE "***XA08 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN.'
23  EXIT XAKIL
24  ELSE
25  SET RETURN PARAMETER TO PROCESSOR ABENDED
26  INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
27  CALL PBLST TO REACTIVATE MANAGER
28  ENDIF
29  ENDIF
30  END IF ***XA07 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT.'
31  IF RETURNED BOTTOM WAS NOT MANAGER THEN
32  SET NAME IN 'OFF' COMMAND
33  CALL MESEE TO 'OFF' THE PROCESSOR
34  ENDIF
35  ENDIF
36  ENDIF
37  ENDIF
38  ENDIF
39  END XAKILL
BEGIN XCNFG

* O D1 FD S CONFIGURATION MANAGER
* O D1 INITIATES AN FDS SYSTEM FOR A REQUESTED TERMINAL OR
* O D1 TERMINATES AN FDS SYSTEM FOR A REQUESTED TERMINAL
* O D1 INITIATED VIA
  R U, FDS, LU, ID, CWA SIZE, P A R M S, O P T I O N S
* O D2 INPUTS
  O D2 LOGICAL UNIT(LU) FOR THE REQUESTED TERMINAL,
  O D2 A P A R M TO DENOTE INITIATION OR TERMINATION
  O D2 A ONE CHARACTER USERID
  O D2 A DEBUG OPTION INDICATOR
  O D2 THE NUMBER OF DNA TRACKS
* O D3 OUTPUTS
  O D3 INITIATION-
    O D3 A BLANK ID-SEGMENT WILL BE CONSTRUCTED FOR THE FDS MANAGER,
    O D3 AND FDS EXECUTIVE
  O D3 THE EMT FOR THE REQUESTED LU WILL BE CONNECTED
  O D3 TO THE FDS ATTENTION TASK
  O D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
  O D3 WILL BE UPDATED TO REFLECT THE INITIATION
  O D3 TERMINATION-
    O D3 THE FD S ID SEGMENTS WILL BE RETURNED TO BLANK STATUS
    O D3 THE EMT FOR THE LU WILL BE REINSTATED
    O D3 THE FDS TABLE(FDSTAB) IN RESIDENT LIBRARY ROUTINE
  O D3 WILL BE UPDATED FOR THE TERMINATION

RETROACTIVITY OF THE ORIGINAL PAGE IS POOR
29 2 * PARM=LU,ID,DVA SIZE,PARN(ON OR OFF),OPTIONS
30 2 * SWITCH INPUT PARN AROUND SO THAT
31 2 * NOW PARM=LU,P2(ON OR OFF),ID,OPTS,DVA SIZE
32 2 * FOR COMPATABILITY TO BUILD 1.
33 2 * CALL AMPAR(PARMS)
34 2 IF LU IS .LT. 0, OR
35 3 ,GT. LMAK(1653), OR
36 3 ,EQ. 6(PRINTER), OR
37 3 THE DRIVER IS .NE. 0 OR 5, THEN
38 3 ISSUE MESSAGE "**XCO4 'LU' IS AN INVALID LU"
39 3 ELSE
40 4 IF PARM P2 IS OFF THEN
41 5 PERFORM XCOFF SIGN OFF
42 5 ELSE
43 6 PERFORM XCON SIGN ON
44 5 ENDIF
45 2 ENDIF
46 2 :EXEC
47 2 CALL EXEC PROGRAM TERMINATION
48 1 END XCMFG
1 BEGIN XCOM
2   * SIGN ON A USER TO FDS
3      UNLESS VALID USER ID (P3)
4      IF ID NOT A - 2 - THEN
5      WRITE 'XCOG ENTER VALID ID (A - 2)'
6      END RESPONSE
7      ENDDO
8      IF USER ID IS BEING USED, THEN
9      WRITE '***XCOG LU 'LU' IS CURRENTLY USING ID 'ID'-SIGN ON REJECTED'
10     EXIT :XCOG
11     ENDDO
12     IF FDS RESOURCE NUMBER NOT DEFINED, THEN
13      CALL RMAQ (GLOBAL ALLOCATE, LOCAL SET)
14     ELSE
15      CALL RMAQ (LOCAL SET)
16     ENDDIF
17     IF NUMBER SIGNED ON (STBNK), THEN
18      ISSUE MESSAGE '***XCOG FDS CURRENTLY AT MAX USER'S.
19     ELSE
20      DO FOR STBNK (NUMBER OF FDS ENTRIES)
21         IF ENTRY'S LU (STBNK) ,EQ. REQUESTING LU (P1) THEN
22            ISSUE MESSAGE '***XCOG 'LU' IS ALREADY SIGNED ON TO FDS'
23            EXIT TO :XCOG
24         ELSE
25            IF THIS ENTRY IS AVAILABLE, THEN
26               SET AS CURRENT-ENTRY-ADDRESS
27            ENDDIF
28         ENDDO
29      ENDDO
30      IF USER NOT FOUND, THEN
31         IF XGR NOT FOUND AND THIS ID .EQ. XGR, THEN
32            SET ID ADDRESS OF XGR
33            INCREMENT NUMBER-FOUND
34         ELSE
35            SET ID ADDRESS OF XATM
36            INCREMENT NUMBER-FOUND
37         ELSE
38            IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK, THEN
39               SET ID ADDRESS OF FIRST-BLANK
40               INCREMENT NUMBER-FOUND
41         ELSE
42            IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK, THEN
43               SET ID ADDRESS OF SECOND-BLANK
44               INCREMENT COUNT
45         ENDF
46         ENDF
47         ENDF
48         ENDF
49         ENDF
50         END
51
CALL SLIBX ENABLE
  ISSUE MESSAGES "**XCO1 CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP
BUILD ENTRY IN XSTUB
SET LU INQ STBLU
SET LU IN ASCII INTO STBLA
SET USER'S ID INTO STID
SET ADDRESS OF FIRST-BLANK INTO STBL
SET ADDRESS OF SECOND-BLANK INTO STBE
INCREMENT ACTIVE COUNT(STBAC)
BUILD XMGNN & XETNN
MOVE PRIORITY THRU DISC ADDRESS FROM XMGN TO FIRST-BLANK
TURN ON TN BIT
SET NAME TO XPMN
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
TURN ON TN BIT
SET NAME TO XETNN
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE ERT OVERLAY
SAVE ERT VALUE IN STBE
SET ID ADDRESS OF XATM INTO ERT
ENDIF
SET INPUT PARS INTO ID OF XGMNN
SCHEDULE XGMNN VIA BLIST
CALL BLIST
CALL SLIBX ENABLE
IF FDS HAS A FATHER, THEN
CALL MESS "OFF, FATHER"
ENDIF
ISSUE MESSAGE "**XCO1 LU "NN" SIGNED ON TO FDS"
ENDSEARCH
:XCETA
CALL RMN (LOCAL CLEAR)
ENDIF
CALL EXEC TERMINATE
END XCON
FORTRAN CALLING PROCEDURE

CALL XELS (XDLCD)

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

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

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

INPUT

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

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

INTERNAL VARIABLES

LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES

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

NOTES

USING .ENTR, XDLST, XERTH

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

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

BEGIN XDLCD

EXTEND DIRECTIVE INDEX FROM MASSTA

CASE (LIST:) INDEX

LIST: CALL XDLST

ENDCASE

CALL XERTH TO RETURN FROM SEGMENT

END XDLCD
40 1 #00 FORTRAN CALLING PROCEDURE
41 1 #00 CALL XELS (XDCLF)
42 1 #00
43 1 #00
44 1 #01 XDCLF DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
45 1 #01 REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
46 1 #01 TOC
47 1 #01 SAVE
48 1 #01 RECALL
49 1 #01 DELETE
50 1 #01 RENAME
51 1 #01 COPY
52 1 #01 CLEAR
53 1 #01 OFF
54 1 #01
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 #02 INTERNAL VARIABLES
62 1 #04 LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
63 1 #04
64 1 #04
65 1 #04
66 1 #05 NOTES
67 1 #05 USES :ENTR, XDLE, XDOP, XDELE, XDME, XDREM, XDRC, XDRE, XDREY
68 1 #05 XDTE, XDTEH
69 1 #05 XDCLF IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
70 1 #05 CONTAINING THE REFERENCED DIRECTIVES
71 1 #05
72 1 #05
73 1 #05
74 1 *
75 1 *
76 1 *
77 1 BEGIN XDCLF
78 2 EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
80 3 :TOC: CALL XDTEC
81 3 :SAVE: CALL XDREY
82 3 :RECA: CALL XDRE
83 3 :DELE: CALL XDELE
84 3 :REMA: CALL XDREM
85 3 :COPY: CALL XDOP
86 3 :CLEA: CALL XDCLF
87 3 :OFF: CALL XDREY
88 2 ENDCASE
89 2 CALL XECHN TO RETURN FROM SEGMENT
90 1 END XDCLF
**FORTRAN CALLING PROCEDURE**

**XOCLU DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS**
**REQUESTED AND CALLS THE APPROPRIATE HANDLE SUBROUTINE**

**INPUT**

**INTERNAL VARIABLES**

**LIST - ORDERED LIST OF APPROPRIATE HANDLE ADDRESSES**

**NOTES**

**USES .ENTR, XOSTO, XDRES, XDUML, XDLOA, XDABAT, XERTH**

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

**CONTAINING THE REFERENCED DIRECTIVES**

BEGIN XOCLU

1. EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 7
2. CASE (:STOR, :REST, :UMLO, :LOAD, :BATCH) INDEX

3. :STOR: CALL XOSTO

4. :REST: CALL XDRES

5. :UMLO: CALL XDUML

6. :LOAD: CALL XDLOA

7. :BATCH: CALL XDABAT

8. END CASE

9. CALL XERTH TO RETURN FROM SEGMENT

END XOCLU
FORTRAN CALLING PROCEDURE

**CALL XDCLE**

**INPUT**
- XE COMMON - CART6, FLG6, LU
- MANAGER - AWA HEADER AND TOC (SEE XMAMA)

**OUTPUT**
- XE COMMON - REBU, REO8TP, COBET-S8ATCH
- XB COMMON - SCRATCH BEYOND XB(200)
- XS COMMON - SCRATCH
- MANAGER - COMMANDS TO PURGE AWA AND RESTORE CHAINS 0 AND 8

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

**NOTES**
- USES EXEC, IAND, PRGN, PURGE, XREQ, XH6, XMMOV, XMMSG, XRO6, XRO8, XX0,
- IN THE EVENT THAT A COMPACTED AWA TOC WILL NOT FIT IN THE ALLOCAT-
- RD BUFFER SPACE AND ELEMENTS OF CHAIN 1 OR 8 EXTEND BEYOND IT, THE
- CLEAR FUNCTION WILL NOT BE PERFORMED.
- SHOULD THE RESTORATION OF CHAIN 1 OR 8 FAIL AFTER THE CLEAR
- REQUEST HAS BEEN COMPLETED BY THE MANAGER, FBS WILL BE TERMINATED
- IN ORDER TO PROTECT USER DISK FILES.
1 BEGIN XOCLE
2 RETRIEVE TOC
3 BUILD REQUEST TO CLEAR AWS
4 DO UNTIL END OF PERMANENT SYSTEM TABLES CHAIN (CHAIN 1)
5 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
6 EXIT TO :ERR48: IF RESTORATION REQUEST QUEUE IS FULL
7 BUILD REQUEST TO REALLOCATE TABLE
8 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
9 BUILD REQUEST TO STORE TABLE INTO AWS FROM SAM
10 ENDDO
11 DO UNTIL END OF DATABASE FILES CHAIN (CHAIN 8)
12 EXIT TO :ERR24: IF CHAIN POINTS BEYOND TOC BUFFER
13 EXIT TO :ERR48: IF RESTORATION REQUEST QUEUE IS FULL
14 BUILD REQUEST TO REALLOCATE TOC ENTRY
15 ENDDO
16 BUILD REQUEST TO TERMINATE LIST
17 DO UNTIL END OF DRDE CHAIN (CHAIN 3)
18 THEN
19 OUTPUT XD13 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
20 EXIT PURGE LOOP
21 ENDF
22 PURGE FILE
23 ENDDO
24 DO FOR EACH BLOCK OF EIGHT REQUESTS
25 TRANSMIT BLOCK TO MANAGER
26 EXIT TO :ERR23: IF REQUESTS FAILED
27 ENDDO
28 EXIT XD24
29 :ERR23:
30 DO FROM FAILING REQUEST TO END OF LIST
31 IF REQUEST TO STORE
32 THEN
33 READ SAM TO FREE BUFFER AND CLASS NUMBER
34 ENDF
35 ENDDO
36 DO UNTIL END OF DATABASE FILE CHAIN (CHAIN 8)
37 IF FILE IS UTDB (TYPE 1)
38 THEN
39 CALL PURGE TO DELETE FILE
40 ENDF
41 ENDDO
42 PURGE ALL UTDB FILES
43 TERMINATE FDS WITH CLEAR FAILURE MESSAGE
44 :ERR48:
45 DO FOR ALL STORE REQUESTS BUILT
46 READ SAM TO FREE BUFFER AND CLASS NUMBER
47 ENDDO
48 EXIT XD24 WITH CLEAR FAILURE MESSAGE
49 1 END XD24
C**\*
240 C**\*
241 C**\* FORTRAN CALLING PROCEDURE
242 C**\*
243 C**\* CALL XDCOP
244 C**\*
245 C**\*
246 C**\*
247 C**\*
248 C**\* XDCOP PROCESSES A USER REQUEST TO COPY A SEQUENCE TABLE,
249 C**\* INTERFACE TABLE, DATA ELEMENT, DDE, UTDB, OR FOR. ONLY
250 C**\* THOSE TABLES OR ELEMENTS LOGGED IN THE USER'S AWA CAN BE COPIED.
251 C**\*
252 C**\*
253 C**\*
254 C**\* INPUT
255 C**\*
256 C**\* COMMON XE - CARTAG, COMBUF, QUAL, REQTR, TOKENS
257 C**\*
258 C**\*
259 C**\*
260 C**\* OUTPUT
261 C**\*
262 C**\* COMMON XE - COMPTR, RERBUF
263 C**\*
264 C**\*
265 C**\*
266 C**\* INTERNAL VARIABLES
267 C**\*
268 C**\*
269 C**\* COMMON XE - (1) DATCLS: DATA CLASS CODE (STORED IN LEFT BYTE)
270 C**\* (2) DSTYPE: TYPE CODE FOR DATA BASE FILES
271 C**\* (3) I: INDEX
272 C**\* (4) IDC: EXEC BUFFER
273 C**\* (148) IERR: FILE MANAGER ERROR RETURN
274 C**\* (149) IMSG: ERROR MESSAGE NUMBERS
275 C**\* (150) FNNAME: NEW FMGR FILE NAME
276 C**\* (153) OMNAME: OLD FMGR FILE NAME
277 C**\* (154) OMNAME: POINTER TO OLD NAME IN COMBUF
278 C**\* (155) WMNAME: NEW FMGR FILE NAME
279 C**\* (156) WMNAME: POINTER TO OLD NAME IN COMBUF
280 C**\* (157) TMNAME: TOC ENTRY FOR DATA BASE RENAME
281 C**\* (166) EOF: EOF RETURN FROM FMGR
282 C**\*
283 C**\* COMMON XE - (204) IBUFF: USER BUFFER FOR EXEC I/O (1024)
284 C**\* (124A) FMRTYPE: FMGR FILE TYPE
285 C**\* (1245) PPARM1: FIRST BYTE OF REQTR
286 C**\* (1246) PPARM2: SECOND BYTE OF REQTR
287 C**\* (1247) PREFIX: PREFIX FOR FILE NAME
288 C**\* (1248) SCRITY: FILE SECURITY CODE (0 OR 88)
289 C**\* (1249) SIZE: FILE SIZE IN BLOCKS
290 C**\* (1251) IDC: EXEC BUFFER
291 C**\*
292 C**\*
293 C**\* Routines Used -
294 C**\*
295 C**\* CLOSE, CREAT, EXEC, OPEN, PULGE, READF, WRITF, XDBOA, XDDOD,
296 C**\* XREQ, XRMOV, X RMSG, XRFPN, XRSEC
297 C**\*
BEGIN XDCOP
EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
SAVE POINTER TO CURRENT NAME
IF NEXT TOKEN IS HYPHEN, THEN
DECODE CLASS NAME
EXIT TO :CLASSER: IF CLASS SPECIFIED IS NOT VALID (S,I,O,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 AWA MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
BUILD AWA MANAGER REQUEST TO VERIFY NEW NAME
CALL X plight
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 XDBBA TO ADD NEW POB TO LOG FILE (XDBBA)
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 :INVTD: IF FILE IS NOB
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 XR�认 TO FORMAT NEW FILE NAME
CALL CREATE FOR NEW FILE
EXIT TO :FILERR: IF FMGR ERROR RETURNED
CALL XR�认 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 TO READ 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
CALL CLOSE FOR OLD FILE
CALL CLOSE FOR NEW FILE
BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY
EXIT TO :TOCRR: IF ERROR IS INDICATED
ELSE
BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
BUILD MANAGER REQUEST FOR DATA RETRIEVAL
CALL XREQ
EXIT TO :TOCRR: 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 :TOCRR: IF ERROR IS INDICATED
ENDIF
EXIT TO :TOCRR: IF ERROR IS INDICATED
EXIT XDCOP
:SYNTAX: CALL XRMSG -"SYNTAX ERROR ..." AND EXIT
:CLASS: CALL XRMSG -"INVALID CLASS DESIGNATOR ..." AND EXIT
:NAMERR: CALL XRMSG -"NEW NAME IS INVALID ..." AND EXIT
:AXERR: CALL XRMSG -"AUTHORIZED LIMIT ..." AND EXIT
:INVAL: CALL XRMSG -"NEW CANNOT BE ..." AND EXIT
:FILEERR: CALL XRMSG -"FILE ACCESS ERROR # ... ON ......." AND EXIT TO :END:
:TOCRR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
:TYPEERR: CALL XRMSG -"INCONSISTENT FILE TYPE ..."
-END:
IF POD HAS BEEN LOGGED IN XPDD, THEN
CALL XDDOD TO DELETE POD FROM XPDD
ENDIF
IF A NEW FILE HAS BEEN BUILT, THEN
PURGE NEW FILE
CLOSE OLD FILE
ENDIF
END XDCOP
FORTRAN CALLING PROCEDURES

CALL XORDB(NAME, IERR, ISIZE)
CALL XORDB(NAME, IERR)

********

1 XORDV, XORDV, AND XORDB PROVIDE THE capability TO ADD AN NDB OR
2 PDB TO NDB/PDB DIRECTORY (XPDB). TO VERIFY THAT AN NDB OR PDB IS
3 LOGGED IN XPDB AND RETRIEVE THE SIZE OF THAT NDB/PDB, AND TO
4 DELETE AN NDB OR PDB FROM XPDB, RESPECTIVELY

********

1 INPUT
2 NAME - 2 WORD INTEGER ARRAY CONTAINING THE UNQUALIFIED ASCII
3 NDB/PDB NAME
4 ISIZE - INTEGER WORD CONTAINING THE NDB/PDB SIZE IN BLOCKS (FOR
5 XORDB ONLY)

********

1 OUTPUT
2 IERR - INTEGER WORD CONTAINING RETURN CODE
3 =0 FAGE ERROR CODE RETURNED FOR XPDB ACCESS
4 =0 NO ERRORS
5 =1 DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE
6 =2 PDB/PDB LIMIT EXCEEDED
7 ISIZE - INTEGER WORD CONTAINING THE NDB/PDB SIZE IN BLOCKS (FOR
8 XORDB ONLY)

********

1 COMMON USED
2 XE - EQUAL
3 CARTAG
4 X - DCD (WORDS 1-144)
5 ********

1 NOTES
2 XRULK, XRLCK, .ENTR, OPEN, CLOSE, READS, AND WRITF ARE USED
3 WHEN WORKING WITH NDB'S, QUAL SHOULD BE SET TO 77 OCTAL
4 ********
BEGIN XDDBA
  STORE RETURN ADDRESS
  CALL XTRK TO SET UP CALLING ARGUMENTS
  CALL XLCK FOR EXCLUSIVE USE OF XDDB
  CALL OPEN FOR EXCLUSIVE USE OF XDDB
  EXIT TO :FILER; IF ERROR RETURNED
  COMPUTE RECORD NUMBER FOR USER'S DIRECTORY  (QUAL-77B)/2+1
  CALL READ FOR RECORD COMPUTED
  EXIT TO :FILER; IF ERROR RETURNED
  DETERMINE PART OF RECORD TO BE USED
  IF REQUEST IS FOR ADD; THEN
  EXIT TO :MAXERR; IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
  START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
  EXIT TO :NAMEERR; IF ENTRY NAME MATCHES PARAMETER NAME
  ENDDO
  ENDSEARCH
  STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
  INCREMENT # OF CURRENT ENTRIES
  ELSE
  START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
  EXIT IF ENTRY NAME MATCHES PARAMETER NAME
  ENDDO
  EXIT TO :NAMEERR:
  ENDDO
  ENDS
  IF REQUEST IS FOR VERIFY, THEN
  STORE WORD 3 OF ENTRY IN ISIZE
  ELSE
  REPLACE ENTRY WITH LAST ENTRY
  STORE ZEROS IN LAST ENTRY
  DECREMENT # OF CURRENT ENTRIES
  ENDIF
  ENDF
  CALL WRIT TO WRITE RECORD TO XDDB
  EXIT TO :RETURN;
  :NAMEERR: SET IERR = 1 AND EXIT TO :RETURN;
  :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
  :FILER: SET IERR = FNAME ERROR CODE
  :RETURN: CALL CLOSE FOR XDDB
  CALL XRK TO RETURN RESOURCE #
FORTRAN CALLING PROCEDURE

CALL XDELE

XDELE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT
SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE AWA.
IF THE ELEMENT IS A DATA BASE, THE ASSOCIATED FILE
MANAGER FILE IS PURGED AND FOR A PBD THE PBD DIRECTORY
IS UPDATED

INPUT

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

OUTPUT

COMMON XE - RERBUF

NOTES

ROUTES USED

EXEC

IANH

PURGE

XDDD

XREG

XRET

XRI6

XMOV

XMSG

 XPAR

XUPK

XUDDG

**********

**********

**********
1 BEGIN XDELE
2 DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED
3 ERRORT IF COMMA IS NOT NEXT LEXICAL ELEMENT :ERROR:
4 IF CLASS DESIGNATOR IS SPECIFIED, THEN
5 SET REQUESTED CLASS APPROPRIATELY (B, S, I, O, OR F)
6 ELSE
7 SET REQUESTED CLASS TO BE (O)
8 ENDIF
9 IF DATA BASE TO BE DELETED, THEN
10 BUILD AND ISSUE AMA MANAGER REQUEST FOR TOC ENTRY
11 ENDIF
12 IF ELEMENT IS NOT A MASTER DATA BASE, THEN
13 IF ELEMENT IS A PERSONAL DATA BASE, THEN
14 CALL PROD TO DELETE THIS PROD FROM XPDB
15 IF ERROR IS RETURNED, THEN
16 CALL XRMSG - "FILE ACCESS ERROR 0... XPDB"
17 SET ERROR FLAG
18 ENDIF
19 ENDIF
20 BUILD AND ISSUE AMA MANAGER REQUEST TO DELETE ELEMENT SPECIFIED
21 IF RETURN CODE INDICATES ELEMENT DOES NOT EXIST, THEN
22 CALL XRMSG - "XXXXX NOT FOUND"
23 SET ERROR FLAG
24 ENDIF
25 IF CLASS IS DATA BASE (C), OR
26 CLASS IS DDED (F), THEN
27 IF ERROR FLAG IS NOT SET, THEN
28 CALL XRDFN TO CONSTRUCT FILE NAME
29 ISSUE RTE PURGE FOR THE FILE
30 IF RETURN CODE FROM PURGE, THEN
31 CALL XRMSG - "FILE ERROR WHEN XXXX" 
32 ENDIF
33 ENDIF
34 ELSE
35 CALL XRMSG - "... IS A HDX. NOT DELETED."
36 ENDIF
37 1 END TO :RETURN:
38 2 :ERROR: CALL XRMSG - "SYNTAX ERROR"
39 2 :RETURN:
40 1 END XDELE
CALL XNLI5 (TOCLST)

XNLI5 PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECORING EACH ELEMENT AND FLAGGING IT IN TOCLST AS AN ELEMENT
TO BE STORED/RESTORED.

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XB(151) ABLFLG, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

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

OUTPUTS IN COMMON:

XB(151) ABLFLG, XB(157) TOTSIZ, XB(158) TRYWD

INTERNAL XB COMMON USED:

XB(151) ABLFLG - (INTEGER, 1 WORD) ABORT FLAG
XB(152) ERLFLG - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER
XB(157) TOTSIZ - (INTEGER, 1 WORD) TOTAL # BLOCKS OF DATA TO
XB(158) TRYWD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
XB(200) HNPTR - (INTEGER, 1 WORD) INDEX TO NAME IN COMBUF.
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCLST

COMMON USED:

EQUIVALENCE

+ (RE(53), MAST(3)), (RE(85), EOS )
+ (RE(85), NAME ), (RE(92), HTYP )
+ (RE(113), COMMA ), (RE(144), COMPTA )
+ (RE(145), COMBUF)
1 BEGIN XDLIS
2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
3 TURN ERFLG OFF

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

25 :RTN2:
26 IF ERROR FLAG IS ON THEN
27 CALL XCOM TO REPRROMPT USER TO CONTINUE
28 ERREXIT IF RESPONSE IS X TO :ERR3:
29 ENDIF
30 ENDDO
31 EXIT XDLIS

32 :ERR1:
33 SET ERROR FLAG ON
34 CALL XMESG TO DISPLAY SYNTAX ERROR
35 GO TO :RTM2:

36 :ERR2:
37 IF ERROR FLAG IS OFF THEN
38 TURN ERROR FLAG ON
39 CALL XMESG TO DISPLAY NOT STORED/RESTORED MESSAGE
40 ENDIF
41 CALL XEXEC TO DISPLAY ELEMENT NAME
42 GO TO :RTM1:

43 :ERR3:
44 SET ABFLG TO ABORT STORE/RESTORE OPERATION
45 END XDLIS
693 1 CD0  FORTRAN CALLING PROCEDURE
694 1 CD0
695 1 CD0  CALL XLST
696 1 CD0
697 1 CD0
698 1 CD0  ********
699 1 CD0
700 1 CD0  PROCESS THE LIST DIRECTIVE TO LIST ALL TABLES AND/OR
701 1 CD0  DATA ELEMENTS SPECIFIED.
702 1 CD0
703 1 CD0  ********
704 1 CD0
705 1 CD0  INPUT
706 1 CD0
707 1 CD0  COMMON XE - COMBUF, COMPR, FLAGS, LU, TOKENS
708 1 CD0
709 1 CD0  ********
710 1 CD0
711 1 CD0  OUTPUT
712 1 CD0
713 1 CD0  PRINT TO LOGICAL UNIT 6 OR TO LOGICAL UNIT 'LU'
714 1 CD0
715 1 CD0  ********
716 1 CD0
717 1 CD0  INTERNAL VARIABLES
718 1 CD0
719 1 CD0  COMMON XE - BUFFER = PRINT LINE TO BE OUTPUT
720 1 CD0  BUFFTR = INDEX INTO BUFFER FOR NEXT ASCII DATA
721 1 CD0
722 1 CD0
723 1 CD0  COMMON XE - POINTR = CHAIN POINTER TO NEXT (OR 1ST) TOC
724 1 CD0
725 1 CD0  ********
726 1 CD0
727 1 CD0  NOTES
728 1 CD0
729 1 CD0  USES ROUTINES - EXEC, XEINT, XLSD, XILST,
730 1 CD0
731 1 CD0  XREX, XRX, XRMV, XRMG,
732 1 CD0
733 1 CD0  XRPCK, XRP, XILST, OPEN,
734 1 CD0
735 1 CD0  CLOSE, READY
783 1 CD**********
784 1 CD0      FORTRAN CALLING PROCEEDURE
785 1 CD0      CALL XD0FF
786 1 CD0      XD0FF
787 1 CD0      CD**********
788 1 CD1      XD0FF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
789 1 CD1      DELETES ALL ORDE AND UT0B FILES LOGGED IN THE ANA,
790 1 CD1      PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
791 1 CD1      RETURNS NORMAL PROGRAMS TO THE FDS MANAGER AND TERMINATES
792 1 CD1      NORMALLY VIA RTE.
793 1 CD1
794 1 CD2      INPUT
795 1 CD2
796 1 CD2
797 1 CD2
798 1 CD2
799 1 CD2
800 1 CD2
801 1 CD2
802 1 CD2
803 1 CD2
804 1 CD2
805 1 CD2
806 1 CD2
807 1 CD2
808 1 CD2
809 1 CD**********
810 1 CD4
811 1 CD4
812 1 CD4
813 1 CD4
814 1 CD4
815 1 CD4
816 1 CD4
817 1 CD4
818 1 CD4
819 1 CD**********
820 1 CD5
821 1 CD5
822 1 CD5
823 1 CD5
824 1 CD5
825 1 CD5
826 1 CD5
827 1 CD**********
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL THEN TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR DRDE FILES IS NOT NEGATIVE THEN
6 DO UNTIL DRDE CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEHIND END OF TOC BUFFER THEN
8 OUTPUT 'XD15 TOC TOO LARGE, PURGE INCOMPLETE'
9 EXIT DO
10 ENDIF
11 CALL XZRN TO CREATE FILE NAME '/XXXX'
12 CALL PURGE TO SCRATCH FILE
13 SET DRDE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
14 ENDDO
15 ENDIF
16 IF CHAIN HEAD FOR DATA BASE IS NOT NEGATIVE, THEN
17 DO UNTIL DATA BASE CHAIN HEAD IS NEGATIVE
18 IF CHAIN POINTS BEHIND END OF TOC BUFFER THEN
19 OUTPUT 'XD15 TOC TOO LARGE, PURGE INCOMPLETE'
20 EXIT DO
21 ENDIF
22 IF TYPE OF DATA BASE IS UTDD, THEN
23 CALL XZRN TO CREATE FILE NAME 'UXXXX'
24 CALL PURGE TO SCRATCH FILE
25 ENDDO
26 SET DATA BASE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
27 ENDDO
28 ENDIF
29 CALL XDATA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XUDBG
32 ENDIF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVWBN - NO RETURN FROM THIS CALL
35 ENDIF
36 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
37 CALL XEXIT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE EXEC
38 ENDF
39 RETURN
40 END XDOFF
**FORTRAN CALLING SEQUENCE:**

```
CALL XDRDE (DATBUF, DBDCB)
```

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

**INPUTS FROM CALLING SEQUENCE:**

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

**INPUTS FROM XB COMMON:**

```
XB(154) ALFLG6, XB(170) ENDBK, XB(182) TOTSIZ,
XB(201) NOTOC
```

**OUTPUTS IN CALLING SEQUENCE:**

```
DATBUF, DBDCB
```

**OUTPUTS IN XB COMMON**

```
XB(151) ABFLG6, XB(152) ERFLG, XB(165) TOCHOS,
XB(169) TOPBLK, XB(170) ENDBK, XB(171) TOCHNO,
XB(201) NOTOC
```

**INTERNAL XB COMMON USED:**

```
XB(151) ABFLG6 - ABORT FLAG
XB(152) ERFLG - ERROR MESSAGE FLAG
XB(154) ALFLG6 - RESTORE ALL UDB FLAG
XB(155) DEBUG - DEBUG FLAG
XB(159) FILNAME - DB FILE NAME
XB(162) DBLKS - # BLOCKS TO READ
XB(163) DATABK - BLOCK # WHERE DATA ITEM BEGINS
XB(165) DATABK6 - BLOCK # WHERE DATA ITEM BEGINS
XB(166) DATABK - BLOCK # WHERE DATA ITEM BEGINS
XB(167) DATABK - BLOCK # WHERE DATA ITEM BEGINS
XB(170) ENDBK - BLOCK # OF LAST BLOCK READ
XB(171) TOCHNO - # WOODS OF DATA TO MOVE
XB(175) IL - # WORDS OF DATA TO MOVE
```

**OTHERS:**

```
XB(170) TOCHNO - # WOODS OF CURRENT DATA ITEM
XB(170) ETCNT - DATA BASE TOC ENTRY
```

**EXTERNAL USES:**

```
XB(201) NOTOC - # TOC ENTRIES IN TOCBUF
```
1 BEGIN XRDE
2 INITIALIZE FILE INDICES TO INDICATE NO DATA IN DATBUF
3 IF ALLFLG IS ZERO OR IF STORE/RESTORE BIT IS ON THEN
4 IF DATA IS NOT CURRENTLY IN DATBUF THEN
5 CALL READ TO READ 1 BUFFER BEGINNING WITH DATBLK FOR THIS ELEMENT
6 ERREXIT IF READ ERROR TO :ERR1:
7 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
8 ELSE, DATA BEGINS IN DATBUF
9 IF DATA DOES NOT END IN DATBUF THEN
10 CALL XMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
11 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
12 CALL READY TO READ ENOUGH TO FILL DATBUF
13 ERREXIT IF READ ERROR TO :ERR1:
14 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
15 ENDIF
16 BUILD AXA REQUEST TO ALLOCATE AND STORE DATA
17 CALL EXEC TO WRITE DATA TO SAN
18 ERREXIT IF ERROR FROM EXEC TO AXA 2:
19 IF AXA REQUEST BUFFER IS FULL THEN
20 CALL XRDE TO ISSUE REQUEST
21 EXIT XRDE IF ERROR IN XRDE
22 ENDIF
23 ENDIF
24 ENDDO
25 EXIT XRDE
26 :ERR1:
27 CALL XR16 TO CONVERT ERROR CODE TO ASCII
28 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
29 GO TO :ERR3:
30 :ERR2:
31 CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
32 :ERR3:
33 SET ABFLG TO SAY ABORT RESTORE
34 END XRDE
FORTRAN CALLING SEQUENCE:

CALL XBDRF (DATBUF, DBDCB)

XBDRF HANDLES THE RESTORING OF DRDE FILES FROM A MDB/UTDB FILE

INPUTS IN CALLING SEQUENCE:

DATBUF - (INTEGER, 140B WORDS) BUFFER USED TO READ IN
RECORDS OF DRDE FILES FROM DATA BASE FILE

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

INPUTS IN XB COMMON:

XB(152) ERRFL, XB(154) ALLFLG, XB(165) TCOVDS,
XB(169) TOPPFL, XB(170) ENDDFL, XB(171) TOCHDS,
XB(182) TOTSIZ

OUTPUTS IN CALLING SEQUENCE:

DBDCB

OUTPUTS IN XB COMMON:

XB(151) ABFLG

INTERNAL XB COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) ERRFL - ERROR MESSAGE FLAG
XB(154) ALLFLG - RESTORE ALL UTDB FLAG
XB(155) DEBUG - DEBUG FLAG
XB(159) FILENAM - DB FILE NAME
XB(162) DATBLK - BLOCK # WHERE ORDE BEGINS
XB(163) DABTBX - WORD INDEX INTO DATBUF WHERE ORDE BEGINS
XB(164) ERR - ERROR FLAG FOR FMGR CALLS
XB(165) TCOVS - # WORDS OF TOC IN TOCBUF (MOTOC *8) + 8
XB(166) NORKS- # BLOCKS OF UTDB/DRDE TO READ/WRITE
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) TOCHDS- INDEX TO CURRENT TOC ENTRY
XB(172) DBRFL - DRDE FILE NAME
XB(175) IL - # WORDS OF DATA TO READ/WRITE
XB(201) TOCBUF - TOC BUFFER

COMMON USED:
1061 1 BEGIN XRDF
1062 2 DO WHILE THERE ARE TOC ENTRIES TO PROCESS
1063 3 IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
1064 4 CALL XRDFN TO CREATE THE DDE FILE NAME
1065 5 CALL CREAT TO CREATE THE DDE FILE
1066 6 IF THERE WAS A CREATE ERROR THEN
1067 7 IF ERROR FLAG IS OFF THEN
1068 8 SET ERROR FLAG ON
1069 9 CALL XRMS TO DISPLAY MAIN MESSAGE
1070 10 ENDIF
1071 11 CALL XRIG TO CONVERT ERROR CODE TO ASCII
1072 12 CALL EXEC TO DISPLAY DDE NAME AND ERROR CODE
1073 13 ELSE
1074 14 CALL CLOSE TO CLOSE FILE
1075 15 ERREXIT IF CLOSE ERROR TO :ERR1:
1076 16 CALL OPEN TO OPEN DDE FILE AS TYPE 1
1077 17 ERREXIT IF OPEN ERROR TO :ERR1:
1078 18 DO UNTIL ALL BLOCKS OF DDE HAVE BEEN PROCESSED
1079 19 IF DATA FOR DDE IS IN DDATA THEN
1080 20 CALL WRITF DATA TO DDE
1081 21 ERREXIT IF WRITF ERROR TO :ERR1:
1082 22 ELSE
1083 23 CALL READF TO READ NEXT BUFFER OF DATA
1084 24 ERREXIT IF READF ERROR TO :ERR2:
1085 25 SET INDICES INDICATING DATA IN BUFFER
1086 26 ENDF
1087 27 ENDDO
1088 28 CALL CLOSE TO CLOSE DDE FILE
1089 29 ERREXIT IF CLOSE FAILED TO :ERR1:
1090 30 BUILD AMA REQUEST TO ALLOCATE DDE IN AMA
1091 31 IF AMA REQUEST BUFFER IS FULL THEN
1092 32 CALL XRDE TO MAKE REQUEST
1093 33 EXIT XRDF IF XRDE ERROR
1094 34 ENDF
1095 35 ENDF
1096 36 ENDF
1097 2 ENDDO
1098 1 EXIT XRDF
1099 2 :ERR1:
1100 2 CALL PURGE TO PURGE DDE FILE
1101 2 :ERR2:
1102 2 CALL XRMS TO DISPLAY ERROR MESSAGE WITH FILE NAME
1103 2 SET ABFLG TO SAY ABORT RESTORE
1104 1 END XRDF
**FORTRAN CALLING PROCEDURE**

**RECALL DIRECTIVE. A UTDO IS CREATED AND THE CONTENTS OF THE SPECIFIED PRO ARE COPIED TO IT.**

**COMMON KE - CARTE, COMBUF, COMPTHD, FL/GH, LV, TOKENS**

**FILES - XXXX (PRO FILE SPECIFIED)**

**OUTPUT**

**COMMON KE - REQBUF, REQPTR**

**FILES - XXXX (UTDO FILE SPECIFIED)**

**INTERNAL VARIABLES**

**DCBPNO - DBC FOR THE PRO FILE; ALLOCATED IN KE COMMON**

**CONTAINS 1552 WORD BUFFER USED TO READ THE PRO**

**AND TO WRITE THE UTDO**

**DCBUTD - DBC FOR THE UTDO FILE; ALLOCATED IS KS COMMON**
1200 2 :ERRO7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERRO9: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERRO16: ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERRO18: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
1204 2 :ERRO19: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERRO21: ISSUE MESSAGE - "AVG OVERFLOW - XXXX NOT LOGGED"
1206 2 :ERRO22: ISSUE MESSAGE - "XXX ALREADY EXISTS"
1207 2 :ERRO24: ISSUE MESSAGE - "FILE ACCESS ERROR #... XPDB"
1208 2 :RETURN:
1209 2 IF STATUS FLAG INDICATES UTDB FILE IS OPEN, THEN
1210 3 PURGE UTDB
1211 2 ENDIF
1212 2 IF STATUS FLAG INDICATES UTDB IS LOGGED IN ANA, THEN
1213 5 CALL XER TO DELETE UTDB FROM ANA
1214 2 ENDIF
1215 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1216 3 CALL CLOSE FOR PDB FILE
1217 2 ENDIF
1218 1 "ND XDREC
1220 1 **
1221 1 CD0
1222 1 CD0 FORTRAN CALLING PROCEDURE
1223 1 CD0 CALL XDREN
1224 1 CD0
1225 1 CD0
1226 1 CD0
1227 1 CD1
1228 1 CD1 XDREN PROCESSES A USER REQUEST TO RENAME A SEQUENCE TABLE,
1229 1 CD1 INTERFACE TABLE, DATA ELEMENT, DBDE, UTOB, OR PDB. ONLY
1230 1 CD1 THOSE TABLES OR ELEMENTS LOGGED IN THE USERS AWA ARE RENAMED.
1231 1 CD1
1232 1 CD1
1233 1 CD2
1234 1 CD2 INPUT
1235 1 CD2 COMMON XE - CARTRG, COMBUF, COMPTR, QUAL, REPTR, TOKENS
1236 1 CD2
1237 1 CD2
1238 1 CD3
1239 1 CD3 OUTPUT
1240 1 CD3 COMMON XE - REPBUF
1241 1 CD3
1242 1 CD3
1243 1 CD3
1244 1 CD3 INTERNAL VARIABLES
1245 1 CD4
1246 1 CD4 COMMON XS - (2) DBTYPE: TYPE CODE FOR DATA BASE FILES
1247 1 CD4 (3) J: INDEX
1248 1 CD4 (148) IERR: FILE MANAGER ERROR RETURN
1249 1 CD4 (149) IMSE: ERROR MESSAGE NUMBERS
1250 1 CD4 (150) FNAME: NEW FGRE FILE NAME
1251 1 CD4 (153) HNAME: OLD FGRE FILE NAME
1252 1 CD4 (154) ONAME: NEW NAME IN COMBUF
1253 1 CD4 (157) OMNTR: NAME IN COMBUF
1254 1 CD4 (158) TOCTR: TOC ENTRY FOR DATA BASE RENAME
1255 1 CD4 (166) FCATS: DATA CLASS CODE (STORED IN LEFT BYTE)
1256 1 CD4
1257 1 CD4 COMMON XE - (201) IDCB: EXEC BUFFER
1258 1 CD4
1259 1 CD4
1260 1 CD4
1261 1 CD5
1262 1 CD5 EXTERNAL REFERENCES
1263 1 CD5
1264 1 CD5 Routines Used -
1265 1 CD5
1266 1 CD5 EXEC
1267 1 CD5 JANG
1268 1 CD5 KVVT
1269 1 CD5 MAMT
1270 1 CD5 XDRA
1271 1 CD5 XDURD
1272 1 CD5 XREG
1273 1 CD5 XREM
1274 1 CD5 XRS
1275 1 CD5 XRSFR
1276 1 CD5
1280 1 BEGIN XDREM
1281 1 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
1282 1 SAVE POINTER TO CURRENT NAME
1283 1 INCREMENT TO NEXT TOKEN
1284 1 IF TOKEN IS A HYPHEN, THEN
1285 1 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT NAME
1286 1 DECODE CLASS NAME
1287 1 EXIT TO :CLASER: IF CLASS SPECIFIED IS NOT VALID (S,L,D,F,D)
1288 1 ELSE
1289 1 SET CLASS TO BE
1290 1 ENDIF
1291 1 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
1292 1 SAVE POINTER TO NEW NAME
1293 1 IF CLASS IS DATA BASE OR DRE, THEN
1294 1 EXIT TO :TOOLNG: IF NEW NAME IS MORE THAN 4 CHARACTERS
1295 1 ENDIF
1296 1 IF CLASS IS DATA BASE, THEN
1297 1 EXIT TO :TOOLNG: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
1298 1 ENDIF
1299 1 IF CLASS IS INTERFACE TABLE, THEN
1300 1 CALL EXEC TO WRITE/READ NEW NAME
1301 1 BUILD AWA MANAGER REQUEST TO CHANGE NAME IN TOC
1302 1 BUILD AWA MANAGER REQUEST TO STORE NEW NAME IN TABLE
1303 1 CALL XRDE TO ISSUE REQUESTS
1304 1 EXIT TO :CALGET: IF RETURN CODE INDICATES ERROR
1305 1 ELSE
1306 1 IF CLASS IS DATA BASE, THEN
1307 1 BUILD AND ISSUE AWA MANAGER REQUEST FOR TOC ENTRY
1308 1 EXIT TO :NAMERR: IF RETURN INDICATES ERROR
1309 1 EXIT TO :INVALID: IF DATA BASE IS AN NDB
1310 1 ENDIF
1311 1 BUILD AND ISSUE MANAGER REQUEST FOR NAME CHANGE
1312 1 EXIT TO :NAMERR: IF RETURN CODE INDICATES ERROR
1313 1 IF CLASS IS DATA BASE OR DRE, THEN
1314 1 CALL XRPFN TO FORMAT FILE NAMES
1315 1 CALL FILE MANAGER TO CHANGE DISC FILE NAME
1316 1 EXIT TO :UNDO: IF FILE MANAGER RETURNS ERROR
1317 1 IF FILE IS A PDB, THEN
1318 1 CALL XJDDD TO DELETE OLD PDB FROM XPD
1319 1 EXIT TO :NAMENG: IF ERROR RETURNED
1320 1 CALL XJDDA TO ADD NEW PDB NAME TO XPD
1321 1 EXIT TO :TELUSER: IF ERROR RETURNED
1322 1 ENDIF
1323 1 ENDIF
1324 1 ENDIF
1325 1 EXIT XDREM
1326 2 :SYNTAX: CALL XRMSG TO DISPLAY SYNTAX ERROR AND EXIT
1327 2 :TOOLNG: CALL XRMSG ("NEW NAME IS TOO LONG") AND EXIT
1328 2 :CALGET:
1329 2 CALL EXEC TO GET BUFFER CONTAINING NEW TABLE NAME
1330 2 :NAMERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT
1331 2 :CLASER: CALL XRMSG TO DISPLAY CLASS DESIGNATION ERROR AND EXIT
1332 2 :INVALID: CALL XRMSG ("AN NDB CANNOT BE RENAMED") AND EXIT
BEGIN XREQ
CALL XREQ TO PROCESS AWA REQUEST(S)
IF AN ERROR RETURNED BY AWA MANAGER, THEN
  IF ERROR FLAG (ERFLG) IS ZERO, THEN
    TURN ON ERFLG INDICATING THAT MSG 234 HAS BEEN ISSUED
    CALL XRMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
  ENDIF
  CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
  IF CLASS OF ELEMENT IS DDBE, THEN
    CALL PURGE TO DELETE THE FILE
  ELSE, ELEMENT RESIDES IN AWA
  CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
  ENDIF
  IF AWA REQUESTS EXIST IN REQBUF BEYOND FAILING REQUEST, THEN
    MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
  ENDIF
  ELSE SET REQPTR TO 1 INDICATING NO REQUESTS PRESENT
ENDIF
EXIT XREQ
END XREQ
**FORTRAN CALLING PROCEDURE**

**CALL XDBES (DATBUF)**

**INPUT**
- LU
- flags
- ERRPR
- ERBUF
- TOKENS
- CARTAG
- COMPR
- CONBUF

**Calling Sequence**

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

**OUTPUT**
- REBUF

**COMMON XE - ABFLG = ABORT FLAG, NON-ZERO VALUE FROM XDRF, XDBF, XDLIS OR XDBE INDICATES ABORT OF RESTORE**
- XDBF, XDLIS OR XDBE INDICATES ABORT OF RESTORE
- LIST OF ELEMENTS SPECIFIED
- DEBUG = DEBUG BIT OF 'FLAGS' IN X E COMMON
- ERLGK = BLOCK # OF LAST BLOCK READ
- XXRGF = SET NON-ZERO IF MESSAGE XXA IS ISSUED SO THAT IT IS ISSUED ONLY ONCE
- FILNAM = UTOB/MDB FILE NAME
- TOCBUF = UTOB TOC ENTRIES, 8 WORDS EACH, MAX 1200 WORDS
- TCNT = COUNT TOC ENTRY FOR THE UTOB
- TOCNRX = INDEX TO NEXT UTOB TOC ENTRY
- TTSIZ = NO. OF BLOCKS REMAINING IN UTOB FILE

**NOTES**

**Routines Called**
- CLOSE
- EXEC
- OPEN
- READ
- XDLIS
- XDBF
- XDRF
- XDBE
- XRDG
- XR0S
1476 1 BEGIN XRES
1477 2 SET ABFLG TO ZERO
1478 3 ERREXIT IF NEXT TOKEN IS NOT A COMMAND: 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 COMMAND, THEN
1483 8 ERREXIT IF TOKEN IS NOT EOS: ERR04:
1484 9 ENDEF
1485 10 BUILD AWA REQUEST FOR TOC ENTRY RETRIEVE
1486 11 CALL XRES TO PROCESS AWA REQUEST
1487 12 ERREXIT IF AWA REQUEST FAILED: ERR10:
1488 13 ERREXIT IF DATA BASE FOUND IS A PDB: ERR33:
1489 14 IF DATA BASE IS A UDB, THEN
1490 15 CALL XNRMN TO CONSTRUCT QUALIFIED FILE NAME
1491 16 ENDEF
1492 17 CALL OPEN TO OPEN SPECIFIED FILE
1493 18 ERREXIT IF OPEN FAILED: ERR05:
1494 19 ERREXIT IF READ FAILED: ERR06:
1495 20 CALL READ RECORD OF DATA BASE FILE INTO TOCBUF
1496 21 ERREXIT IF READ FAILED: ERR06:
1497 22 INITIALIZE ENDBLK TO NUMBER OF TOC BLOCKS
1498 23 IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1499 24 CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1500 25 ERREXIT IF READ FAILED: ERR06:
1501 26 UPDTE TOTBZ TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY ENDBLK)
1502 27 CLEAR ERROR MESSAGE FLAG (ERRFLG)
1503 28 IF TOKEN IS EOS (i.e., NO LIST OF ELEMENTS), THEN
1504 29 SET ALFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1505 30 ELSE
1506 31 SET ALFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1507 32 CALL XDISL TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1508 33 EXIT XRES IF ABFLG SET BY XDISL
1509 34 ENDEF
1510 35 CALL XDRES TO RESTORE AWA RESIDENT ELEMENTS
1511 36 1 EXIT XRES IF ABFLG SET BY XDRES
1512 37 CALL XDRES TO RESTORE DRE'S
1513 38 EXIT XRES IF ABFLG SET BY XDRES
1514 39 CALL CLOSE TO CLOSE DATA BASE FILE
1515 40 ERREXIT IF CLOSE FAILED: ERR05:
1516 41 DO WHILE AWA REQUESTS REMAIN IN RERBUF
1517 42 CALL XDRES TO PROCESS AWA REQUESTS
1518 43 EXIT XRES IF ABFLG SET BY XDRES
1519 44 ENDOF
1520 1 EXIT XRES

1521 2 :ERR04: CALL XMFS - 'SYNTAX ERROR. EXTRAMEOS DATA'
1522 2 :ERR05: CALL XMFS - 'FILE MANAGER ERROR ... ........'
1523 2 :ERR06: CALL XMFS - 'SYNTAX ERROR. MISSING OR ILLEGAL FIELD'
1524 2 :ERR10: CALL XMFS - '...... NOT FOUND'
1525 2 :ERR33: CALL XMFS - 'CAN NOT RESTORE A PDB'
1526 2 DO UNTIL ALL AWA REQUESTS IN RERBUF HAVE BEEN PROCESSED
1527 3 IF REQUEST IS TO STORE DATA, THEN
1528 4 CALL EXEC TO FREE THE SPECIFIED CLASS NO. AND SAM BUFFER
1529 3 ENDF
**FORTRAN CALLING PROCEDURE**

**INPUT**

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

FILES - XXXX (UTDB FILE SPECIFIED)

**OUTPUT**

COMMON XE - REGBUF, REPTR

FILES - XXXX (PDB FILE SPECIFIED)

**INTERNAL VARIABLES**

DCBPDB - DCB FOR THE PDB FILE; ALLOCATED IN XS COMMON

DEBUTO - DCB FOR THE UTDB FILE; ALLOCATED IN XDB COMMON;

CONTAINS 152 WORD BUFFER USED TO READ THE UTDB AND TO WRITE THE PDB.
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>172</td>
<td>BEGIN YDSAV</td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>ERROR IF &quot;&quot; IS NOT NEXT TOKEN :ERROR:</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>INCREMENT TO NEXT TOKEN</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>ERROR IF NAME IS NOT &quot;NAME&quot; :ERROR:</td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>ERROR IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR6:</td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>ERROR IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR6:</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>RETAIN THIS NAME AS UTOB</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>ERROR IF TOKEN IS NOT &quot;NAME&quot; :ERROR:</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>INCREMENT TO NEXT TOKEN</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>ERROR IF NAME IS MORE THAN 4 CHARACTERS :ERROR:</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>ERROR IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>RETAIN THIS NAME AS PDB</td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>ERROR IF TOKEN IS NOT EOS :ERROR4:</td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>BUILD AWA REQUEST TO RETRIEVE UTOB'S TOC ENTRY</td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>CALL XHEN TO PROCESS AWA REQUESTS</td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>ERROR IF TOC RETRIEVE FAILED :ERROR6:</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>ERROR IF TOC ENTRY DOES NOT INDICATE UTOB :ERROR6:</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>CALL YDOA TO ADD PDB TO XPD</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>ERROR IF FNER ERROR :ERROR:</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>ERROR IF PDB LIMIT EXCEEDED :ERROR2:</td>
<td></td>
</tr>
<tr>
<td>193</td>
<td>ERROR IF OPEN FAILED :ERROR18:</td>
<td></td>
</tr>
<tr>
<td>194</td>
<td>ERROR IF PDB LIMIT EXCEEDED :ERROR2:</td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>SET STATUS FLAG INDICATING PDB LOGGED</td>
<td></td>
</tr>
<tr>
<td>196</td>
<td>CALL XRNBN TO BUILD UTOB FILE NAME</td>
<td></td>
</tr>
<tr>
<td>197</td>
<td>CALL OPEN TO OPEN UTOB FILE (SPECIFYING TYPE 1)</td>
<td></td>
</tr>
<tr>
<td>198</td>
<td>ERROR IF OPEN FAILED :ERROR18:</td>
<td></td>
</tr>
<tr>
<td>199</td>
<td>ERROR IF OPEN FAILED :ERROR18:</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>SET STATUS FLAG INDICATING UTOB FILE OPEN</td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>BUILD AWA REQUEST TO ALLOCATE PDB -- USE SIZE OF UTOB</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>CALL XHEN TO PROCESS AWA REQUESTS</td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>ERROR IF PDB LIMIT EXCEEDED :ERROR2:</td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>ERROR IF OVERFLOW :ERROR2:</td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>SET STATUS FLAG INDICATING PDB LOGGED TO AWA</td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>CALL XRNBN TO BUILD PDB FILE NAME</td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>CALL CREAT TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE</td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>OF UTOB FILE FROM TOC ENTRY</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>ERROR IF CREATE FAILED :ERROR18:</td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>ERROR IF DATA IN UTOB FILE</td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>DO FOR ALL DATA IN UTOB FILE</td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>CALL LREAD TO READ 1 BUFFER OF UTOB DATA</td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>ERROR IF READ FAILED :ERROR04:</td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>CALL WRITB TO WRITE 1 BUFFER TO PDB FILE</td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>ERROR IF WRITE FAILED :ERROR:</td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>ENDDO</td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>CLOSE UTOB FILE</td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>CLOSE PDB FILE</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>EXIT YDSAV</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>ERROR9: ISSUE MESSAGE &quot;SYNTAX ERROR -- ILLEGAL OR MISSING FIELD&quot;</td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>ERROR6: ISSUE MESSAGE &quot;INVALID UTOB FILE NAME ...&quot;</td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>ERROR16: ISSUE MESSAGE &quot;INVALID PDB FILE NAME ...&quot;</td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>ERROR4: ISSUE MESSAGE &quot;SYNTAX ERROR. EXTRANEOUS DATA&quot;</td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>ERROR17: ISSUE MESSAGE &quot;PDB .... ALREADY EXISTS&quot;</td>
<td></td>
</tr>
</tbody>
</table>
1625  2  :ERR10: ISSUE MESSAGE "PRO FILE ACCESS ERROR ... "
1626  2  :ERR00: ISSUE MESSAGE "FILE MANAGER ERROR ........" 
1627  2  :ERR20: ISSUE MESSAGE "AUTHORIZED LIMIT OF ... PRO'S ALREADY REACHED"
1628  2  :ERR21: ISSUE MESSAGE "ANA OVERFLOW. NOT LOGGED"
1630  2  :RETURN:
1631  2  IF STATUS FLAG INDICATES PRO FILE IS OPEN, THEN
1632  3  PURGE PRO FILE
1633  2  ENDIF
1634  2  IF FLAG INDICATES PRO IS IN ANA, THEN
1635  3  CALL REN TO DELETE PRO FROM ANA
1636  3  ENDIF
1637  2  IF FLAG INDICATES UTOB IS OPEN, THEN
1638  3  CLOSE UTOB
1639  2  ENDIF
1640  2  IF FLAG INDICATES PRO IS IN XPRB, THEN
1641  3  CALL XORD TO DELETE PRO FROM XPRB
1642  3  ENDIF
1643  2  END XBSAV
164: 1 C60  F0RTH© CALLING PROCEDURE
164: 2 C60
164: 1 C60  CALL XBSTA (LU)
164: 2 C60
164: 1 C60  **********
165: 1 C60  OUTPUT AMA AND DMA USAGE DATA AND STATISTICS ON AMA COMPACTION
165: 1 C60  PHASES
165: 2 C60
165: 1 C60  **********
165: 1 C60  INPUT
165: 1 C60  LU  - LOGICAL UNIT OF OUTPUT DEVICE
165: 2 C60
165: 1 C60  XD COMMON
165: 2 C60
165: 1 C60  C2  AS2  - TOTAL SIZE OF AMA
165: 2 C60
165: 1 C60  C2  FRE  - AMOUNT OF FREE SPACE IN AMA
165: 2 C60
165: 1 C60  C2  DSZ  - TOTAL SIZE OF DMA (IN 64 WORD SECTORS)
165: 2 C60
165: 1 C60  C2  DFR  - AMOUNT OF FREE SECTORS IN DMA
165: 2 C60
165: 1 C60  C2  NP  - ARRAY OF THREE COMPACTON PHASE COUNTS
165: 2 C60
165: 1 C60  C22  OUTPUT
165: 2 C60
165: 1 C60  C3  USAGE AND STATISTICS TO LU
165: 2 C60
165: 1 C60  **********
165: 1 C60  NOTES
165: 2 C60
165: 1 C60  C5  USES EXEC, XRIG
165: 2 C60
165: 1 C60  **********
165: 1 C60  +
165: 2 C60
165: 1 C60  +
165: 2 C60
165: 1 C60  +
165: 2 C60
165: 1 C60  BEGIX XBSTA
165: 1 C60  2 FORMAT AND PRINT AMA USAGE DATA
165: 1 C60  2 FORMAT AND PRINT DMA USAGE DATA
165: 1 C60  2 FORMAT AND PRINT COMPACTION STATISTICS
165: 1 C60  END XBSTA
1683 1 C6***************
1684 1 C8 CALL XSTD0 (DATBUF)
1685 1 C8***************
1686 1 C8***************
1687 1 C8***************
1688 1 C8***************
1689 1 C8***************
1690 1 C8***************
1691 1 C8***************
1692 1 C8***************
1693 1 C8***************
1694 1 C8***************
1695 1 C8***************
1696 1 C8***************
1697 1 C8***************
1698 1 C8***************
1699 1 C8***************
1700 1 C8***************
1701 1 C8***************
1702 1 C8***************
1703 1 C8***************
1704 1 C8***************
1705 1 C8***************
1706 1 C8***************
1707 1 C8***************
1708 1 C8***************
1709 1 C8***************
1710 1 C8***************
1711 1 C8***************
1712 1 C8***************
1713 1 C8***************
1714 1 C8***************
1715 1 C8***************
1716 1 C8***************
1717 1 C8***************
1718 1 C8***************
1719 1 C8***************
1720 1 C8***************
1721 1 C8***************
1722 1 C8***************
1723 1 C8***************
1724 1 C8***************
1725 1 C8***************
1726 1 C8***************
1727 1 C8***************
1728 1 C8***************
1729 1 C8***************
1730 1 C8***************
1731 1 C8***************
1732 1 C8***************
1733 1 C8***************
1734 1 C8***************
1735 1 C8***************
1237  1 BEGIN XSTO,
1238    2  :SET ABFLG TO ZERO (ABORT FLAG)
1239    3  :ERREXIT IF UTDB NAME IS NOT VALID TO :ERR2:
1240    4  :BR BUILD REQUEST FOR AWA TOC
1241    5  :CALL XRQ TO MAKE MANAGER REQUEST
1242    6  :CALL XEC TO GET AWA TOC
1243    7  :IF SIZE OF TOC > MAXIMUM SIZE TO :ERR2:
1244    8  :DO FOR ALL DATA BASE CLASS ENTRIES
1245    9  :ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1246   10  :ENDDO
1247    1  :ERREXIT ERROR MESSAGE OFF (ERROR MESSAGE FLAG)
1248    2  :SET TOTSZ = 0 (UTDB TOTAL SIZE)
1249    3  :IF WHOLE AWA IS TO BE STORED THEN
1250    4  :DO FOR IT, ST, DE, ODRE AWA TOC ENTRIES
1251    5  :DO FOR EACH ENTRY IN THIS CHAIN
1252    6  :IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1253    7  :PREFIX IS NOT AN AMPERSAND THEN
1254    8  :STORE/RESTORE BIT ON IN TOC ENTRY
1255    9  :INCREMENT TOTSZ BY SIZE OF THIS ELEMENT
1256   10  :ENDIF
1257   11  :ENDIF
1258   12  :ENDIF
1259   13  :ELSE
1260   14  :CALL XOLIS TO PROCESS LIST TO BE STORED
1261   15  :ERREXIT IF ABFLG IS NOT ZERO TO :ERR2:
1262   16  :ENDIF
1263   17  :SET NOTOC = 0 (NUMBER OF UTDB TOC ENTRIES)
1264   18  :DO FOR IT, ST, DE, ODRE CHAINS
1265   19  :DO FOR EACH ENTRY IN THIS CHAIN
1266   20  :IF STORE/RESTORE BIT IS ON THEN
1267   21  :TURN STORE/RESTORE BIT OFF
1268   22  :BUILD UTDB TOC ENTRY
1269   23  :INCREMENT NOTOC BY 1
1270   24  :ENDIF
1271   25  :ENDIF
1272   26  :ENDIF
1273   27  :ERREXIT IF THERE ARE NO UTDB TOC ENTRIES (NOTOC=0) TO :ERR2:
1274   28  :COMPUTE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1275   29  :CALL XP 'X' TO CREATE FILE NAME
1276   30  :CALL TO CREATE UTDB FOR TOTSZ
1277   31  :ERREXIT ERROR IN CREATE TO :ERR3:
1278   32  :CALL XWRT TO WRITE UTDB FILE
1279   33  :ERREXIT IF ABFLG IS 4 (ORDE LARGER THAN SPECIFIED) TO :ERR1:
1280   34  :ERREXIT IF ABFLG IS 3 (ORDE FILE ERROR) TO:ERR4:
1281   35  :ERREXIT IF ABFLG IS 2 (UTDB FILE ERROR) TO:ERR3:
1282   36  :CALL WRIT TO WRITE TOC RECORDS AT RECORD 1
1283   37  :ERREXIT IF ERROR IN WRIT TO :ERR3:
1284   38  :CALL CLOSE TO CLOSE UTDB FILE
1285   39  :ERREXIT IF ERROR IN CLOSE TO :ERR3:
1286   40  :BR BUILD REQUEST TO ALLOCATE UTDB IN AWA
1287   41  :CALL XRQ TO MAKE REQUEST
1288   42  :ERREXIT IF AWA OVERFLOW TO :ERR1:
1289   43  :EXIT XSTO
1290   44  :ERR1:
1291   45  :CALL XRMG TO DISPLAY MSGNO
1292   46  :GO TO :ERR4:
1293   47  :ERR2:
1294   48  :ERR3:
1295   49  :ERR4:
1805 1 C00  FORTRAN CALLING PROCEDURE
1807 1 C00   CALL XDOTC
1809 1 C00   **********
1810 1 C01   BODY AW/AW TABLE OF CONTENTS DIRECTIVE HANDLER. XDOTC INTER-
1812 1 C01   PRETS THE TOC DIRECTIVE, RETRIEVES THE INDICATED TOC (AW OR
1814 1 C01   DATABASE FILE) AND FORMATS AND OUTPUTS THE REQUESTED ENTRIES.
1815 1 C00   **********
1816 1 C02   INPUT
1818 1 C02   X COMMON - COMBUF, COMPTR, FLAGS, LU
1820 1 C02   MANAGER - AW HEADER AND TOC (SEE MODULE XMWA)
1822 1 C02   FILES - DATABASE FILES AS APPROPRIATE
1823 1 C02   **********
1825 1 C03   OUTPUT
1827 1 C03   X COMMON - COMBUF, REBUF, REPTR
1829 1 C03   PRINTED OUTPUT - AW OR UDB TOC DISPLAY
1831 1 C03   **********
1833 1 C04   LOCAL VARIABLES
1835 1 C04   AZ - TOTAL ALLOCATABLE SIZE OF AW (SEE MODULE XMWA)
1837 1 C04   ANA - FIRST PORTION OF AW (HEADER AND TOC)
1839 1 C04   CLASS - ARRAY OF ALLOWABLE CLASS DESIGNATORS FOR TOC DIRECTIVE
1841 1 C04   CLES - CLASS CHAIN NUMBER BEING PROCESSED PLUS ONE
1843 1 C04   CODE - CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'
1845 1 C04   EOC - END OF TOC CHAIN DESIGNATOR (-32768)
1847 1 C04   FRE - CURRENT AMOUNT OF FREE SPACE IN AW (SEE MODULE XMWA)
1849 1 C04   HD - ARRAY OF TOC CHAIN HEADS (SEE MODULE XMWA)
1851 1 C04   IDIM - TOC ENTRY DIM FIELD (WORD 8)
1853 1 C04   LINE - BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE
1855 1 C04   MULT - ARRAY OF MULTIPLIERS USED TO DETERMINE NUMBER OF WORDS
1857 1 C04   IN COLUMNS OF TWO DIMENSIONAL DATA ELEMENTS (FUNCTION OF
1859 1 C04   DATA TYPE)
1861 1 C04   OPTION - ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY
1863 1 C04   ORG - ORIGIN ADDRESS OF AW (SEE MODULE XMWA)
1865 1 C04   POS - VALUE OF 2 OR 22 INDICATING FIRST OR SECOND POSITION IN
1867 1 C04   TOC DISPLAY LINE
1869 1 C04   SIZE - TOC ENTRY SIZE FIELD (WORD 7)
1871 1 C04   TOCMAX - MAXIMUM SIZE OF TOC WHICH CAN BE TOTALLY ACCOMMODATED BY
1873 1 C04   INTERNAL BUFFER
1875 1 C04   TYPE - TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)
1877 1 C04   TYPEID - ARRAY OF TYPE ID CODES FOR DE AND ORDE DISPLAYS
1879 1 C04   UNIT - LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY
1881 1 C04   **********
1883 1 C05   NOTES
1885 1 C05   USES CLOSE, EXEC, IAND, IXOR, KCYT, OPEN, READF, XDSTA, XRQ,
1887 1 C05   XRIG, XRMOV, XRSG, XROG, XRFN, XUCBG
1889 1 C05   IN THE EVENT THE COMPACTED AW TOC AND HEADER DATA WILL NOT FIT
1891 1 C05   IN THE ALLOCATED BUFFER SPACE, AS MUCH OF EACH CHAIN AS POSSIBLE
1893 1 C05   WILL BE DISPLAYED UNTIL THE LINK FIELDS LEAD BEYOND THE BUFFER.
1895 1 C05   IN THE EVENT THE TOC AND HEADER DATA WILL NOT FIT
1897 1 C05   IT IS ASSUMED THAT XDOTC AND XDSTO USE THE SAME SIZE BUFFER FOR
1899 1 C05   TOC MANIPULATION; THUS, A DATABASE FILE MAY NOT HAVE A TOC TOO
1901 1 C05   LARGE FOR THE XDOTC INTERNAL BUFFER.
1866 1 BECOME XDTOC
1867 2 SCAN FOR OUTPUT OF ALL USER CLASSES FROM AMA TO TERMINAL
1868 2 IF NEXT TOKEN IS NOT EOS
1869 THEN
1870 3 IF TOKEN IS A HYPHEN
1871 THEN
1872 4 INCREMENT TO NEXT TOKEN
1873 IF TOKEN IS NOT A CHARACTER T
1874 THEN
1875 EXIT TO :ERROR: IF TOKEN IS NOT A CHARACTER P
1876 SET OUTPUT UNIT FOR LINE PRINTER
1877 ENDIF
1878 ENDIF
1879 IF TOKEN IS NOT EOS
1880 THEN
1881 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1882 INCREMENT TO NEXT TOKEN
1883 IF TOKEN IS NOT A COMMA
1884 THEN
1885 EXIT TO :ERROR: IF TOKEN DOES NOT INDICATE A VALID CLASS (0, F, I, S OR B)
1886 SET OPTION FOR INDICATED CLASS
1887 INCREMENT TO NEXT TOKEN
1888 ENDIF
1889 IF TOKEN IS NOT EOS
1890 THEN
1891 EXIT TO :ERROR: IF TOKEN IS NOT A COMMA
1892 INCREMENT TO NEXT TOKEN
1893 EXIT TO :ERROR: IF TOKEN IS NOT A THREE CHARACTER NAME
1894 INCREMENT TO NEXT TOKEN
1895 EXIT TO :ERROR: IF TOKEN IS NOT EOS
1896 CALL XREQ TO RETRIEVE DATA BASE FILE TOC ENTRY
1897 EXIT TO :ERROR: IF NOT LOGGED IN TOC
1898 GET TYPE AND SET NAME IN HEADER
1899 CONSTRUCT FILE NAME
2000 READ FIRST DATA BASE FILE TOC RECORD
2001 INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AMA TOC
2002 READ SUBSEQUENT TOC RECORDS
2003 EXIT TO :ERROR: IF FILE ACCESS FAILS
2004 DO FOR EACH NON-EMPTY TOC CHAIN
2005 INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
2006 IF NOT FIRST ENTRY IN TOC, I.E., A PREVIOUS NON-NULL CHAIN EXISTED
2007 THEN
2008 MARK PREVIOUS ENTRY AS AN END OF CHAIN
2009 ENDIF
2010 EMDO
2011 DO FOR EACH ENTRY IN TOC
2012 IF NOT MARKED AS AN END OF CHAIN
2013 THEN
2014 STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
2015 ENDIF
2016 EMDO
2017 EXCLUDE CHAIN 8 (DATA BASE FILES) FROM DISPLAY
2018 ENDIF
2019 ENDIF
2020 EMDO
2021 IF REFERENCING AMA TOC
2022 THEN
2023 CALL XREQ TO RETRIEVE AMA TOC
2024
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
ENDO
ENDO
IF PROCESSING AWA TOC
THEN
CALL XSTA TO DISPLAY AWA USAGE STATISTICS
ENDIF
EXIT XDTOC
:ERROR5: EXIT WITH INVALID OUTPUT DEVICE ID
:ERROR4: EXIT WITH SYNTAX ERROR
:ERROR5: EXIT WITH INVALID CLASS DESIGNATOR
:ERROR6: EXIT WITH INVALID UTDB FILE NAME
:ERROR7: EXIT WITH UTDB FILE ACCESS ERROR
1 END XDTOC
1 BEGIN XDWRT
2 SET WRDNO = 0 (WORD INDEX INTO DAWRE WHERE ELEMENT BEGINS)
3 SET NORES = 0 (NUMBER OF AWA REQUESTS IN RENBUFF)
4 DO FOR ALL UDB TOC ENTRIES UNOT CLASS IS DRE
5 BUILD REQUEST FOR DATA FROM AWA
6 INCREMENT NORES BY 1
7 IF REQUEST BUFFER IS FULL (NORES=8) THEN
8 PERFORM READ TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
9 ENDF
10 IF THERE ARE REMAINING REQUESTS (NORES>0) THEN
11 SET NEXT REQUEST TO SAY END OF REQUEST LIST
12 PERFORM READ TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
13 IF THERE IS A PARTIAL DATA RECORD LEFT (WRDNO>128) THEN
14 PERFORM WRITE TO OUTPUT DATA TO UDB FILE
15 ENDF
16 DO FOR EACH DRE UDB TOC ENTRY
17 STORE DAWRE IN UDB TOC ENTRY
18 CALL OPENM TO CREATE FILE NAME
19 IF DRE FILE IS TYPE 3 THEN
20 CALL OPEN TO OPEN FILE AS CORRECT TYPE
21 ERROR IF OPEN ERROR TO :ERR1:
22 DJ UNTIL EOF IS READ
23 CALL READ TO READ 1 RECORD
24 ERROR IF READ ERROR TO :ERR1:
25 STORE RECORD LENGTH AT FRONT AND REAR OF DATA
26 INCREMENT WRDNO BY LENGTH + 2
27 IF THERE IS ENOUGH DATA TO WRITE (WRDNO>128) THEN
28 PERFORM WRITE TO OUTPUT DATA TO UDB FILE
29 ENDF
30 IF THERE IS REMAINING DATA (WRDNO>1) THEN
31 PERFORM WRITE TO OUTPUT DATA TO UDB FILE
32 ENDF
33 SET DAWRE TO NEXT AVAILABLE RECORD FOR DATA
34 ELSE
35 CALL OPEN TO OPEN FILE AS TYPE 1
36 ERROR IF OPEN ERROR TO :ERR1:
37 COMPUTE TOTAL SIZE OF FILE IN WORDS
38 DO UNTIL ALL DATA IS COPIED TO UDB (SIZE=0)
39 IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
40 SET LENGTH = SIZE
41 ENDIF
42 CALL READ TO READ LENGTH DATA
43 ERROR IF READ ERROR TO :ERR1:
44 CALL WRITE TO WRITE LENGTH DATA
45 ERROR IF WRITE ERROR TO :ERR5:
46 INCREMENT DATAREC BY NUMBER OF RECORDS WRITTEN
47 DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
48 ENDF
49 ENDF
50 CALL CLOSE TO CLOSE DRE FILE
51 ERROR IF CLOSE ERROR TO :ERR1:
52 ENDF
53 EXIT XDWRT
1) CALL XERBS (THREE WORD ARRAY CONTAINING 'XECAL')
2) FDS EXECUTIVE INITIALIZATION SEGMENT MAIN ROUTINE
3) INPUT
4) COMMON XE - SUBST
5) COMMON XB - INITIALIZED ACCORDING TO THE VALUE OF SUBST (SEE APPROPRIATE INITIALIZATION SUBROUTINE)
6) NOTE!
7) USES XEINO, XEINI, XEINS, XEINS, XERN
8) ********
9) CASE (:GLOBAL:, :DIRECT:, :EXECUT:, :SEREDT:, :INTEDT:) SUBST (-1 TO 3)
10) :GLOBAL: CALL XEINO TO INITIALIZE GLOBAL COMMON
11) :DIRECT: CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES
12) :EXECUT: CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL
13) :SEREDT: CALL XEINS TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING
14) :INTEDT: CALL XEINS TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING
15) END CASE
16) END XECAL
FORTRAN CALLING PROCEDURE

CALL XEIND

INITIALIZE XB COMMON FOR DIRECTIVE LEVEL

INPUT

COMMON XB - CARTAG, FLAGS

XOPRM - FDS DIRECTIVE PROMPT FILE

OUTPUT

COMMON XB - DIRECT, NUMDIR

NOTES

USES CLOSE, EXEC, OPEN, PRMT, READF, XREXT, XRIG, XRMG, XVABN

BEGIN XEIND

READ FDS DIRECTIVE PROMPT FILE

STORE DIRECTIVES IN DYNAMIC COMMON

END XEIND
FORTRAN CALLING PROCEDURE

CALL XEIME

************
INITIALIZE XE COMMON FOR FDS START UP

************
INPUT
COMMON XE - CARTS, FLAGS, LU, QUAL
XLIBD - LIBRARY DIRECTORY FILE
XPDB - PDB AND RDB DIRECTORY

************
OUTPUT
COMMON XE - MASSTA, NOPROC, REBUFF, REOPTR, SUBSTA, TKNLNG, TOKENS
COMMON XB - INITIALIZED FOR DIRECTIV: LEVEL (SEE XEIRD)
AWA - LIBRARY DIRECTORY NAME TABLE
MOD'S AND PDB'S LOGGED INTO AWA

************
LOCAL
IDSP - DISPLACEMENT (NEGATIVE) BETWEEN ORIGIN OF XE COMMON AND
KEYRD - ID SEGMENT POINTER LIST
LEN - LENGTH OF LIBRARY DIRECTORY RECORD BEING PROCESSED
NAMDSP - DISPLACEMENT (NEGATIVE) BETWEEN ORIGIN OF XE COMMON AND
ORG - MAPPED ADDRESS OF ORIGIN OF XE COMMON

************
NOTES
USES CLOSE, EXEC, IAMD, OPEN, FRMT, READF, XEINI, XRCPR, XREQ,
XRET, XRS, XRLD, XRNOV, XRMNG, XURBG, XVARN

************
1 BEGIN XEINE
2 INITIALIZE COMMON TO ZEROS
3 SET FILE (ART-DEC NUMBERS
4 SET STATES TO DIRECTIVE LEVEL
5 SET TKNMLG
6 INITIALIZE TOKENS
7 READ LIBRARY DIRECTORY FIRST RECORD
8 EXIT TO :ERROR1: IF READ FAILED
9 STORE NUMBER OF PROCESSORS IN NPROC
10 READ PROCESSOR NAME RECORD
11 EXIT TO :ERROR1: IF READ FAILED
12 CALL XER0 TO ALLOCATE AND STORE PROCESSOR DIRECTORY IN AWA
13 EXIT TO :ERROR1: IF REQUEST FAILED
14 DO FOR EACH ID SEGMENT IN SYSTEM
15 DO FOR EACH PROCESSOR IN LIBRARY DIRECTORY
16 IF NAMES ARE THE SAME
17 THEN
18 NOTE EXISTANCE OF ID
19 ENDIF
20 ENDDO
21 IF ANY PROCESSORS NOT MARKED
22 THEN
23 LIST PROCESSORS MISSING ID SEGMENTS
24 END
25 ENDM WITH INITIALIZATION FAILURE
26 ENDDO
27 CALL OPEN TO OPEN PDB/MDB DIRECTORY (XPOB)
28 EXIT TO :ERROR1: IF OPEN FAILED
29 CALL READ TO READE 1ST RECORD OF XPOB (LIST OF MDB'S)
30 EXIT TO :ERROR1: IF READ FAILED
31 PERFORM DLOG TO LOG MDB'S IN AWA
32 CALCULATE RECORD NO. OF MDB'S FOR THIS QUALIFIER
33 CALL READ TO READ THAT RECORD OF XPOB
34 CALCULATE CORRECT INDEX INTO BUFFER (EACH RECORD IS FOR 2 QUALIFIERS)
35 CALL CLOSE TO CLOSE XPOB
36 PERFORM DLOG TO LOG MDB'S IN AWA
37 IF THERE ARE REQUESTS IN THE AWA REQUEST BUFFER, THEN
38 CALL XER0 TO PROCESS THE AWA REQUESTS
39 ENDIF
40 CALL XEINO TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVE LEVEL
41 EXIT XEINE
42 BEGIN DLOG
43 DO WHILE THERE ARE DATA BASE FILES TO BE LOGGED
44 BUILD AN ENTRY IN AWA REQUEST BUFFER TO ALLOCATE THIS DATA BASE
45 IF 8 AWA REQUESTS HAVE BEEN BUILT, THEN
46 CALL XER0 TO PROCESS AWA REQUESTS
47 ENDIF
48 EXIT TO :ERROR1: IF A REQUEST FAILED
49 ENDDO
50 END DLOG
51 :ERROR1: LIBRARY INITIALIZATION ERROR TERMINATION
52 END XEINE
FORTRAN CALLING PROCEDURE

CALL XEINI

*********

initialize xe and xb common for interface table editing

*********

common xe -

MASTA = MASTER STATE

tokens = lexical token values

NPROC = number of entries in library directory

COMTP = index of next token in combuf

COMBUF = communications buffer

PRCNAM = processor name from execution controller

common xb -

LIBRARY -

INTNAM = old interface table to be edited

<xxxxx = default interface table

>xxxxx = prompt table

*********

common xe -

SUBSTA = substate -- set to 1 if init. error

RBUF = request buffer for awa inputs

PRCNAM = processor name on inte directive

common xb -

WKBLNG = length of wkbuf (constant)

WKBUF = working buffer containing inte. table with

short prompts

direct = supported inte directives

HNUMIR = no. of entries in direct

MENTAB = new inte. table name

HNUMARG = no. of arguments

HARG = no. of words in spc. and header of wkbuf

LITLEN = no. of words in literal area

LITTPX = index to start of literal area

ISIZES = array mapping arg. type to effective length

*********

notes -

<xxxxx = default interface table for this processor

>xxxxx = prompt table for this processor

*********

uses files -

*********

uses routines -

*********

begin xeini

if called as a result inte directive, then
This page contains a block of code that appears to be a computer program written in a low-level language, possibly assembly. The code includes conditional statements, loops, and calls to functions or libraries. The syntax and structure suggest it is part of a larger program, possibly for data processing or file management.

Here is a transcription of the code:

```
220 3 POSITION TO 1ST TOKEN AFTER 'INTC'
221 3 ERXMIT IF TOKEN IS NOT ',' :ERR1:
222 3 INCREMENT TO NEXT TOKEN
223 3 ERXMIT IF TOKEN IS NOT A NAME :ERR2:
224 3 SET PRCNAM TO THIS NAME
225 3 INCREMENT TO NEXT TOKEN
226 3 SET INTNAM TO D
227 3 SET NEWTAB TO 'EINTAB:
228 4 IF TOKEN IS NOT EOS (END-OF-STATEMENT), THEN
229 4 ERXMIT IF TOKEN IS NOT A COMM, :ERR2:
230 4 INCREMENT TO NEXT TOKEN
231 4 IF TOKEN IS NOT EOS, THEN
232 5 IF TOKEN IS A NAME, THEN
233 6 SET INTNAM TO THE NAME
234 6 INCREMENT TO NEXT TOKEN
235 6 ENDIF
236 5 IF TOKEN IS NOT EOS, THEN
237 6 ERXMIT IF TOKEN IS NOT A COMM, :ERR2:
238 6 INCREMENT TO NEXT TOKEN
239 6 IF TOKEN IS A NAME, THEN
240 7 SET NEWTAB TO THE NAME
241 7 INCREMENT TO NEXT TOKEN
242 6 ENDIF
243 5 ERXMIT IF TOKEN IS NOT EOS :ERR2:
244 5 ENDIF
245 4 ENDIF
246 2 ELSE
247 3 INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER
248 3 NEWTAB IS SET TO 'EINTAB:
249 3 ISSUE MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED
250 2 ENDIF
251 2 MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE
252 2 ERXMIT IF REQUEST IS UNSUCCESSFUL :ERR11:
253 2 START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR
254 2 EXIT IF PROCESSOR PRCNAM IS FOUND
255 2 ORELSE
256 3 INCREMENT TO NEXT ENTRY
257 3 ENDOLOOP
258 2 ERXMIT :ERRB:
259 2 ENDS S EARCH
260 2 IF VERS = VERSION NO. OF DIRECTORY ENTRY FOUND
261 3 IF INTNAM = 0, THEN
262 3 CONSTRUCT THE NAME OF THE DEFAULT INTERFACE TABLE AS "PRCNAM"
263 3 OPEN THE DEFAULT INTERFACE TABLE FILE
264 3 ERXMIT IF OPEN FAILED :ERR2:
265 3 READ THE DEFAULT INTERFACE TABLE
266 3 ERXMIT IF THE READ FAILED :ERR16:
267 3 READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM
268 3 OF WKBUP
269 3 OK
270 3 ERXMIT IF READ FAILED :ERR14:
271 3 ELSE
272 3 MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE
273 3 ERXMIT IF ERROR OR COULD NOT FIND :ERR4:
274 2 ENDIF
275 2 ERXMIT IF VERS .ME, VERSION NO. OF TABLE TO BE EDITED :ERR10:
276 2 CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"
277 2 OPEN THE PROMPT TABLE
278 2 ERXMIT IF OPEN FAILED :ERR15:
```
READ 2ND RECORD OF THIS PROMPT TABLE
ERREXIT IF READ FAILED :ERR17:
MOVE LITERAL DATA ENTRIES UP FROM BOTTOM OF WKBUF TO AREA FOLLOWING
THE PROMPT TABLE
THIS IS DONE BY LOOPING THROUGH ALL ARGUMENTS TO FIND THE
LITERAL DISPL. THAT MATCH EACH LITERAL DATA ENTRY
THE TYPE OF EACH ARGUMENT FOUND TO HAVE LITERAL DATA IS USED IN THE
MOVING PROCESS TO DETERMINE THE NUMBER OF WORDS IN EACH DATA ELEMENT
AS WELL AS THE NUMBER OF WORDS TO LEAVE FOR INCOMPLETE ELEMENTS.
EXIT XEINI

:ERR2: "SYNTAX ERROR ... "
:ERR4: "INTNAM" NOT FOUND
:ERR6: INVALID NAME FIELD
:ERR8: "PRCNAM" NOT IN DIRECTORY
:ERR10: VERSION OF INTNAM DOES MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
304  1 C00  FORTRAN CALLING PROCEDURE
305  1 C00
306  1 C00
307  1 C00  CALL XEINS
308  1 C00
309  1 C00*********
310  1 C01
311  1 C01  INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING
312  1 C01
313  1 C01*********
314  1 C02
315  1 C02  INPUT
316  1 C02
317  1 C02  COMMON XE - CARTRG, COMBUF, COMPTR, LU, NOPROC, TOKENS
318  1 C02
319  1 C02*********
320  1 C03
321  1 C03  OUTPUT
322  1 C03
323  1 C03  COMMON XE - REQBUF, REQPTR, SUBSTA,
324  1 C03
325  1 C03  COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,
326  1 C03  OLDTAB, PRMTAB, WKBLNG, WKBUF, XLIBD
327  1 C03
328  1 C03*********
329  1 C05
330  1 C05  NOTES
331  1 C05
332  1 C05  USES FILES - >XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE
333  1 C05
334  1 C05  USES ROUTINES
335  1 C05
336  1 C05  EXEC
337  1 C05  IAND
338  1 C05  OPEN
339  1 C05  READF
340  1 C05  XRCPR
341  1 C05  XREQ
342  1 C05  XREXT
343  1 C05  XMMOV
344  1 C05  XRMG
345  1 C05  XUDBG
346  1 C05
347  1 C05*********
1 BEGIN XEMS
2 BUILD AWA REQUEST TO RETRIEVE "XLIBD"
3 EXIT IF TOKEN IS NOT COMMA :ERROR2:
4 INCREMENT TO NEXT TOKEN
5 IF TOKEN IS NAME, THEN
6 BUILD AWA REQUEST TO RETRIEVE OLDTAB
7 RETAIN THIS NAME AS OLDTAB
8 INCREMENT TO NEXT TOKEN ELSE
9 SET OLDtab TO ZERO
10 ENDIF
11 EXIT IF TOKEN IS NOT COMMA :ERROR2:
12 INCREMENT TO NEXT TOKEN
13 EXIT IF TOKEN IS NOT NAME :ERROR2:
14 IF NEWTAB NOT EQUAL TO OLDTAB, THEN
15 BUILD AWA REQUEST TO VERIFY EXISTENCE OF NEWTAB
16 ENDIF
17 CALL XERQ TO PROCESS AWA REQUEST(S)
18 IF NEWTAB NOT EQUAL TO OLDTAB, THEN
19 EXIT IF NO ERROR FROM XERQ, :ERROR12:
20 ENDIF
21 IF OLDtab NOT ZERO, THEN
22 EXIT IF 2ND REQUEST (RETRIEVE OLDtab) FAILED :ERROR4:
23 SET PROMPT MODE AS UPDATE
24 SET NO. ENTRIES AS OLDtab SIZE / 7
25 READ OLDtab INTO WORKING BUFFER
26 EXIT IF FIRST REQUEST (RETRIEVE "XLIBD") FAILED :ERROR1:
27 READ "XLIBD" INTO COMMON
28 ELSE
29 SET PROMPT MODE AS CREATE
30 SET NO. ENTRIES TO ZERO
31 ENDIF
32 OPEN, READ AND CLOSE FILE XSPRM
33 SET COUNT AND SEREDIT DIRECTIVES INTO XEMS COMMON
34 SET SUBSTATE FLAG TO Seq. Edit. (=2)
35 EXIT XEMS
36 EXIT XEMS
37 2 :ERROR1: CALL XEMSG - 'INITIALIZATION ERROR ....'
38 2 :ERROR2: CALL XEMSG - 'SYNTAX ERROR'
39 2 :ERROR4: DEFAULT MESSAGE TO '....NOT FOUND'
40 IF ERROR WAS NO AWA SPACE THEN
41 SET MSG TO '....NO AWA SPACE'
42 ENDIF
43 CALL XEMSG TO DISPLAY MESSAGE
44 2 :ERROR12: CALL XEMSG - '.... ALREADY EXISTS'
45 1 END XEMS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XB - LITPTR, NUMARGS, WKBLEN, WKBUF

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

OUTPUT

COMMON XB - LITPTR, LITOWN, NARG, WBUF

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

NOTES

USES Routines
1 BEGIN XEINT
2 DO UNTIL ALL LITERAL ENTRIES HAVE BEEN PROCESSED
3 START SEARCH UNTIL ALL ARGUMENTS SEARCHED
4 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
5 D-BIT IS ON
6 BUILD 3 WORD LITERAL ENTRY (3, I-SUB, J-SUB)
7 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
8 D-BIT IS ON
9 IF ARGUMENT IS COMPLETE (C-BIT ON), THEN
10 CREATE BIT MASK WORDS IN NEW LITERAL ENTRY AREA
11 ELSE
12 MOVE BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA
13 ENDDF
14 DO UNTIL ALL BITS OF BIT MASK PROCESSED
15 DETERMINE NUMBER OF CONTIGUOUS BITS ON (OR OFF)
16 AND MOVE CORRESPONDING NUMBER OF DATA WORDS
17 (OR O'S) INTO NEW LITERAL ENTRY AREA
18 ENDDO
19 OR ELSE
20 INCREMENT TO NEXT ARGUMENT
21 ENDDOLOOP
22 ERROR IF NOT CALLED BY LIBRARY MAINTENANCE :ERR5:
23 SKIP TO NEXT LITERAL ENTRY
24 ENDSCEARCH
25 1 EXIT TO :RETURN:
26 1 :ERR5: CALL XRMG - 'SYSTEM INITIALIZATION ERR 5 '
27 2 :RETURN:
28 1 END XEINT
FORTRAN CALLING PROCEDURE

CALL XEIMX

initialize xe and xb common for execution controller

common xe - combuf, compr, flags, lu, massta, nopr, tokens

aux - sequence table, library directory name table

common xe - compr, massta, rebuf, reptr, segend, segnam, seqptr

serstr, substa

common xb - libd, nopr, seng, wkblng, wkbuf

notes

uses exec, prtm, xreq, xrext, xrig, xrmov, xmsg, xudeg, xvabn
BEGIN XEINX
IF INITIALIZE FROM DIRECTIVE
    IF DIRECTIVE IS NAME
        THEN
            EXIT TO :ERROR2: IF NEXT TOKEN IS NOT EOS
    ELSE
        IF DIRECTIVE IS AUTO
            THEN
                IF TOKEN IS A Coroutine
                    THEN
                        INCREMENT TO NEXT TOKEN
                EXIT TO :ERROR3: IF TOKEN IS NOT THE NAME 'T'
        CHANGE EXECUTION CONTROL MODE TO AUTO-T
        INCREMENT TO NEXT TOKEN
    ENDIF
ENDIF
EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A COMMA
INCREMENT TO NEXT TOKEN
EXIT TO :ERROR2: IF NEXT TOKEN IS NOT A NAME
STORE NAME IN SEGNM
CALL XREG TO RETRIEVE SEQUENCE TABLE
EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
INCREMENT TO NEXT TOKEN
SET SERSTA TO FIRST SEQUENCE NUMBER
SET SEREND TO LAST SEQUENCE NUMBER
IF TOKEN NOT EOS
THEN
EXIT TO :ERROR2: IF TOKEN NOT A COMMA
INCREMENT TO NEXT TOKEN
IF TOKEN IS AN INTEGER
THEN
STORE STARTING RANGE NUMBER
SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
EXIT TO :ERROR13: IF NOT FOUND
INCREMENT TO NEXT TOKEN
ENDIF
IF TOKEN NOT EOS
THEN
EXIT TO :ERROR2: IF TOKEN NOT A COMMA
INCREMENT TO NEXT TOKEN
EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
ERREXIT TO :ERROR5: IF TENDING SEGN < BEGINNING SEGN
SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
EXIT TO :ERROR13: IF NOT FOUND
INCREMENT TO NEXT TOKEN
EXIT TO :ERROR2: IF TOKEN NO. EOS
ENDIF
SET SERPTR TO SERSTA
ENDIF
ENDIF
INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
EXIT TO :ERROR1: IF INITIALIZATION FAILS
1 EXIT KEINF
:ERROR1: INITIALIZATION FAILURE TERMINATION
:ERROR2: SET SUBSTA 1/0 DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
XEINX
551 2 :ERROR3: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRANCE 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 XE1NX
FORTRAN CALLING PROCEDURE:

CALL XELDS (SEGNN) TO CALL SEGMENT FROM MAIN
CALL XERTN TO RETURN TO MAIN PROGRAM

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

Inputs in calling sequence:

SEGNN - INTEGER, 3 WORDS) ARRAY CONTAINING NAME OF THE
SEGMENT TO BE LOADED

Subroutines and functions called:
EXEC

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

BEGIN XELDS

CALL .ENTRY TO RETRIEVE PARAMETERS AND RETURN ADDRESS
MOVE THE ADDRESS OF THE SEGMENT NAME INTO EXEC PARAMETER LIST
CALL EXEC TO LOAD AND PASS CONTROL TO SEGMENT

BEGIN XERTN
RETURN TO MAIN PROGRAM
END XELDS
FORTAN CALLING PROCEDURE

CALL XESC

**Comment**

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

**Comment**

INPUT

X COMMON - CARTRG

ANA - EDOSTK, ESCTBD (SEE DO AND SCAN PROCESSORS)

OUTPUT

X COMMON - REBUF, REPTR

X COMMON - SCRATCH

**Comment**

EXTERNAL ROUTINES

CLOSE, EXEC, PURGE, XREQ, XRIG, XRMOV, XRUNG

**Comment**

BUILD REQUESTS TO DELETE EDOSTK AND RETRIEVE ESCTBD THEN DELETE IT

CALL XREQ TO ATTEMPT REQUESTS

ISSUE MESSAGE XEB FOR SUCCESSFUL DELETES

IF XREQ COMPLETED REQUESTS, I.E., ESCTBD EXISTED, THEN DO FOR EACH SCAN CONTROL ENTRY IN ESCTBD

CALL CLOSE TO CLOSE THE DATA BOX FILE ASSOCIATED WITH THIS SCAN

CALL PURGE TO PURGE THE FILE

IF PURGE RETURNED AN ERROR, THEN

CALL XRMSS TO DISPLAY WARNING MESSAGE

ENDIF

BUILD REQUEST TO DELETE DATA BOX FROM TOC

ENDDO

CALL XREQ TO DELETE DATA BOXES FROM ANA TOC

ENDIF

END XESC
1 CD9  FDS EXECUTIVE TASK MAIN PROGRAM. SCHEDULED BY FDS MANAGER.
1 CD1  ONE PROGRAM PER SIGNED-ON USER
1 CD1  
1 CD1  656
1 CD1  ************
1 CD1  659
1 CD1  INPUT
1 CD1  661
1 CD2  SCHEDULING PARAMETERS - LU, CLASNO, OUAL, FLAGS (SEE XE COMMON)
1 CD2  662
1 CD2  ************
1 CD2  663
1 CD3  OUTPUT
1 CD3  664
1 CD5  COMMON XE - COMBUF, COMPTR, NASSTA, SUBSTA, PLUS XEINE
1 CD5  665
1 CD5  INITIALIZATION
1 CD5  666
1 CD5  COMMON XB - INITIALIZATIONS FROM XEIND, XEINI, XEINS, XEINX
1 CD5  667
1 CD5  668
1 CD3  669
1 CD5  NOTES
1 CD5  670
1 CD5  USES BMPAR, XDCLD, XDCLF, XEDLS, XINTE, XRCP, XRSIG, XSEGE,
1 CD5  671
1 CD5  XTCON, XXTCN
1 CD5  672
1 CD5  THE CALLS TO XEDLS PROVIDE LINKAGE TO THE INITIALIZATION SEGMENT
1 CD5  673
1 CD5  XECAL AND DIRECTIVE SEGMENTS XDCLD AND XDCLF.
1 CD5  674
1 CD5  675
1 CD5  676
1 CD5  677
1 CD5  THE LOOP STRUCTURE ASSOCIATED WITH EXECUTION CONTROL OCCURS
1 CD5  678
1 CD5  BECAUSE OF PARTITION SIZE LIMITATIONS WHICH PROHIBIT XXTCN FROM
1 CD5  679
1 CD5  CALLING XSEGE AND XINTE DIRECTLY. LOGIC FLOW BETWEEN THESE
1 CD5  680
1 CD5  MODULES IS GOVERNED BY THE VALUE OF SUBSTA. CYCLING TERMINATES
1 CD5  681
1 CD5  WHEN NASSTA IS SET TO THE DIRECTIVE LEVEL.
1 CD5  682
1 CD5  683
1 CD5  ************
BEGIN EXEC
RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASMO, QUAL & FLAGS
CALL XEMR TO INITIALIZE GLOBAL COMMON
DO FOREVER -- TERMINATES INSIDE HANDLER FOR XOFF
CALL ATOM FOR INPUT OF DIRECTIVE
IF ERROR OR NOT A VALID DIRECTIVE NAME
THEN
ISSUE MESSAGE E06
ELSE
IF NAME IS INTE
THEN
SET STATES TO INTE LEVEL
CALL XEINI TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XINTE TO EDIT TABLE
ELSE
IF NAME IS SERE
THEN
SET STATES TO SERE LEVEL
CALL XEINS TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XSERE TO EDIT TABLE
ELSE
IF NAME IS FOR SOME EXECUTION CONTROL OPTION
THEN
SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
DO UNTIL MASTER IS AT DIRECTIVE LEVEL
CALL XEXIN TO INITIALIZE DYNAMIC COMMON
EXIT TO :RESET; IF ERROR
CALL XXCMNT TO PERFORM EXECUTIONS
IF SUBSTA IS SET TO SERE LEVEL
THEN
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
ENDIF
IF EXECUTION MODE WAS SEMI OR AUTO
THEN
CALL XESCN TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES
ELSE
CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?
ENDIF
ENDIF
ENDIF
ELSE
IF SUBSTA IS NOT DIRECTIVE LEVEL
THEN
CALL XEIND TO REINITIALIZE DYNAMIC COMMON
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDER
737 4 :RESET:
738 1 IF SUBSTA IS NOT DIRECTIVE LEVEL
739 1 THEN
740 5 CALL XEIND TO REINITIALIZE DYNAMIC COMMON
741 4 ENDIF
742 3 ENDIF
743 5 ENDIF
744 4 ENDIF
745 3 ENDIF
746 4 ENDIF
747 4 ENDIF
748 3 ENDIF
749 4 ENDIF
750 3 ENDIF
751 4 ENDIF
752 3 ENDIF
753 4 ENDIF
END
REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
FORTRAN CALLING PROCEDURE

CALL XINTE

OVERLAY INTERFACE ROUTINE FOR INTERFACE TABLE EDITOR

INPUT

COMMON XE -  
LU = USER'S LOGICAL UNIT NO.

COMMON XB -  
DEBUG = DEBUG AND TRACE FLAG FOR INTERFACE TABLE EDITOR Routines

NOTES

USES ROUTINES XINIX XUDMG XERTW

* XINTE IS THE INTERFACE ROUTINE FOR THE INTERFACE TABLE EDITOR
* WHEN CALLED IN THE FDS EXECUTIVE'S OVERLAY STRUCTURE.

BEGIN XINTE

CALL XINIX TO EXECUTE INTERFACE TABLE EDITOR

CALL XERTW TO RETURN TO XEXEC IN MAIN SEGMENT

END XINTE
FORTAN CALLING PROCEDURE

CALL XINIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

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

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

OUTPUT

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

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

USES ROUTINES

EXEC

XLI

XLIT

XIPRM

XINPT
1 CD$ XCMOV
2 CD$ XBRG
3 CD$ XCTOM

104 1 "**********
105 1 * THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'INUT' DIRECTIVE
106 1 * OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
107 1 * PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
108 1 * TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
109 1 * OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
110 1 * NOTE : ALL INITIALIZATION, INCLUDING WKBFA (OLD INTERFACE TABLE),
111 1 * WAS BEEN PERFORMED BY XEINIT.
112 1 *
113 1 BEGIN XEINIT
114 2 IF UP (NO. OF PARAMETERS) > 0, THEN
115 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
116 3 * PRMTMD = 1 => CREATE N MODE
117 3 * PRMTMD = 3 => CREATE A MODE
118 3 * PRMTMD = 4 => CREATE CONTINUOUS MODE
119 3 * PRMTMD = 5 => MODIFY MODE
120 3 DO UNTIL 'EXIT' OR 'X' IS ENTERED
121 4 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTMD, SIZE, TYPE, AND STATUS
122 4 OF NEXT ARGUMENT
123 4 CALL XTCOM TO PROMPT USER AND RETURN PARSED INPUT
124 4 IF 'X' WAS NOT ENTERED, THEN
125 5 IF \" \" WAS ENTERED, THEN
126 6 SET PRMTMD TO 5
127 7 ELSE
128 8 IF NOTHING WAS ENTERED (I.E. TOKEN IS CD$), THEN
129 9 INCREMENT TO NEXT ARGUMENT
130 9 ELSE
131 9 CALL XINPT TO PROCESS THE USER'S INPUT
132 9 ENIF
133 9 ENIF
134 9 ENDO
135 9 IF A 'X' WAS ENTERED, THEN
136 9 SET RETURN CODE INDICATING X (I.E. MASSTA = 0)
137 9 ELSE
138 9 COMPRESS THE LITERAL LIST AREA
139 9 ENIF
140 9 ENIF
141 9 STORE INTERFACE TABLE AS NNEWNAME
142 9 IF STORE INTO AM NAME FAILED, THEN
143 9 SET MASSTA TO INDICATE DIRECTIVE LEVEL (=0)
144 9 ELSE
145 9 SET GOOD RETURN CODE
146 9 ENIF
147 9 ENIF
148 9 ENDO
149 1 END XEINIT
152 1 CD0   FORTRAN CALLING PROCEDEURE
153 1 CD0
154 1 CD0   CALL XIPRM
155 1 CD0
156 1 CD0   C********
157 1 CD1   PROMPT CONSTRUCTOR FOR INTERFACE TABLE EDITOR
158 1 CD1
159 1 CD1
160 1 CD1   C*******
161 1 CD2   INPUT
162 1 CD2
163 1 CD2
164 1 CD2   COMMON XB -
165 1 CD2   ARGNO, CFLAG, DFLAG, IARG, IFLAG,
166 1 CD2   ISIZE, ITYPE, LMEHFF, LITSIX, MARG,
167 1 CD2   MOXBTH, NUMARG, PRMTMD, WKBUF
168 1 CD2
169 1 CD2   C*******
170 1 CD3   OUTPUT
171 1 CD3
172 1 CD3
173 1 CD3   COMMON XB -
174 1 CD3   ARGNO, ISUB, NODSAV, PRMLEN,
175 1 CD3   PRMTMD, PROMPT
176 1 CD3
177 1 CD3   C*******
178 1 CD5   NOTES
179 1 CD5
180 1 CD5
181 1 CD5   USES ROUTINES
182 1 CD5   CMMND
183 1 CD5   XIEXT
184 1 CD5   XILSN
185 1 CD5   XRMOV
186 1 CD5   XRMOV
187 1 CD5   XRPK
188 1 CD5
189 1 CD5
190 1 CD5   C*******
1 = CONSTRUCT PROMPT TO BE ISSUED
2   BEGIN XIPMN
3   DO UNTIL A PROMPT IS CONSTRUCTED
4   IF PRMTMD = 5, THEN
5   CONSTRUCT A '!' PROMPT
6   ELSE
7     IF PRMTMD = 4 (CONTINUE MODE), OR
8     IF PRMTMD = 6 (CONTINUE HERE MODE), THEN
9     IF ARGNO IS A SCALAR, THEN
10    CONSTRUCT PROMPT AS '\\ARG=:' OR ' ARG=':
11     ELSE
12      IF PRMTMD NOT = 6, THEN
13       COMPUTE ISUB AS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LASTE)
14       ELSE
15        SET PRMTMD TO 4
16        COMPUTE CURRENT ELEMENT NO. (SUBSCRIPT) FROM CURRENT INDEX
17     ENDIF
18     IF THERE ARE NO EMPTY SLOTS BEYOND LASTE, THEN
19       SET PRMTMD TO NOOSAV (EXIT THE CONTINUE MODE)
20     ELSE
21       IF ARGNO IS DOUBLY SUBSCRIPTED PARAMETER, THEN
22         COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
23         CONSTRUCT PROMPT AS 'ARG='(I,J):
24       ELSE
25         CONSTRUCT PROMPT AS 'ARG=(ISUB):'
26     ENDIF
27     ENDIF
28   ENDIF
29   ELSE
30     IF ARGNO = BP, THEN
31       SET PRMTMD TO 5
32     ELSE
33       INCREMENT ARGNO TO NEXT PARAMETER
34       SET IFLAG TO 1, 2, OR 3 INDICATING 1.0, OR 10
35       SET LAST ENTERED INDICATOR (LASTE) TO 0
36     IF PRMTMD = 3, THEN
37       IF SOME DATA VALUE(S) OR PARM NAME EXISTS FOR ARGNO, THEN
38         CALL XILSD TO LIST DATA FOR THIS ARGUMENT
39     ENDIF
40     CONSTRUCT PROMPT AS 'ARG=:' OR 0 OR =
41     ELSE
42       IF ARGNO IS MARKED INCOMPLETE, THEN
43         IF A PARTIAL LITERAL LIST EXISTS, OR
44           THIS ARGUMENT IS A SCALAR, THEN
45           COMPUTE ISUB AS FIRST EMPTY ELEMENT
46           IF DOUBLY SUBSCRIPTED PARAMETER, THEN
47             COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
48             CONSTRUCT PROMPT 'ARG=(I,J):
49           ELSE
50             CONSTRUCT PROMPT 'ARG=(ISUB):'
51         ENDIF
52       ELSE
53         CONSTRUCT PROMPT 'ARG=:' OR 0 OR =
54       ENDIF
55     ENDIF
56     ENDIF
57   ENDIF
58   ENDIF
59
FOR CALLING PROCEDURE

CALL XIPMT

***/ PROTOTYPE PROCESS

***/ INPUT

COMMON XE - COMBUF, COMPTA, TOKENS

***/ OUTPUT

COMMON XE - ARGNO, PRMTMD

* XIPMT PROCEDURE IS THE PROMPT DIRECTIVE

BEGIN XIPMT

IF TOKEN IS NOT COMMA :ERROR2:

POSITION TO NEXT TOKEN

IF NAME IS NOT NAME :ERROR2:

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

IF NAME IS 'N', THEN

SET PRMTMD TO 1

ELSE

IF NAME IS 'A', THEN

SET PRMTMD TO 3

ELSE

ENDIF

ENDIF

SET ARGNO TO 0

EXIT TO :RETURN:

:ERROR2: CALL XMSGE FOR 'INVALID SYNTAX'

:RETURN:

END XIPMT

RETURN
337 * XILST PROCESSES THE LIST DIRECTIVE
338 1 BEGIN XILST
339 2 IF TOKEN IS '\', THEN
340 3 POSITION TO NEXT TOKEN
341 4 ERREXIT IF TOKEN IS NOT NAME ; ERRO2:
342 5 ERREXIT IF NAME IS NOT 'C', 'V', OR 'A' ; ERRO2:
343 6 SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)
344 7 POSITION TO NEXT TOKEN
345 8 ELSE
346 9 SET MODEFG TO 2
347 10 ENDF
348 11 IF TOKEN IS EOS, THEN
349 12 WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
350 13 AND STATUS
351 14 DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
352 15 IF MODEFG = 1 OR MODEFG = 3, THEN
353 16 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
354 17 ENDF
355 18 IF MODEFG = 2 OR MODEFG = 3, THEN
356 19 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
357 20 ENDF
358 21 ENDDO
359 22 ELSE
360 23 DO UNTIL EOS IS REACHED
361 24 ERREXIT IF TOKEN IS NOT COMMA ; ERROR2:
362 25 ERREXIT IF NEXT TOKEN IS NOT NAME ; ERROR2:
363 26 SET ARGNO TO 1
364 27 START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
365 28 EXIT IF NAME = ARGNO'S NAME IN PROPPT TABLE
366 29 IF MODEFG = 1 OR MODEFG = 3, THEN
367 30 CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
368 31 ENDF
369 32 IF MODEFG = 2 OR MODEFG = 3, THEN
370 33 CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
371 34 ENDF
372 35 ENDDO
373 36 ENLOOP
374 37 PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
375 38 ENSEARCH
376 39 INCREMENT TO NEXT TOKEN
377 40 ENDO
378 41 EXIT TO :RETURN:
379 42 :ERROR2: CALL XRMNG TO WRITE 'INVALID SYNTAX'
380 43 :RETURN:
381 44 END XILST
FORTRAN CALLING PROCEDURE

CALL XISUB

**EFFECTIVE SUBSCRIPT CALCULATION ROUTINE**

**INPUT**

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XE - IDIM, ISIZE, LENEFF

**OUTPUT**

COMMON XE - IRETC, ISUB

USES ROUTINES

XRMSG

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

**THE INPUT SUBSCRIPT**

**BEGIN ISUB**

INCREMENTS TO NEXT TOKEN

E Rexit IF TOKEN IS NOT INTEGER VALUE :ERR14:

IF IDIM FOR THIS ARGUMENT > 0, THEN

E Rexit IF SPECIFIED INTEGER VALUE > IDIM :ERR16:

INCREMENTS TO NEXT TOKEN

E Rexit IF TOKEN IS NOT COMMA :ERR15:

INCREMENTS TO NEXT TOKEN

E Rexit IF TOKEN IS NOT INTEGER VALUE :ERR14:

CALCULATE ISUB AS (j-1)*IDIM+1

ELSE

SET ISUB TO INTEGER VALUE

END IF

E Rexit IF ISUB = 0 :ERR16:

INCREMENTS TO NEXT TOKEN

E Rexit IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:

EXIT TO RETURN:

:ERR14: CALL XRMSG - 'INVALID SUBSCRIPT SYNTAX'

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

:ERR16: CALL XRMSG - 'INVALID SUBSCRIPT VALUE'

RETURN:

END XISUB
FORTRAN CALLING PROCEDURE

CALL XDAT

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

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

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - CFLAG, COMPLE, DFLAG, IARG, IARG4,
ISIZE, ISIZE, IUP, LITSP, LITDUP, LITDSP,
LITON, LITSIZE, MODSRW, MXBUF, MBITM,
PRMTRD, SFLAG, WKBLS, WKBUF

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

COMMON XB - IRETC, ISUB, LITDSP, LITON, MBITM,
PRMTRD, WKBUF

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

COMMON X S -

IDISP = INDEX INTO WKBUF OF LOCATION FOR THIS
LITERAL DATUM
NUMCL = NUMBER OF CONSECUTIVE ELEMENTS TO BE
MARKED COMPLETED AS A RESULT OF THIS
LITERAL DATUM
STACK = PUSH-DOWN LIST (MAX. OF 4 ENTRIES)
DESCRIBING NESTED REPEAT GROUPS.
 Each ENTRY IS 3 WORDS:
 WORD 1 = INDEX TO 1ST TOKEN (IN COMBUF)
 WORD 2 = REPEAT COUNT
 WORD 3 = FLAG INDICATING WHETHER
 REPEAT GROUP IS PARENTHESETICALLY
 GROUPED

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

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

USES ROUTINES

XISUB
XRBIT
XRMOV
XRMSG

***********
XNMDB
XSET

* XDAT PROCESSES THE INPUT LITERAL LIST

1 BEGIN XDAT
2 DO UNTIL END IS REACHED, OR
3 UNTIL AN ERROR OCCURS
4 IF THIS IS A DATA ELEMENT, THEN
5 (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
6 IF ARG. TYPE (IYYPE) IS INTEGER, REAL, OR DOUBLE, THEN
7 EREXIT IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
8 SET LENGTH TO BE MOVED (LENOV) TO EFFECTIVE LENGTH FOR DATA
9 OF ARG.'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
10 ELSE
11 IF ARG. TYPE IS FREE, THEN
12 SET LENGTH TO BE MOVED (LENOV) TO EFFECTIVE LENGTH
13 FOR DATA INPUT
14 ELSE THIS MUST BE CHARACTER DATA BEING INPUT
15 ERREXIT IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ERR10:
16 INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
17 COMPUTE NO. WORDS IN INPUT CHARACTER STRING
18 EREXIT IF NO. WORDS (LENOV) > EFFECTIVE LENGTH OF
19 THIS ARGUMENT'S DATA (LENEFF) :ERR10:
20 ENDIF
21 ENDIF
22 INCREMENT TOKEN POINTER TO THE DATA INPUT
23 VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
24 ARGUMENT FOR DATA INPUT (ISUB <= ISIZE - LENOV + 1)
25 EREXIT IF INSUFFICIENT SPACE :ERR11:
26 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
27 ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
28 ENDIF
29 MOVE DATA FROM INPUT COMMUNICATIONS BUFFERS TO LITERAL AREA
30 SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
31 ARGUMENT, TO LENOV
32 IF LENOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
33 MOVE LENOV-LENEFF BLANKS INTO LITERAL AREA AS A FILL
34 ENDIF
35 INCREMENT TO NEXT TOKEN
36 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENOV
37 SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
38 ELSE ITOKEN INDICATES NON-DATA ELEMENT
39 IF THIS IS A "(" , THEN
40 CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
41 BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (IYYPE) AND
42 SECONDARY DIMENSION (IDIM)
43 EXIT XDAT IF ERROR (IRET < 0)
44 ELSE
45 IF THIS IS AN "\", THEN
46 CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
47 ELSE
48 MARK ONE ELEMENT OF THIS ARGUMENT AT ISUB AS INCOMPLETE
49 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
50 LENGTH OF ONE ELEMENT (LENFEF)
51 ENDIF
52 TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
53 INCREMENT TO NEXT TOKEN
ELSE
    IF Token IS A REPEAT COUNT, THEN
        IF NEXT Token IS ",", THEN
            SET PARENFLAG TO 1
            INCREMENT TO NEXT Token
        ELSE
            SET PARENFLAG TO 0
            ENDIF
        ENDIF
        SAVE REPEAT COUNT, TOKEN INDEX, AND PARENFLAG IN A PUSH-DOWN STACK
        ERROR (INVALID FIELD) :ERROR?
    ENDIF
    INCREMENT TO NEXT Token
    DO UNTIL Token IS NOT ")"
        IF PUSH-DOWN STACK IS NOT EMPTY, AND
            (PREVIOUS Token WAS DATA, AND
                PARENFLAG OF TOP OF STACK ENTRY IS 0), OR
            (CURRENT Token IS ")", AND
                PARENFLAG OF TOP OF STACK ENTRY IS 1), THEN
                GET REPEAT COUNT OF TOP OF STACK ENTRY
                DECREMENT REPEAT COUNT BY 1
        ELSE
            IF Token IS ",", THEN
                INCREMENT TO NEXT Token
            ELSE
                EXIT LOOP
            ENDIF
        ENDIF
        IF PREVIOUS Token WAS NOT A SUBSCRIPT, THEN
            IF THIS Token IS NOT AN EOS, THEN
                ERROR IF Token IS NOT A COMMA :ERROR2;
            ENDIF
            INCREMENT TO NEXT Token
        ENDIF
    ENDIF
ENDDO
ENDDO
ENDO
615 1 EXIT TO END XIDAT
616 2 :ERROR: CALL XMS65 "ONLY DATA VALID TO RIGHT OF "
617 2 :ERROR: CALL XMS65 "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
618 2 :ERROR: CALL XMS65 "INCOMPATIBLE WITH TYPE OF ARGUMENT"
619 1 :RETURN:
620 1 END XIDAT

5-106
422  1 CDO	 FORTRAN CALLING PROCEDURE
423  1 CDO
424  1 CDO
425  1 CDO	 CALL XINPT
426  1 CDO
427  1 C********
428  1 CDO
429  1 CDO	 INTERFACE TABLE EDITOR'S INPUT PROCESSOR
430  1 CDO
431  1 C********
432  1 CDO
433  1 CDO	 INPUT
434  1 CDO
435  1 CDO	 COMMON XE - COMBUF, COMPTR, TOKENS
436  1 CDO
437  1 CDO	 COMMON XB - DIRECT, IFLAG, ISUB, NARG,
438  1 CDO
439  1 CDO
440  1 C********
441  1 CDO
442  1 CDO	 OUTPUT
443  1 CDO
444  1 CDO	 COMMON XB - ARGNO, IRETC
445  1 CD5
446  1 C********
447  1 CD5
448  1 CD5	 NOTES
449  1 CD5
450  1 CD5	 USES ROUTINES
451  1 CD5
452  1 CD5	 XIDAF
453  1 CD5	 XIEE
454  1 CD5	 XILST
455  1 CD5	 XIPAM
456  1 CD5	 XINWE
457  1 CD5	 XRCPR
458  1 CD5	 XROV
459  1 CD5	 XRWSG
460  1 CD5
461  1 C********
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 'O', THEN
10 CALL XIPAR TO PROCESS A PARAMETER FIELD
11 ELSE
12 ERREXIT IF IFLAG IS NOT I ('-') :ERROR8:
13 CALL XIDAT TO PROCESS DATA LIST
14 ENDIF
15 EXIT XINPT
16 EXIT TO :RETURN:
17 :ERROR2: CALL XMSG "INVALID SYNTAX"
18 :ERROR8: CALL XMSG "MUST USE PARAMETER NAME TO RIGHT OF @ OR =" 
19 :RETURN:
20 END XINPT
**FORTRAN CALLING PROCEDURE**

**CALL XPAR**

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

**PROCESS AN INPUT PARAMETER NAME AND ANY ASSOCIATED SUBSCRIPT FIELD(S)**

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

**INPUT**

**.COMMON XE - COMBUF, COMPTK, TOKENS**

**COMMON XB - DFLAG, IARG, IAREA, ICLASS, LITDSP, SFLAG, WKBLNG, WRBUF**

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

**OUTPUT**

**COMMON XB - IRET, LITDWN, WRBUF**

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

**NOTES**

**USES ROUTINES**

**XRMC, XRMSG, XRSET**
1 * XIPAR PROCESSES A USER SPECIFIED PARAMETER FIELD

1 BEGIN XIPAR
2 IF A NAME IS SPECIFIED, THEN
3 INC o xE MENT TO NEXT TOKEN
4 IF TOKEN IS '(', THEN
5 PROCESS I AND J SUBSCRIPTS
6 EXIT IF INVALID SUBSCRIPTING :ERR14:
7 IF DOUBLY SUBSCRIP TED, THEN
8 SET S-FLAG IN ARGNO'S SPECS. FIELD
9 SET LITOSP IN ARGNO'S SPECS. TO NEXT LITERAL AREA SPACE (LITOWN)
10 PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT
11 ELSE
12 SET LITOSP IN ARGNO'S SPECS TO ISUB
13 END IF
14 ELSE
15 SET LITOSP IN ARGNO'S SPECS TO 0
16 END IF
17 ERREXIT IF ORDER HAS MORE THAN 4 CHARACTERS :ERR18:
18 ERREXIT IF EXTRAN'GULOUS FIELD INPUT :ERR02:
19 TURN OF D-FLAG (SOME LITERAL DATA) IN ARGNO'S SPECS.
20 SET PARAMETER NAME INTO ARGUMENT'S CHARACTERISTICS
21 SET COMPLETE (AND S) FLAG IN ARGUMENT'S CHARACTERISTICS
22 IF ALL ARGUMENTS ARE COMPLETE, THEN
23 SET INTERFACE TABLE COMPLETE FLAG
24 ELSE
25 EL:`. MUST BE A $ INPUT
26 ?REXIT IF NOT AN AMPERSAND ($) INPUT :ERR02:
27 CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS
28 SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
29 END IF
30 EXIT TO :RETURN:
31 :ERR02: CALL XRM SG - "INVALID SYNTAX"
32 :ERR14: CALL XRM SG - "INVALID SUBSCRIPT SYNTAX"
33 :ERR18: CALL XRM SG - "INVALID ORDER NAME"
34 :RETURN:
35 END XIPAR
1 + XILSD WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
2 * IS RETURNED AS A PROMPT.
3 BEGIN XILSD
4 SET ARGUMENT NAME INTO BUFFER
5 USE TOFLAG TO DETERMINE WHICH OF 'B', 'A', OR 'AB'
6 WILL GO INTO THE PRINT BUFFER
7 IF B-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
8 IF A PARAMETER NAME IS SPECIFIED, THEN
9 PUT PARAMETER NAME INTO BUFFER
10 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
11 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
12 PUT SUBSCRIPT INTO BUFFER
13 ELSE
14 IF LISTSP OF ARGUMENT IS > 0, THEN
15 COMPUTE AND CONVERT THIS SUBSCRIPT
16 PUT SUBSCRIPT INTO BUFFER
17 ENDIF
18 WRITE OUT THE PRINT BUFFER BUILT
19 ENDIF
20 ELSE
21 LOCATE LITERAL LIST AND MASK
22 IF SYMBOLIC STRING, THEN
23 CALL FILES TO PRINT SYMBOLIC STRING
24 ELSE
25 DO UNTIL ALL ELEMENTS PROCESSED
26 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
27 UNTIL ALL ELEMENT'S PROCESSED
28 COMPUTE AND CONVERT THE SUBSCRIPT
29 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
30 PUT "B" INTO BUFFER
31 ELSE
32 CONVERT THE DATA USING XR06, XRE14, OR XR16
33 PUT DATA AND "S" INTO BUFFER
34 ENDIF
35 ENDIF
36 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
37 REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
38 WRITE OUT THE PRINT BUFFER BUILT
39 ENDIF
40 END
41 1 END XILSD
<table>
<thead>
<tr>
<th>ENTRY</th>
<th>TOKEN</th>
<th>WORD 1</th>
<th>WORD 2</th>
<th>WORD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTEGER</td>
<td>3</td>
<td>-1&gt;CALL XE16</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>REAL</td>
<td>7</td>
<td>-2&gt;CALL XE14</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>DOUBLE</td>
<td>9</td>
<td>-3&gt;CALL XE18</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>NAME</td>
<td>0</td>
<td>0&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>NAME</td>
<td>0</td>
<td>0&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>NAME</td>
<td>0</td>
<td>0&gt;ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>CHAR. STR.</td>
<td>-1&gt;USE VALUE</td>
<td>-5&gt;USE SIZE+2WORDS</td>
<td>-1&gt;USE SIZE+2 WORDS</td>
</tr>
</tbody>
</table>
BEGIN XILSS

MOVE A * CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER

EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32

USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)

EXIT TO ERROR 2 IF FIELD = 0

IF SIZE < 0, THEN
SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMBOL STRING

ENDIF

IF TOKSIZ < 0, THEN
SET TOKSIZ TO SIZE + 2

ENDIF

IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE

CLEAR PRINT BUFFER TO BLANKS

ENDIF

IF FIELD > 0, THEN
MOVE FIELD INTO CURRENT PRINT BUFFER POSITION

ELSE
CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD

:ONE: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING

AND PUT RESULTS INTO PRINT BUFFER

:TW0: CALL XRI14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING

AND PUT RESULTS INTO PRINT BUFFER

:THREE: CALL XRD18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING

AND PUT RESULTS INTO PRINT BUFFER

:FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER

:FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING

:SEI: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS INTO PRINT BUFFER FOLLOWED BY AN "R"

:EXIT: PUT A * CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER

INDEX BY 1

CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE

EXIT XILSS

ENDCASE

INCREMENT PRINT BUFFER INDEX BY SIZE
1053  1  CD0     FORTRAN CALLING PROCEDURE
1054  1  CD0
1055  1  CD0
1056  1  CD0     CALL XIEXT
1057  1  CD0
1058  1  CD0
1059  1  CD0     ********
1060  1  CD1     EXTRACT VARIOUS FIELDS OF AN ARGUMENTS CHARACTERISTICS
1061  1  CD1
1062  1  CD1     AND PUT VALUES INTO COMMON
1063  1  CD1
1064  1  CD0     ********
1065  1  CD2     INPUT
1066  1  CD2
1067  1  CD2
1068  1  CD2     COMMON XB - ARGNO, ISIZES, WBUF
1069  1  CD2
1070  1  CD2
1071  1  CD0     ********
1072  1  CD0
1073  1  CD3     OUTPUT
1074  1  CD3
1075  1  CD3
1076  1  CD3     COMMON XB - CFLAG, DFLAG, IARG, IARG4,
1077  1  CD3     ICCLASS, IDIM, IFLAG,-ISIZE,
1078  1  CD3
1079  1  CD3
1080  1  CD5     NOTES
1081  1  CD5
1082  1  CD5
1083  1  CD5     USES ROUTINES
1084  1  CD5
1085  1  CD5
1086  1  CD5     IAND
1087  1  CD5
1088  1  CD5
1089  1  CD0     ********
1090  1  CD0     * EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS
1091  1  CD0
1092  1  CD0
1093  1  CD0
1094  1  CD0
1095  1  CD0
1096  1  CD0     BEGIN XIEXT
1097  1  CD0
1098  1  CD0
1099  1  CD0     USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
1100  1  CD0
1101  1  CD0
1102  1  CD0
1103  1  CD0     IN THE WORKING BUFFER
1104  1  CD0
1105  1  CD0
1106  1  CD0     EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE
1107  1  CD0
1108  1  CD0
1109  1  CD0
1110  1  CD0
1111  1  CD0
1112  1  CD0     END XIEXT
FORTRAN CALLING PROCEDURE
CALL XILIT

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

INPUTS
COMMON XB = ARGNO, DFLAG, IAIGA, ISIZE,
LENSTR, LITOSP, LITOWN, LITPTR,
LITSZ, MARG, MXDTM, MOBTH
NUMARG, SFLAG, WBLNS, WBUF

OUTPUTS
COMMON XB = LITOWN, LITLEN, LITPTR, WBUF

INTERNAL VARIABLES
COMMON XS = 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
XBRST
XBRMOV
XRNXT
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 5 ELSE
1153 6 DO UNTIL ALL BITS OF BIT MASK PROCESSED
1154 7 IF THE BIT IS ON, THEN
1155 8 MOVE CORRECT NUMBER OF WORDS (LENFF) OF LITERAL
1156 8 UP IN THE WORKING BUFFER
1157 7 ENDEF
1158 6 ENDDO
1159 5 ENDEEF
1160 5 CALCULATE NEW DISPLACEMENT AND SET IN LIT0SP
1161 4 ENDF
1162 3 ENDDO
1163 2 ENDDO
1164 1 END XILIT
CD************
CD
CD
CD RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:
CD
CD :RU,XLMAN,LU,O,ISECU,FLAGS
CD
CD************
CD
CD OFFLINE MAINTENANCE PROGRAM, XLMAN CREATES, DELETES AND
CD MODIFIES FDS FILES.
CD
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 TOKENZ - (INTEGER, 29 WORDS) THE 29 TOKENS 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 XLD5, XLDIF, XLDL, XLINT, XLMOD, XLMGR
CD XLRM, XLPK, XLCMN, XRMV, XRMGR
CD
CD XE COMMON USED:
CD
CD EQUVALENCE (XE(1), LU )
CD
CD + (XE(2), ICLASS), (XE(3), ISECU )
CD + (XE(4), FLAGS ), (XE(5), MASS),
CD + (XE(6), SUBSTA), (XE(7), NUMBER ),
CD + (XE(8), VALFLO), (XE(9), TOKENS ),
CD + (XE(142), ICR ), (XE(145), COMBUF)

CD************
1 BEGIN XLMAN
2 CALL RMHRP TO GET INPUT PARAMETERS
3 INITIALIZE COMMON TO ZERO
4 SET CLASS NUMBER TO ZERO
5 CALL EXEC TO GET A CLASS NUMBER
6 EXIT XLMAN IF SECURITY CODE IS NOT VALID
7 CALL XNMO TO INITIALIZE TOKENS IN COMMON
8 DO FOREVER
9 3 : P R O M P T :
10 3 : INITIALIZE MASTER AND SUBSTATE FLAGS
11 3 CALL XTCM TO PROMPT FOR OPTION
12 3 IF XTCOM RETURN CODE IS NOT ZERO OR
13 4 FIRST TOKEN IS NOT AN INTEGER OR
14 4 INTEGER > 7 THEN
15 4 CALL XNMSG TO WRITE INVALID RESPONSE
16 4 GO TO : PROMPT:
17 3 ENDF:
18 3 CASE INTEGER (:XLPRM:, :XLPRM:, :XLPRM:, :XLPRO:, :XLDELI:, :XLMOD:,
19 4 :XLMSG:, :XLDBF:, :XLDBF:, :EXIT:)
20 4 :XLPRM:
21 4 SET NUMOR TO INTEGER
22 4 CALL XELS TO LOAD XLPRM TO CREATE SYSTEM PROMPT FILE
23 4 :XLPRO:
24 4 SET VALFLG TO SAY ORIGINAL XLPRO REQUEST
25 4 CALL XELS TO LOAD XLPRO TO ADD A PROCESSOR
26 5 DO UNTIL VALFLG SAYS EXIT (X)
27 5 CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
28 5 CALL XELS TO LOAD XLPRO TO COMPLETE PROCESSING
29 4 ENDDO
30 4 :XLDELI:
31 4 CALL XELS TO LOAD XLDEL TO DELETE A PROCESSOR
32 4 :XLMOD:
33 4 SET VALFLG TO SAY OPTIMAL XLMOD REQUEST
34 4 CALL XELS TO LOAD XLMOD TO MODIFY A PROCESSOR
35 5 DO UNTIL VALFLG SAYS EXIT (X)
36 5 CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
37 5 CALL XELS TO LOAD XLMOD TO COMPLETE PROCESSING
38 4 ENDDO
39 4 :XLMSG:
40 4 CALL XELS TO LOAD XLMSG TO ADD A MESSAGE
41 4 :XLDBF:
42 4 CALL XELS TO LOAD XLDBF TO HANDLE DATA BASE FILES
43 3 ENDCASE
44 2 ENDDO
45 2 :EXIT:
46 2 CALL EXEC TO RELEASE CLASS NUMBER
47 1 END XLMAN
FORTRAN CALLING PROTOCOL:

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 XRMSG

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:

KCYT,CLOSE,CREATE,PURGE,WAITF

FRS FUNCTIONS AND ROUTINES USED:

XMOV,XRMSG,XTCOM

COMMON USED:

ERUVALENCE

+FX(3), XSECU), (FX(7), XNUMBR),

+FX(145), FCR), (FX(145), COMEGF),

+FX(145), NOTOK), (FX(145), COMEGF),

+FX(146), COMA ; (FX(1), IBUF),

+FXB(128), MODIR)
160 1 BEGIN XLPRM
161   2 :PRIM:
162   4 CALL XICOM TO PROMPT USER FOR O DIRECTIVES
163   4 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
164   4 CALL CREAT TO CREATE PROMPT FILE
165   4 IF FILE ALREADY EXISTS THEN
166   4 CALL XICOM TO PROMPT USER FOR SCRATCH/CANCEL
167   4 EXIT XLPRM IF RESPONSE IS CANCEL
168   4 CALL PURGE TO PURGE PROMPT FILE
169   4 ERREXIT IF PURGE ERROR TO :FILERR:
170   4 GO TO :PRIM1:
171   7 ELSE (CREATE NEW FILE)
172   7 ERREXIT IF CREAT ERROR TO :FILERR:
173   7 CALL XICOM TO PROMPT USER FOR LIST OF DIRECTIVES
174   7 ERREXIT IF LIST IS INCONSISTENT WITH O OF DIRECTIVES TO :ERR1:
175   7 DO FOR EACH DIRECTIVE
176   8 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
177   8 CALL XMNOV TO MOVE DIRECTIVE INTO BUFFER
178   8 ENDDO
179   8 CALL WRITF TO WRITE LIST OF DIRECTIVES TO FILE
180   8 ERREXIT IF WRITF ERROR TO :FILERR:
181   8 DO FOR EACH DIRECTIVE
182   9 CALL XICOM TO PROMPT USER FOR DEFINITION
183   9 DO UNTIL EOS IS REACHED IN RESPONSE
184   9 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185   9 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186   9 MOVE RESPONSE INTO BUFFER
187   9 SET CONTROL CHARACTERS IN BUFFER
188   9 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
189  10 ENDDO
190  10 SET REMAINDER OF BUFFER TO NULL
191  10 CALL WRITF TO WRITE DEFINITION
192  10 ERREXIT IF WRITF ERROR TO :FILERR:
193  10 ENDDO
194  10 CALL CLOSE TO CLOSE FILE
195  10 ERREXIT IF CLOSE ERROR TO :FILERR:
196  10 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
197  2 ENDF
198  1 EXIT XLPRM
199  2 :ERR1: (ERROR IN RESPONSE)
200  4 CALL XMSG TO DISPLAY ERROR IN RESPONSE
201  4 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
202  2 :FILERR: (FILE ACCESS ERROR)
203  4 CALL XMSG TO DISPLAY ERROR
204  4 CALL CLOSE TO CLOSE FILE
205  4 CALL PURGE TO PURGE FILE
206  1 END XLPRM
1 BEGIN XLPRO
2 IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN
3 INITIALIZE MASTER AND SUB STATES
4 IF PROCESSOR HAS AN INTERFACE TABLE THEN
5 SET RETN = 2
6 CALL ITCOM TO PROMPT FOR # PARAMETERS
7 ERREXIT IF ITCOM RETURN CODE IS NOT ZERO OR
8 ERREXIT IF # PARAMETERS NOT INTEGER 1-63 TO :PRMERR:
9 CREATE HEAD ENTRY WITH #PARAMETERS AND PROCESSOR NAME
10 CALL XLFLS TO WRITE INSTRUCTIONS FOR ENTERING SPECS
11 DO FOR # PARAMETERS
12 CALL XLPS TO CREATE ONE PARAMETER ENTRY
13 ENDIF
14 SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS
15 CALL XLPLF TO CREATE PROMPT FILE
16 SET RETN = 3
17 CALL ITCOM FOR DEFAULT VALUES DECISION
18 ERREXIT IF RETURN CODE IS NOT ZERO OR
19 ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:
20 IF RESPONSE WAS YE THEN
21 SET FLAG TO CALL INTERFACE TABLE EDITOR
22 EXIT XLPRO
23 ELSE
24 SET CODES ARRAY TO ADD ONLY ABSTRACT
25 CALL XLPLF TO CREATE PROMPT FILE
26 PERFORM XLPRO - NO RETURN EXPECTED
27 ENDIF
28 ENDIF
29 CALL XLPLF TO CREATE THE DEFAULT INTERFACE TABLE FILE
30 PERFORM XLPRO - NO RETURN EXPECTED
31 :PRMERR:
32 CALL XRMG TC DISPLAY ERROR MESSAGE
33 GO TO (:PRMPT2;:PRMPT2;:PRMPT3); RETN
34 END XLPRO
315 1 BEGIN LIBD
316 2 CALL OPEN TO OPEN LIBRARY DIRECTORY
317 3 IF RETURN CODE SAYS FILE NOT FOUND THEN
318 4 SET RECORD 1 TO ALL ZEROS
319 5 ELSE
320 6 ERREXIT IF FILE ERROR TO :FILERR:
321 7 CALL READF AND CLOSE TO READ IN LIBRARY DIRECTORY
322 8 ERREXIT IF FILE ERROR TO :FILERR:
323 9 IF # PROCES SORS + 1 > 5D THEN
324 10 CALL XRMNG TO WRITE ERROR: TOO MANY PROCESSORS
325 11 EXIT XLPRO
326 12 ENDF
327 13 SET RETM = 1
328 14 :PRNPT1:
329 15 CALL XTCOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
330 16 EXIT XLPRO IF RETURN CODE Says X ENTERED
331 17 ERREXIT IF RETURN CODE > 0 OR
332 18 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
333 19 CALL XRMV TO MOVE PROCESSOR NAME INTO ENTRY
334 20 ERREXIT IF VERSION IS NOT INTEGER VALUE 0-127 TO :PRMERR:
335 21 CALL XRSET TO SET VERSION IN ENTRY
336 22 ERREXIT IF INTERFACE TABLE OPTION IS NOT YE OR NO TO :PRMERR:
337 23 SET IT BIT = 0
338 24 IF RESPONSE IS YES THEN
339 25 SET IT BIT = 1
340 26 ENDF
341 27 CALL XRSET TO SET BIT ON/OFF
342 28 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
343 29 INCREMENT # PROCES SORS BY 1
344 30 CALL XRMV TO MOVE NEW ENTRY INTO XLIBD
345 31 IF # PROCESSORS > 1 THEN
346 32 CALL PURGE TO PURGE OLD FILE
347 33 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
348 34 ENDF
349 35 CALL CREATE, WRITE AND CLOSE TO CREATE NEW LIBRARY DIRECTORY
350 36 ERREXIT IF FILE ERROR TO :FILERR:
351 37 ENDF
352 38 1 END LIBD
410   1 BEGIN XLCDB
411   2 CALL OPEN TO OPEN OLDfil
412   2 ERREXIT IF OPEN ERROR TO :ERR3;
413   3 CALL READ TO READ HEADER RECORD
414   3 ERREXIT IF READ ERROR TO :ERR2;
415   4 COMPUTE NUMBER OF TOC RECORDS (NOTOC) AS(# ENTRIES + 16)/16
416   5 IF NOTOC > 1 THEN
417   5 CALL READ TO READ IN REMAINING TOC RECORDS
418   5 ERREXIT IF READ ERROR TO :ERR2;
419   6 ENDF
420   7 SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NOTOC
421   7 CALL CREAT TO CREATE NEWFIL
422   7 ERREXIT IF CREAT ERROR TO :ERR2;
423   8 DO FOR EACH TOC ENTRY
424   9 IF REQUEST WAS RDB TO NEWF THEN
425  10   ERREXIT IF NAME > 4 CHARS OR
426  10   ERREXIT IF CLASS IS DNO AND NAME > 2 CHARS TO :ERR1:
427  11   APPEND DNB CONVENTION TO FRONT OF NAME
428  11   ELSE
429  12   REMOVE DNB CONVENTION FROM NAME
430  13   ENDF
431  14 ENDDO
432  15 CALL WRITE TO WRITE NEW TOC RECORDS TO NEWFIL
433  15 ERREXIT IF WRIT ERROR TO :ERR1:
434  16 SET TOCPTR TO FIRST TOC ENTRY
435  16 SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
436  17 CONVERT SIZE FROM BLOCKS TO WORDS
437  18 DO UNTIL ALL RECORDS ARE CUPED (SIZE = 0)
438  19 SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRIT (1024)
440  21 IF SIZE < LENGTH THEN
441  22 SET LENGTH TO SIZE
443  24 ENDF
444  25 CALL READ TO READ LENGTH WORDS OF DATA
445  25 ERREXIT IF READ ERROR TO :ERR1:
446  26 DECREMENT SIZE BY LENGTH READ
447  26 UPDATE FIRST AND LAST RECORD NUMBERS
448  27 START SEARCH FOR TOCPTR = TOCPTR TO LAST TOC ENTRY
449  27 EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
450  28 IF CLASS IS AN INTERFACE TABLE THEN
451  29 COMPUTE INDEX INTO DATBUF FROM FIRST RECORD #, Record # IN TOC ENTRY
452  30 AM INDEX IN TOC ENTRY
453  31 SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
455  33 ENDS
456  35 ENDDO
458 2 CALL CLOSE TO CLOSE NEWFIL
459 2 CALL CLOSE TO CLOSE OLDFIL
460 2 CALL XDOBD TO DELETE OLDFIL FROM PDB LOG FILE
461 2 CALL PURGE TO PURGE OLDFIL FROM SYSTEM
462 1 EXIT XLCD0
463 2 :ERR1:
464 2 CALL CLOSE TO CLOSE NEWFIL
465 2 CALL PURGE TO PURGE NEWFIL
466 2 :ERR2:
467 2 CALL CLOSE TO CLOSE OLDFIL
468 2 :ERR3:
469 2 IF REQUEST WAS PDB TO MDG THEN
470 3 SET QUAL TO SAT DELETE MDG FILE
471 3 ELSE (REQUEST WAS MDG TO PDB)
472 3 SET QUAL TO SAT DELETE PDB FILE
473 2 ENDF1
474 2 CALL XDOBD TO DELETE MDG/PDB FROM LOG FILE
475 2 IF ERROR WAS FILE MANAGER THEN
476 3 CALL XRM5G TO DISPLAY ERROR AND RETURN CODE
477 3 ELSE
478 3 CALL XRM5G TO DISPLAY ERROR
479 2 ENDF1
480 1 END XLCD0
482 1 CD************
483 1 CD
484 1 CD0
485 1 CD0  FORTRAN CALLING SEQUENCE:
486 1 CD0  CALL XLDOS ('XLDOS')
487 1 CD0
488 1 CD************
489 1 CD
490 1 CD0  XLDOS IS THE DATA BASE FILES MAIN ROUTINE. IT DETERMINES WHICH
491 1 CD0  DATA BASE ROUTINE IS NEEDED AND GATHERS INPUTS NEEDED BY THAT
492 1 CD0  ROUTINE. HANDLES ALL PDB LOG FILE RELATED TRANSACTIONS.
493 1 CD
494 1 CD************
495 1 CD2
496 1 CD2  INPUTS IN COMMON:
497 1 CD2  XE(3) ISECU, XE(7) NUMBR,
498 1 CD2  XE(85)TOKENS, XE(142) ICR
499 1 CD2
500 1 CD2
501 1 CD2
502 1 CD2
503 1 CD4
504 1 CD4  INTERNAL VARIABLES IN COMMON
505 1 CD4  XE(3) QUAL - USER QUALIFIER REQUIRED BY XDB ROUTINES AND XROFW
506 1 CD4  XE(6) SECU - TEMPORARY SLOTS FOR ISECU DURING THIS OVERLAY
507 1 CD4  XE(5) OLDFILE - OLD FILE NAME
508 1 CD4  XE(6) NEWFILE - NEW FILE NAME
509 1 CD4  XE(9) TOTSIZ - SIZE OF OLDFILE/NEWFILE IN BLOCKS
510 1 CD4  XE(10) FILCHR - A BASE CHARACTERS OF FILE NAME
511 1 CD4  XE(12) QUALIF - QUALIFIER ENTERED BY USER
512 1 CD4
513 1 CD4
514 1 CD5
515 1 CD5  RTE ROUTINES USED:
516 1 CD5  KCVT, OPENW
517 1 CD5
518 1 CD5
519 1 CD5
520 1 CD5  FDS ROUTINES USED:
521 1 CD5  XDDBA, XDDBY, XERTW, XLDDB, XLPCH
522 1 CD5  XLPCC, XERT, XMADE, XNFR, XSER, XCON
523 1 CD5
524 1 CD5
525 1 CD5
526 1 CD5  COMMON USES:
527 1 CD5  EQUIVALENCE (XE(3), ISECU ),
528 1 CD5  +XE(3) QUAL , (XE(7), NUMBR ),
529 1 CD5  +XE(8) SECU , (XE(85) TOKENS ),
530 1 CD5  +XE(142) ICR , (XE(145) COMDFU ),
531 1 CD5  +XE(5) OLDFILE , (XE(6) NEWFILE ),
532 1 CD5  +XE(9) TOTSIZ , (XE(10) FILCHR ),
533 1 CD5  +XE(12) QUALIF , (XE(99) IERR ),
534 1 CD5  +XE(100) IORCB )
535 1 CD************
1 BEGIN XDBF:
2    IF REQUEST IS TO CREATE/MODIFY LOG FILE THEN
3       CALL OPEN TO OPEN MDB/PDB LOG FILE
4       IF OPEN ERROR SAYS FILE NOT FOUND THEN
5          CALL XLPCR TO CREATE MDB/PDB LOG FILE
6          ELSE
7             ERREXIT IF OPEN ERROR TO :FILEERR:
8          ENDIF
9       ENDIF
10      ELSE
11         DO UNTIL USER REQUESTS EXIT (?)
12         CALL XCOM TO PROMPT USER FOR NAME AND USER ID
13         IF RESPONSE IS NOT EXIT (PERCENT) THEN
14             ERREXIT IF RESPONSE IS INVALID TO :ERR1:
15             SAVE A CHARACTER NAME AND ID IN COMMON
16             SET QUALIFIER TO SEARCH FOR PDB NAME
17             ELSE (REQUEST WAS FOR MDB TO PDB)
18                SET QUALIFIER TO SEARCH FOR MDB NAME
19                ENDIF
20         ENDIF
21         CALL XDBOV TO VERIFY EXISTENCE OF MDB/PDB DEPENDING ON QUALIFIER
22         ERREXIT IF NAME WAS NOT FOUND TO :ERR1:
23         ERREXIT IF FILE MANAGER ERROR TO :FILEERR:
24         IF REQUEST WAS PDB TO MDB THEN
25            SET QUALIFIER TO ADD MDB TO LOG FILE
26            ELSE (REQUEST WAS MDB TO PDB)
27                SET QUALIFIER TO ADD PDB TO LOG FILE
28                ENDIF
29         ENDIF
30         CALL XDBA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
31         ERREXIT IF DUPLICATE NAME OF
32         ERREXIT IF MAXIMUM NUMBER OF ENTRIES EXIST TO :ERR1:
33         ERREXIT IF FILE MANAGER ERROR TO :FILEERR:
34         IF REQUEST WAS PDB TO MDB THEN
35            CALL XRDFM TO SET OLDFILE TO PDB NAME
36            SET NEWFILE TO PDB NAME
37            ELSE (REQUEST WAS MDB TO PDB)
38                SET OLDFILE TO MDB NAME
39                CALL XRDFM TO SET NEWFILE TO MDB NAME
40                ENDIF
41                ENDIF
42         CALL XLBDB TO COPY OLDFILE TO NEWFILE
43     ENDIF
44     ENDIF
45     ENDDO
46 1 EXIT XDBF
47
48 2 ERR1:
49 2 CALL XRMSG TO DISPLAY ERROR
50 2 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
51 1 END XDBF
1 CD**********
1 CD0
1 CD0
1 CD0
582 1 CD0
583   FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
584 1 CD0
585
586 1 CD0
587   CALL XELD'S ('XLDEL')
588 1 CD0
589
590 1 CD0
591
592 1 CD0
593
594 1 CD**********
595 1 CD1
596 1 CD1
597 1 CD1
598 1 CD1
599 1 CD1
600 1 CD**********
601 1 CD4
602 1 CD4
603 1 CD4
604 1 CD4
605 1 CD4
606 1 CD4
607 1 CD4
608 1 CD4
609 1 CD**********
610 1 CD5
611 1 CD5
612 1 CD5
613 1 CD5
614 1 CD5
615 1 CD5
616 1 CD5
617 1 CD5
618 1 CD5
619 1 CD5
620 1 CD5
621 1 CD5
622 1 CD5
623 1 CD5
624 1 CD**********
1 BEGIN XLDEL
2 :PRMPT:
3 CALL XTCOM TO PROMPT FOR PROCESSOR NAME
4 IF XTCOM RETURN CODE IS NOT ZERO OR
5 RESPONSE IS NOT A VALID PROCESSOR NAME THEN
6 CALL XRMG TO WRITE ERROR MESSAGE
7 GO TO :PRMPT:
8 EXIT XLDEL IF RETURN CODE SAYS % ENTERED
9 ENDF:
10 CALL OPEN, READ AND CLOSE TO READ IN LIBRARY DIRECTORY
11 ERREXIT IF FILE ERROR TO :FILERR:
12 IF PROCESSOR IS NOT IN LIBRARY DIRECTORY THEN
13 CALL XRMG TO DISPLAY ERROR
14 GO TO :PRMPT:
15 ENDF:
16 DECIMATE # PROCESSORS BY 1
17 CALL PURGE TO PURGE OLD LIBRARY DIRECTORY
18 ERREXIT IF FILE ERROR TO :FILERR:
19 1 EXIT XLDEL IF #PROCESSORS IS ZERO
20 CALL CREAT, WRITF AND CLOSE TO RECREATE LIBRARY DIRECTORY
21 ERREXIT IF FILE ERROR TO :FILERR:
22 CREATE THE PROMPT FILE NAME
23 CALL PURGE TO PURGE THE PROMPT FILE
24 ERREXIT IF PURGE ERROR TO :FILERS:
25 CALL XRMG TO SAY FILE PURGED SUCCESSFULLY
26 IF PROCESSOR HAD AN INTERFACE TABLE THEN
27 CREATE DEFAULT IT NAME
28 CALL PURGE TO PURGE DEFAULT IF
29 ERREXIT IF RETURN CODE IS NOT ZERO TO :FILERR:
30 CALL XRMG TO DISPLAY 'FILE :AGED' MESSAGE
31 ENDF:
32 GO TO :PRMPT:
33 :FILERR: CALL XRMG TO WRITE FILE ACCESS ERROR
34 1 END XLDEL
FORTRAN CALLING SEQUENCE:
CALL XLIFL

XLIFL CREATES THE DEFAULT INTERFACE TABLE FILE

INTERNAL VARIABLES

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

FDS ROUTINES USED:
XREX, XMSG

RTE ROUTINES USED:
CLOSE, CREAT, WRITF

COMMON Used:

EQUIVALENCE (X)(1), ISECU),
+ (X)(14), ICR ), (X)(90), LITPR),
+ (X)(91), LITLEN),
+ (X)(96), MOPARM), (X)(101), MEMP ),
+ (X)(108), PARMS ), (X)(1), IERR ),
+ (X)(2), IDCB )
699 1 BEGIN XLIIFL
700 2 EXTRACT LITERAL LENGTH FROM HEADER
701 2 COMPUTE # BLOCKS FOR THIS FILE
702 2 CALL CREAT TO CREATE DEFAULT INT TABLE FILE
703 2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
704 2 CALL WRITF TO WRITE HEADER AND SPECS
705 2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
706 2 IF THERE IS A LITERAL RECORD THEN
707 3 CALL WRITF TO WRITE LITERALS
708 2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
709 2 ENDIF
710 2 CALL CLOSE
711 2 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
712 2 CALL XMMSG TO DISPLAY FILE (NAME) CREATED MESSAGE
713 1 EXIT XLIIFL
714 2 :FILERR:
715 2 CALL XMMSG TO WRITE FILE ACCESS ERROR
716 2 CALL XERIN TO RETURN TO MAIN PROGRAM
717 1 END XLIIFL
**FORTAN CALLING SEQUENCE:**

```
CALL XILNS
```

**INTERNAL VARIABLES:**

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

**RTE ROUTINES USED:**

```
EXEC
```

**COMMON USED:**

```
EQUIVALENCE (RE(1), LU)
```

```
1 BEGIN XILNS
2 CALL EXEC TO DISPLAY ALL 7 LINES
3 END XILNS
```
FORTRAN CALLING PROCEDURE:

CALL XELDS (XLINT)

XLINT SEGMENT SETS UP COMMON TO CALL THE INTERFACE TABLE TO

ACCEPT DEFAULT VALUES FOR THE INTERFACE TABLE

FDS FUNCTIONS AND SUBROUTINES USED:

XEXIT, XERTW, XIMIX, XANOV

COMMON USED:

EQUIVALENCE (REX5, RASESTA), (REX6, SUBSTA), (RBI1, NUMDR),
(REX2, DIREC), (RBI2, LSTFLG), (REX3, ARCHO), (RBI3, PANTP),
(REX4, ISSIZE), (RBI4, LRTDWN), (REX90, LITPHT), (RBI91, LITLEN),
(REX92, NAMP), (RBI97, NUMARG), (RBI100, NKDLNG), (REX1015, NKBUFF),
(REX1400, ENDB)
1 BEGIN XINT
2 SET MASTER STATE AS EXEC
3 SET SUBSTATE AS INTERFACE TABLE EDITOR
4 SET UP LIST OF VALID DIRECTIVES
5 SET UP LIST FLAG TO GET ENTIRE LIST
6 SET CURRENT ARGUMENT TO ZERO
7 SET PROMPT MODE TO ALL
8 SET INDEX INTO PARMS OF SHORT PROMPTS
9 SET NEW TABLE NAME TO ZERO
10 INITIALIZE ARRAY OF TYPE LENGTHS
11 IF LITERAL LENGTH IS ZERO THEN
12 SET LITERAL POINTER TO FIRST AVAILABLE WORD
13 ELSE
14 SET LITERAL POINTER TO FIRST WORD OF LITERALS
15 CALL XEINT TO UNPACK LITERALS
16 ENDIF
17 CALL XEINT TO ACCEPT DEFAULT VALUES
18 CALL ZERHN TO RETURN
19 END XINT
FORTRAN CALLING SEQUENCE:
CALL XLMOD ('XLMOD')

XLMOD ALLOWS A USER TO MODIFY THE VERSION, SPECS, AND PROMPTS FOR
A PROCESSOR AS WELL AS ADD/DELETE PARAMETERS FROM AN INTERFACE
TABLE.

RTE Routines Used:
CLOSE, IAND, OPEN, POSHT, READF, WRITE

FDS Routines Used:
XLFIL, XLINS, XLIPF, XLSPF, XRECP, XREXT,
XRMOV, XRMG, XRPEK, XRSET, XRUPK, XTCOM

Common Used:

EQUIVALENCE (XEC(3), ISECU),
  + (XEC(5), IASSA), (XEC(6), SUBSTA),
  + (XEC(7), VALFLG), (XEC(13), PKNAM),
  + (XEC(8), PRCWAM), (XEC(85), EDSTOK),
  + (XEC(86), INTTOK), (XEC(69), HARTOK),
  + (XEC(97), NECTOK), (XEC(142), ECR ),
  + (XEC(105), COMNUP), (XEC(260), VERS ),
  + (XEC(33), SYFLG ), (XEC(35), CFLAG ),
  + (XEC(37), ARNO ), (XEC(38), IT ),
  + (XEC(43), LEM ), (XEC(44), NDFLGS),
  + (XEC(88), LITBSP ), (XEC(90), LITPBR),
  + (XEC(92), IMOCK ), (XEC(96), NWOPAF),
  + (XEC(101), NWBR ), (XEC(103), LEHB ),
  + (XEC(104), LIDBO ), (XEC(105), NRMPS)
BEGIN XLMD
IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
SET RTN = 1

CALL XLNS TO DISPLAY SPEC INSTRUCTIONS
INITIALIZE MASTER AND SUB STATE FLAGS
CALL XTCOP TO PROMPT FOR PROCESSOR NAME
EXIT XLMD IF RETURN CODE SAYS % ENTERED
ERREXIT IF XICOM RETURN CODE NON-ZERO OR
ERREXIT IF INVALID PROCESSOR NAME (NOT CHAR NAME) TO :PRMERR:
CALL OPEN, READF AND CLOSE TO READ IN LIBRARY DIRECTORY
ERREXIT IF THERE IS A FILE ERROR TO :FILERR:
ERREXIT IF PROCESSOR IS NOT IN LIBRARY DIRECTORY TO :PRMERR:
SAVE INTERFACE TABLE BIT AND VERSION NUMBER
PERFORM VERSION TO UPDATE VERSION NUMBER
IF THE PROCESSOR HAD AN INTERFACE TABLE THEN
SET NEW VERSION NUMBER IN INTERFACE TABLE
CREATE DEFAULT INTERFACE TABLE NAME
CALL OPEN AND READF TO READ IN NEBR AND SPECS
IF THERE ARE LITERALS THEN
CALL READF TO READ IN LITERALS
ENDIF
CALL CLOSE TO CLOSE FILE
ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
CREATE PROMPT TABLE NAME
CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
CALL NAME TO RENAME PROMPT FILE >XLMP
ERREXIT IF NAME ERROR TO :FILERR:
SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPECS
PERFORM DELPM TO DELETE PARAMETERS
PERFORM MODPM TO MODIFY PARAMETERS
PERFORM ADDPM TO ADD PARAMETERS
CALL XLPL TO CREATE NEW PROMPT FILE
PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
ELSE
CALL NAME TO RENAME PROMPT FILE >XLMP
ERREXIT IF NAME ERROR TO :FILERR:
SET CODES ARRAY TO MODIFY ABSTRACT ONLY
CALL XLPL TO CREATE NEW PROMPT FILE
PERFORM XLMD - NO RETURN EXPECTED
ENDIF
ENDIF
CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
ERREXIT IF FILE ERROR TO :FILERR:
CALL XLXIF TO CREATE NEW DEFAULT INTERFACE TABLE FILE
SET VALFLE TO SAY ORIGINAL REQUEST TO MODIFY
PERFORM XLMD - NO RETURN EXPECTED

PRMERR:
CALL XMSG TO DISPLAY ERROR MESSAGE
GO TO (:PRON1,:PRON2,:PRON3,:PRON4,:PRON5,:PRON6,:PRON7),RTN

FILERR:
CALL XMSG TO DISPLAY FILE ERROR
END XLMD
905 1 BEGIN VERSION
906 2 SET RTN = 2

907 2 :PRM2:
908 2 CALL XCTOM TO PROMPT USER FOR VERSION NUMBER
909 2 IF RETURN CODE IS NOT CR THEN
910 3 ERREXIT IF RETURN CODE IS NON-ZERO CR
911 3 ERREXIT IF VERSION IS INVALID TO :PRM2:
912 3 IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
913 4 CALL XRSET TO PUT NEW VERSION IN ENTRY
914 4 CALL OPEN, WRITF, CLOSE TO UPDATE LIBRARY DIRECTORY
915 4 ERREXIT IF FILE ERROR TO :FILEEXIT:
916 3 ENDF
917 2 ENDF
918 1 END VERSION
919 1 =
920 1 =
921 1 =
922 1 BEGIN DELPRM
923 2 SET RTN = 3

924 2 :PRM3:
925 2 DO UNTIL RETURN CODE IS CR ENTERED
926 3 IF NUMBER OF PARAMETERS > 1 THEN
927 4 CALL XCTOM TO PROMPT FOR DELETE PARAMETER NAME
928 4 IF RETURN CODE IS NOT CR ENTERED THEN
929 5 PERFORM RSPMD TO INTERPRET RESPONSE
930 5 PERFORM CHDAT TO CHECK FOR EXISTING DATA
931 5 SET ARGMD TH WORD IN CODES TO SAY "DELETED"
932 5 CALL XERMOV TO MOVE DATA TO DELETE PARAMETER
933 5 DECERNENT # PARAMETERS BY 1
934 4 ENDF
935 3 ELSE
936 4 CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
937 4 EXIT DELPRM
938 3 ENDF
939 2 ENDOO
940 1 END DELPRM
941 1 =
942 1 =
943 1 =
944 1 BEGIN MODPRM
945 2 SET RTN = 4

946 2 :PRM4:
947 2 DO UNTIL RETURN CODE IS CR ENTERED
948 3 CALL XCTOM TO PROMPT FOR MODIFY PARAMETER NAME
949 3 IF RETURN CODE IS NOT CR ENTERED THEN
950 4 PERFORM RSPMD TO INTERPRET RESPONSE
951 4 PERFORM CHDAT TO CHECK FOR EXISTING DATA
952 4 SET ARGMD TH NON-DELETED WORD IN CODES TO SAY "MODIFIED"
953 4 CALL XERMOV TO MOVE USER FOR SPECIFICATIONS
954 4 SET IT COMPLETE BIT OFF
955 4 ENDF
956 2 ENDOO
957 1 END MODPRM
BEGIN RTM = 5

DO 1 = 1, 5

1  K = A$(K)

IF K = "A" THEN SET RTM = 1
ELSE IF K = "B" THEN SET RTM = 2
ELSE IF K = "C" THEN SET RTM = 3
ELSE IF K = "D" THEN SET RTM = 4
ELSE IF K = "E" THEN SET RTM = 5

END IF

END RTM = 7

IF RTM = 1 THEN CALL ADDPAR
ELSE IF RTM = 2 THEN CALL MODIFY
ELSE IF RTM = 3 THEN CALL DELETE
ELSE IF RTM = 4 THEN CALL DEFAULT
ELSE IF RTM = 5 THEN CALL Defaults

END IF

END RTM = 7

BEGIN RTM = 5

END RTM = 1

END
**FORTRAN CALLING PROCEDURE**

CALL YELDS ('XMSG ')

**MESSAGE PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XMSG**

**INPUT**

**COMMON - LUL, ISEQU, FLAGS, TOKENS, ICR**

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

**UPDATE MODE, MESSAGE NUMBER AND TEXT**

**MESSAGE FILE - DIRECTORY AND OLD TEXT**

**OUTPUT**

**COMMON - COMBUF**

**MESSAGE FILE - DIRECTORY AND TEXT UPDATES**

**LOCAL**

**AREA - NUMERICAL AREA INDICATOR FOR MESSAGE**

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

**INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY**

**FILE MANAGER DATA CONTROL BLOCK**

**FILE MANAGER & XTCOM RETURN CODE**

**MESSAGE NUMBER WITHIN MESSAGE AREA**

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

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

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

**NOTES**

USES APOSN, CLOSE, CREATION, EXEC, IAND, KCVT, OPEN, READF, WRITF,

XERN, XRIG, XRMV, XMSG, XTCOM, XUDBG

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

**MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X**
1067 1 BEGIN XLMSG
1068 2   OPEN XLMSG
1069 3   IF FILE NOT FOUND
1070 4   THEN
1071 5     OUTPUT 'MESSAGE FILE CREATION'
1072 6     DO FOR EACH OF THE 32 DIRECTORY ENTRIES
1073 7       PROMPT FOR AREA ID AND MAXIMUM NUMBER OF MESSAGES FOR THIS ENTRY NUMBER
1074 8       IF RESPONSE WAS NULL
1075 9       THEN
1076 10      CLEAR ENTRY
1077 11    ELSE
1078 12      STORE ID
1079 13      COMPUTE AREA ORIGIN AND STORE
1080 14      CLEAR LAST MESSAGE NUMBER
1081 15      COMPUTE NUMBER OF BLOCKS AND STORE
1082 16    ENDIF
1083 17 ENDDO
1084 18 CREATE A CLEARED FILE OF TOTAL REQUIRED SIZE
1085 19 ELSE
1086 20   READ DIRECTORY
1087 21 ENDIF
1088 22 DO UNTIL USER INPUTS "X"
1089 23   PROMPT FOR MESSAGE NUMBER
1090 24   SEPARATE AREA AND MESSAGE NUMBER AND COMPUTE BLOCK NUMBER AND MESSAGE LOC
1091 25   IF VALID AREA AND BLOCK NUMBER <= NUMBER OF BLOCKS
1092 26 THEN
1093 27     READ BLOCK
1094 28     IF FIRST WORD OF MESSAGE IS NOT NULL (MESSAGE ALREADY EXISTS)
1095 29     THEN
1096 30        DISPLAY OLD MESSAGE TEXT
1097 31    ENDIF
1098 32    PROMPT FOR TEXT
1099 33    IF NON-NULL RESPONSE
1100 34 THEN
1101 35     STORE TEXT IN BLOCK
1102 36     REWRITE BLOCK
1103 37     IF MESSAGE NUMBER > LAST MESSAGE NUMBER
1104 38 THEN
1105 39       REPLACE LAST MESSAGE NUMBER WITH NEW NUMBER
1106 40    ENDIF
1107 41    ENDIF
1108 42 ELSE
1109 43   OUTPUT 'XL29 AREA INVALID OR NUMBER TOO LARGE'
1110 44 ENDIF
1111 45 ENDDO
1112 46   REWRITE DIRECTORY BLOCK
1113 47 END XLMSG
1115 1 C0************
1116 1 C00
1117 1 C00 FORTRAN CALLING SEQUENCE:
1118 1 C00 CALL XLPCR
1119 1 C00
1120 1 C00 C0************
1121 1 C01 XLPCR CREATES AND INITIALIZES THE PDB LOG FILE
1122 1 C01
1123 1 C01 C0************
1124 1 C02 INPUTS FROM COMMON:
1125 1 C02
1126 1 C02 XE(3) ISECU, XE(142) ICR
1127 1 C02
1128 1 C02 C0************
1129 1 C03 RTE ROUTINES USED:
1130 1 C03
1131 1 C03 C0************
1132 1 C04 CLOSE, CREAT, WRITF
1133 1 C04
1134 1 C04 COMMON USED:
1135 1 C04
1136 1 C04 C0************
1137 1 C05 FDS ROUTINES USED:
1138 1 C05
1139 1 C05 C0************
1140 1 C06 COMMON USED:
1141 1 C06
1142 1 C07 C0************
1143 1 C08 EQUIVALENCE
1144 1 C08 + (XE(85), TOKENS), (XE(142), ICR )
1145 1 C08 + (XE(145), COMBUF), (XB(99), IERR )
1146 1 C08 + (XB(100), ICOMP ), (XB(116), IBUF )
1147 1 C08 C0************
1 BEGIN XLCR
2 CALL CREAT TO CREATE MDB/PDB LOG FILE
3 ERREXIT IF CREATE ERROR TO :FILERR;
4 INITIALIZE LOG RECORD BUFFER TO ZEROS
5 SET # MDB FILES CURRENTLY USED TO ZERO
6 SET MAXIMUM NUMBER MDBS TO 20
7 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
8 ERREXIT IF WRITE ERROR TO :FILERR;
9 DO FOR EACH REMAINING MDB RECORD
10 DO FOR EACH PAIR OF USER ID'S
11 CALL XCOM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
12 SET MAXIMUM # PDB FILES TO RESPONSE
13 SET # PDB FILES CURRENTLY USED TO 0
14 ENDDO
15 CALL WRITE TO WRITE 1 PDB RECORD TO LOG FILE
16 ERREXIT IF WRITE ERROR TO :FILERR;
17 ENDDO
18 CALL CLOSE TO CLOSE FILE
19 ERREXIT IF CLOSE ERROR TO :FILERR;
20 EXIT XLCR

21 :FILERR:
22 CALL XMSG TO DISPLAY FILE ERROR
23 CALL CLOSE TO CLOSE FILE
24 END XLCR
FORTRAN CALLING SEQUENCE:

CALL XLPL (NOCOD, CODES)

XLPL CREATES PROMPT FILE FOR PROCESSORS. IT ALSO RE-CREATES IT WHEN A PROCESSOR IS MODIFIED.

INPUTS IN CALLING SEQUENCE:

NOCOD - NUMBER OF CODES IN THE CODE ARRAY
CODES - ARRAY OF CODES THAT REPRESENT:
CODES(1) PROCESSOR ABSTRACT
CODES(2) THRU CODES(NOCOD) PARAMETERS 1 THRU N

WITH VALUES OF:
0 - NO CHANGE
1 - MODIFIED
2 - DELETED
3 - ADDED

INPUTS IN COMMON:

XE(1) LU, XE(3) ISCU, XE(16) PRCHAM,
XE(142) ICN, XE(96) NOPARM, XE(108) PARMS

RTE FUNCTIONS USED:
CLOSE, CREAT, EXEC, OPEN,
POSN, PUNGE, READF, WRITF

FDS FUNCTIONS USED:
XERTH, XORCP, ZPMOV, ZRMGS,
XRPCK, XORPK, XICOM

COMMON USED:

EQUIVALENCE (XE(1), LU )
+ (XE(3), ISCU ), (XE(77), VALFL6),
+ (XE(16), PRCHAM ), (XE(85), TOKEMS),
+ (XE(142), ICN ), (XE(144), COMPTK),
+ (XE(145), COMBUS ), (XE(96), NOPARM),
+ (XE(108), PARMS)

WARNING:
The correctness of the XLPL source code is not guaranteed.

For more details, refer to the source code or contact the developer.
BEGIN XLFL
1227  COMPUTE SIZE OF FILE AS 0 PARAMETERS *3
1228  CALL OPEN TO CREATE PROMPT FILE
1229  ERREXIT IF OPEN ERROR TO :FILERR:
1230  STUFF SYNTAX RECORD 0 AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS
1231  CALL WRITE TO WRITE SHORT PROMPT RECORD(S)
1232  ERREXIT IF WRITE ERROR TO :FILERR:
1233  IF ABSTRACT CODE IS MODIFIED THEN
1234      CALL OPEN TO OPEN OLD PROMPT FILE >XLTMP
1235      ERREXIT IF OPEN ERROR TO :FILERR:
1236      CALL READ TO READ EXISTING ABSTRACT
1237      ERREXIT IF READ ERROR TO :FILERR:
1238      CALL EXEC TO DISPLAY EXISTING ABSTRACT
1239      CALL XCTOM TO PROMPT USER TO MODIFY ABSTRACT
1240      IF RESPONSE IS CR (NO MODIFICATION) THEN
1241        CALL WRITE TO WRITE EXISTING ABSTRACT TO NEW FILE
1242        ERREXIT IF WRITE ERROR TO :FILERR:
1243      ELSE (NEW ABSTRACT WAS ENTERED)
1244        PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1245        ENSDF
1246      ELSE (ABSTRACT CODE IS AD)"
1247        CALL XCTOM TO PROMPT USER TO ENTER NEW ABSTRACT
1248        PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1249      ENDF
1250      IF PARAMETERS IS NOT ZERO THEN
1251          PERFORM EXPRN TO GET DEFINITIONS FOR EACH PARAMETER
1252          ENSDF
1253      ENSDF
1254      CALL CLOSE TO CLOSE PROMPT FILE
1255      ERREXIT IF CLOSE ERROR TO :FILERR:
1256      IF THERE WAS AN OLD FILE THEN
1257        CALL CLOSE TO CLOSE OLD FILE >XLTMP
1258        CALL PURGE TO PURGE OLD FILE >XLTMP
1259      ENSDF
1260      EXIT XLFL:
1261  :FILERR:
1262  CALL XERMSG TO DISPLAY ERROR CODE
1263  CALL CLOSE TO CLOSE NEW FILE
1264  CALL CLOSE TO CLOSE OLD FILE >XLTMP
1265  CALL PURGE TO PURGE OLD FILE >XLTMP
1266  SET VALFLG = 0
1267  CALL XERWTO RETURN TO MAIN ***NO RETURN TO HERE***
1268  END XLFL
1270 1 BEGIN FORMAT
1271 2 INITIALIZE TOKEN POINTER AND TOTAL WORD COUNT
1272 2 DO UNTIL EOS IS DETECTED IN RESPONSE
1273 2 ERREXIT IF RESPONSE IS NOT CHARACTER STRING TO :ERR1:
1274 2 ERREXIT IF RESPONSE IS TOO LONG (>128 WORDS) TO :ERR1:
1275 2 CALL INMOV TO MOVE RESPONSE TO BUFFER
1276 2 SET CONTROL CHARACTERS IN BUFFER
1277 2 INCREMENT TOTAL WORD COUNT BY THIS RESPONSE
1278 2 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
1279 2 INCREMENT TOKEN POINTER TO NEXT CHARACTER STRING
1280 2 ENDDO
1281 2 SET REMAINDER OF BUFFER TO NULL
1282 2 CALL WRTIF TO WRITE NEW RESPONSE TO PROMPT FILE
1283 2 ERREXIT IF WRTIF ERROR TO :FILEERR:
1284 1 END FORMAT
1285 1 :ERR1:
1286 1 CALL ERRMSG TO DISPLAY INVALID RESPONSE
1287 1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
BEGIN EXTRMN
DO FOR EACH ENTRY IN CODES ARRAY
BEGIN CASE (MOD, DEL, ADD), CODES+1
   WHEN (MOD) (SPECs WERE MODIFIED)
      CALL READ TO READ EXISTING DEFINITION FROM >XLTMP
      HDERR IF READING ERROR TO :FILERR:
      CALL EXEC TO DISPLAY EXISTING DEFINITION
      IF RESPONSE IS CR (NO RESPONSE) THEN
         CALL WRITE TO WRITE EXISTING DEFINITION TO NEW FILE
         HDERR IF WRITING ERROR TO :FILERR:
         ELSE (NEW DEFINITION WAS ENTERED)
            PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
            ENDIF
   WHEN (DEL) (SPECs WERE DELETED)
      CALL POSN TO POSITION >XLTMP OVER THIS ENTRY
      HDERR IF POSN ERROR TO :FILERR:
   WHEN (ADD) (SPECs WERE ADDED)
      CALL XT.COM TO PROMPT USER TO ENTER NEW DEFINITION
      PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
   DO UNTIL RESPONSE IS CR (EXIT)
      CALL XT.COM TO PROMPT USER FOR SHORT PROMPT TO MODIFY DEFINITION
      EXIT EXTRMN IF RESPONSE IS CR
      SEARCH SHORT PROMPTS FOR RESPONSE
      HDERR IF NOT FOUND TO :ERR:
      CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
      HDERR IF READING ERROR TO :FILERR:
      CALL EXEC TO DISPLAY EXISTING DEFINITION
      CALL XT.COM TO PROMPT USER TO MODIFY DEFINITION
      IF RESPONSE IS POSITIVE
         CALL POSN TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
         HDERR IF POSN ERROR TO :FILERR:
         PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
         ENDIF
   ENDIF
END CASE
END
CO*************
CO FORTRAN CALLING SEQUENCE:
CO CALL XLPMO
CO*************
CO XLPMO MODIFIES THE EXISTING PDB LOG FILE
CO*************
CO INPUTS FROM COMMON:
CO XE(1) LU, XE(3) QUAL, XE(8) SECU,
CO XE(142) ICR, XE(99) IERX, XE(100) INCB,
CO*************
CO INTERNAL VARIABLES IN COMMON:
CO XE(52) UNO - NUMBER OF USER ID A-Z (1-26)
CO XE(53) RECNO - RECORD # WHERE UNO'S PDB LIST IS (2-14)
CO XE(52) PHAM - FILE NAME OF USER'S PDB (FULL 6 CHAR)
CO XE(55) DIFF - NUMBER OF FILES THAT MUST BE DELETED
CO WHEN NEW PDB file CURRENTLY IN USE
CO XE(244) OUTBUF-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.
CO*************
CO RTE ROUTINES USED:
CO CLOSE, EXEC, KEVT, PURGE, READF, WRTF
CO XLPMO
CO*************
CO FDS ROUTINES USED:
CO XLPMO
CO*************
CO COMMON USED:
CO*************
CO EQUIVALENCE (XE(1), LU ),
CO + (XE(3), QUAL ), (XE(8), SECU ),
CO + (XE(85), TOWENS), (XE(142), ICR ),
CO + (XE(144), LCOMPT ), (XE(143), COMMFB ),
CO + (XE(50), UNO ), (XE(51), RECNO ),
CO + (XE(52), PHAM ), (XE(55), DIFF ),
CO + (XE(56), RTN ), (XE(57), MSG ),
CO + (XE(99), IERX ), (XE(100), INCB ),
CO + (XE(116), IBUFF ), (XE(244), OUTBUF ),
CO + (XE(304), IBCD02 )
CO*************
1387 1 BEGIN XLPMO
1388 2 DO UNTIL USER RESPONSE IS EXIT (Y)
1389 3 CALL XTCOM TO PROMPT USER FOR USER ID
1390 4 ERRERT IF RESPONSE IS INVALID TO :ERR1:
1391 5 SAVE USER ID AND NUMBER (1-26) IN COMMON
1392 6 CALL READ TO READ IN THE RECORD CONTAINING THIS ID
1393 7 ERRERT IF READ ERROR TO :FILERR:
1394 8 DISPLAY MAX # ALLOWED FOR THIS USER AND # CURRENTLY USED
1395 9 CALL XTCOM TO PROMPT USER FOR NEW MAXIMUM
1396 10 IF RESPONSE IS A MODIFICATION (NOT CR) THEN
1397 11 ERRERT IF RESPONSE IS NOT VALID TO :ERR1:
1398 12 COMPUTE DIFFERENCE AS # CURRENTLY USED - NEW MAXIMUM
1399 13 IF DIFFERENCE <= 0 THEN
1400 14 CALL WRTT TO REWRITE UPDATED RECORD TO FILE
1401 15 ERRERT IF WRITE ERROR TO :FILERR:
1402 16 ELSE
1403 17 DISPLAY LIST OF CURRENT FILES
1404 18 SET #PURGED = 0
1405 19 DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1406 20 CALL XTCOM TO PROMPT USER TO DELETE # PURGED FILES
1407 21 DO FOR EACH FILE NAME IN RESPONSE
1408 22 SEARCH LIST FOR FILE NAME
1409 23 ERRERT IF INVALID NAME TO :ERR1:
1410 24 MOVE FILE NAME TO PURGE LIST
1411 25 INCREMENT #PURGED BY 1
1412 26 DECREMENT DIFFERENCE BY 1
1413 27 COMPRESS OLD FILE NAME OUT OF LIST
1414 28 ENDDO
1415 29 ENDDO
1416 30 IF RESPONSE WAS NOT EXIT (N) THEN
1417 31 CALL WRTT TO REWRITE UPDATED LOG RECORD
1418 32 ERRERT IF WRITE ERROR TO :FILERR:
1419 33 DO FOR #PURGED FILES
1420 34 CALL XRGN TO CREATE FILE NAME
1421 35 CALL PURGE TO PURGE FILE
1422 36 ENDDO
1423 37 ENDDI
1424 38 ENDDO
1425 39 ENDDO
1426 40 CALL CLOSE TO CLOSE NDB/PDB LOG FILE
1427 41 EXIT XLPMO
1428 42 :ERR1:
1429 43 CALL XRMSG TO DISPLAY ERROR IN RESPONSE
1430 44 RETURN TO REISSUE LAST PROMPT
1431 45 :FILERR:
1432 46 CALL XRMSG TO DISPLAY FILE ERROR
1433 47 CALL CLOSE TO CLOSE NDB/PDB FILE
1434 48 ENDPXLPMO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

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

ROUTINES USED:
KCVT, XRCP, XRMOV, XRMSP, XRSST, XTCM

XE COMMON USED:

EQUIVALENC (XE(8), RETC), (XE(9), RETH),
+ (XE(10)); CLASS), (XE(11), TYPE ),
+ (X(145), COMBF), (X(37), ARGIN ),
+ (X(96), WOPARM), (X(108), PARMS )
1474 1 BEGIN XLSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
1476 2 :PRMPT1:
1477 2 SET RETURN TO 1
1478 CALL XTCOM TO PROMPT FOR PROMPT,CLASS,TYPE AND I/O FLAGS
1479 ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
1480 INITIALIZE THIS SHORT PROMPT AND ENTRY
1481 ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
1482 ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
1483 CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
1484 ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
1485 CALL XRSET TO SET CLASS IN ENTRY
1486 ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
1487 ERREXIT IF CLASS IS ORDE AND TYPE IS SYMBOIC STRING TO :PRMERR:
1488 CALL XRSET TO SET TYPE IN ENTRY
1489 ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
1490 ERREXIT IF TYPE IS SYMBOIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
1491 CALL XRSET TO SET I/O FLAGS IN ENTRY
1492 IF CLASS IS DATA ELEMENT THEN
1493 3 IF TYPE IS NOT SYMBOIC STRING THEN
1494 4 SET RETURN TO 2
1495 4 :PRMPT2:
1496 4 CALL XTCOM TO PROMPT FOR I AND J DIMENSIONS
1497 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1498 4 ERREXIT IF I DIMENSION IF NOT VALID TO :PRMERR:
1499 4 I: J DIMENSION IS NOT ENTERED THEN
1500 5 SET J DIMENSION TO 1
1501 5 ELSE
1502 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 7 ENDIF
1509 4 SET SIZE =I DIMENSION * J DIMENSION * TYPE LENGTH
1510 4 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1511 3 ELSE
1512 4 SET RETURN TO 3
1513 4 :X:
1514 4 CALL XTCOM TO PROMPT FOR MAXIMUM SIZE
1515 4 ERREXIT IF RETURN CODE IS NOT NORMAL TO :PRMERR:
1516 4 ERREXIT IF MAXIMUM SIZE IS NOT VALID TO :PRMERR:
1517 4 SET MAXIMUM SIZE INTO IDIM FIELD OF ENTRY
1518 3 ENDIF
1519 2 ENDIF
1520 1 EXIT XLSPS
1521 2 :PRMERR: CALL XRMSG TO DISPLAY ERROR MESSAGE
1522 2 GO TO (:PRMPT1,:PRMPT2,:PRMPT3), RETURN
1523 1 END XLSPS
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
**FDX Manager**

1. **FDX Manager**. Scheduled by FDS Configuration Manager. One Program per signed on User.

2. **INPUT**
   - **FD5** FDS Status Table (See SDD 6.2.2)
   - **FD6** FDS Management Requests (See SDD 6.2.6.2)

3. **OUTPUT**
   - **FD7** FD Manager Responses (See SDD 6.2.6.3)

4. **NOTES**
   - **FD8** USES XMAN, XMAVA, XMPAW, XMSCH, XMXT, XDMP
   - **FD9** RMPAR, SLIBX, SLIBR, ANRR, CHKUSD, XES

5. **XGVA** is a Manager Global Data Area Containing the XAWA and Various Control Values

6. **BEGIN XGVR**

7. **END**
61 6 * I1=REQUEST & NN IS LU
62 6 * SET UP FOR FDS TERMINATION
63 5 ENIF
64 5 EXIT TO :XEND
65 4 ENDF
66 3 ENDF
67 2 ENDDO

:XEND
68 2 IF DUMP OPTION THEN
69 3 CALL XDUMP TO DUMP OUR PARTITION
70 2 ENIF
71 2 RELEASE DWA TRACK SPACE
72 2 FREE EXEC'S & PROCESSOR'S CLASS NUMBERS
73 2 CALL RRNR SET LOCK ON STATUS TABLE
74 2 UNLOCK THE MANAGER FROM THE PARTITION
75 2 CALL SLIBR DISABLE
76 2 CLEAR EXEC'S ID SEGMENT
77 2 CALCULATE EQT ADDRESS FOR ATTENTION ID
78 2 CLEAR THE STATUS TABLE ENTRY
79 2 CALL SLIBR ENABLE
80 2 CALL RRNR CLEAR THE LOCK ON THE STATUS TABLE
82 2 ISSUE MESSAGE '***XMDZ SIGN OFF FOR LU "NN"'
83 2 CALL SLIBR GET PRIVLEDGED AND DISABLED
84 2 CALL SLIST TO MAKE US DORMANT
85 2 CLEAR OUR OWN ID
86 2 RESET EQT TO REMOVE ATTENTION ID
87 1 EXIT :SEXQ TO THE RTE DISPATCHER
88 1 END
90 1 *D0  CALLING PROCEDURE
91 1 *D0  JSB XMXQT
92 1 *D0
93 1 ***********
94 1 *D1  SEQUENCE TABLE EXECUTION FROM &SEQTB
95 1 *D1
96 1 ***********
97 1 *D2  INPUT
98 1 *D2  XMAWA, XMPRM, XVSTA
99 1 *D2
100 1 ***********
101 1 *D3  OUTPUT
102 1 *D3  XMCP, XMPRM, XVSTA
103 1 *D3
104 1 ***********
105 1 *D5  ROUTINES USED
106 1 *D5  CNM, EXEC, XMANG, XMKIL, XMPAW (XMSCH),
107 1 *D5  XMSST, XMTFN, SLIBR, SLIB2
108 1 *D5
109 1 ***********
BEGIN XNXT
CALL XNTM.FIND INSET TO ENTRY
FIND ADDRESS OF INSET
CALL XNNT TO CONVERT ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
CALL XNXT 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 XNTM 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 SCHEDULED
ERREXIT TO :ERROR: IF NOT FOUND
CALL SLIBR TO BE PRIVILEGED
SET CURRENT TASK TO THIS PROCESSOR
CALL SLIBX TO BE UN-PRIVILEGED
IF PROCESSOR REQUIRES AN INTERFACE TABLE THEN
CALCULATE LENGTH OF INTERFACE TABLE HEADER AND SPECs
CALL EXE TO CLASS 1/0 WRITE HEADER AND SPECs
ENDIF
CALL EXE TO SCHEDULE PROCESSOR WITH WAIT
CALL XASCH TO RETRIEVE PARAMETERS FROM PROCESSOR
CALL XNTF TO FIND INSET TO ENTRY
FIND ADDRESS OF INSET
COMPUTE CURRENT ENTRY ADDRESS
DO WHILE PROCESSOR REQUESTS AWA MANAGEMENT (PARMS = 1)
CALL XNXT TO HONOR AWA REQUEST
ENDDO
CLEAR OUT CLASS BUFFER FROM LAST PROCESSOR
ERREXIT TO :ERROR: IF PROCESSOR REQUESTED TERMINATION (PARMS = 8)
ERREXIT TO :ERROR: IF PROCESSOR ABENDED (PARMS = -32768)
IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PARMS = 3) THEN
ERREXIT TO :ERROR: IF RESET NUMBER IS ZERO
CALL XNNT TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY · DISPLACEMENT
ERREXIT TO :ERROR: IF RESET SEQUENCE NUMBER IS NOT FOUR OF
IF TERMINAL ENTRY WAS JUST EXECUTED THEN
SET UP PARM TO SHOW RESET SEQUENCE NUMBER
PERFORM :END: - **NO RETURN EXPECTED**
ENDIF
ELSE
IF REQUEST IS NOT NORMAL COMPLETION (PARMS = 0) THEN
DISPLAY ERROR MESSAGE - INVALID REQUEST
ERREXIT TO :ERROR: IF CURRENT AT IS DORMANT
CALL XNXT TO SET PARAMETERS TO ABEND ASSOCIATED TASK
CALL XMPAR TO RESCHEDULE PROCESSOR
ERREXIT TO :ERROR: TO TERMINATE SEQUENCE
ENDIF
INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
ENDDO
171 2 SET PARM1 = 0 (NORMAL COMPLETION)
172 2 PERFORM :END: - NO RETURN EXPECTED
173 2 :ERROR1: SET PARM1 = 1
174 2 :ERROR2: SET PARM1 = 8 AND PARM5 TO APPROPRIATE REASON CODE
175 2 :END:
176 2 CALL SLIBR TO BECOME PRIVLEDGED
177 2 SET CURRENT TASK IN GR, STATUS TABLE, AND ANA TO EXEC
178 2 CALL SLIBX TO BECOME UN-PRIVLEDGED
179 2 CALL XMPAM TO POST EXEC AND WAIT FOR NEXT REQUEST
180 1 END XMXQT
CALLING PROCEDURE

JSB XMAFR
DEF **3
DEF ADDR
DEF SIZE

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

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

INPUT

ADDR - ADDRESS OF AREA BEING FREED
SIZE - SIZE OF AREA BEING FREED. IF LESS THAN 3 THE FOLLOWING
WORKS WILL ALSO BE FREED SUCH THAT THE MINIMUM FE SIZE
OF 3 WORDS IS MAINTAINED.

EXTERNAL SYMBOLS FROM XMAWA

XMBCP, XMFCP, XMFNC, XMFRE

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

OUTPUT (EXTERNAL SYMBOLS FROM XMAWA)

XMBCP, XMFCP, XMFNC, XMFRE

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

NOTES

USES .ENTR
1 BEGIN XMAFR  
2 SET NEW FE SIZE FIELD TO MAX(SIZE, 3)  
3 INCREMENT TOTAL FREE SPACE BY NEW FE SIZE  
4 INDEX TO FORWARD CHAIN POINTER (FCP) HEAD  
5 START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)  
6 EXIT IF 'ADDR' < FCP VALUE  
7 SET NEW FE FCP TO CURRENT FCP VALUE  
8 SET CURRENT FCP VALUE TO 'ADDR'  
9 SET NEXT FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE  
10 SET NEXT FE'S BCP VALUE TO 'ADDR'  
11 PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE  
12 OR ELSE  
13 INDEX TO NEXT FE FCP  
14 END LOOP  
15 SET NEXT FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)  
16 SET CURRENT FE'S FCP VALUE TO 'ADDR'  
17 SET NEW FE BCP TO BCP HEAD VALUE  
18 SET BCP HEAD TO 'ADDR'  
19 END SEARCH  
20 IF NEW FE BCP NOT = END-OF-CHAIN  
21 THEN  
22 PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE  
23 END IF  
24 IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW (OR CONSOLIDATED) FE, THEN  
25 MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE  
26 ENDIF  
27 1 END XMAFR  
28 1 BEGIN MERGE  
29 IF FE 1 IS ADJACENT TO FE 2  
30 THEN  
31 INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD  
32 SET FE 1 FCP TO VALUE OF FE 2 FCP  
33 IF FE 1 FCP NOT = END-OF-CHAIN  
34 THEN  
35 SET FE 3 BCP TO ADDRESS OF FE 1  
36 ELSE  
37 SET BCP HEAD TO ADDRESS OF FE 1  
38 ENDIF  
39 2 ENDIF  
40 1 END MERGE
CALLING PROCEDURE

255 400  JEE XMAGT
256 400  DEF *+3
257 400  DEF OPTM
258 400  DEF SIZE
259 400  DEF
260 400  DEF
261 400  DEF
262 400  DEF
263 400  DEF
264 400  DEF
265 400  DEF
266 400  DEF
267 400  DEF
268 400  DEF
269 400  DEF
270 400  DEF
271 400  DEF
272 400  DEF
273 400  DEF
274 400  DEF
275 400  DEF
276 400  DEF
277 400  DEF
278 400  DEF
279 400  DEF
280 400  DEF
281 400  DEF
282 400  DEF
283 400  DEF
284 400  DEF
285 400  DEF
286 400  DEF
287 400  DEF
288 400  DEF
289 400  DEF
290 400  DEF
291 400  DEF
292 400  DEF
293 400  DEF

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

INPUT

OPIN - INDICATOR OF WHICH FREE CHAIN TO SEARCH
O = FORWARD POINTER CHAIN (FOR TDC SPACE)
1 = BACKWARD POINTER CHAIN (FOR DATA SPACE)
SIZE - NUMBER OF WORDS NEEDED (A MINIMUM OF 3 WORDS WILL BE
ALLOCATED EVEN IF 'SIZE' IS 1 OR 2)
EXTERNAL SYMBOLS FROM XMAWA
XMFCP, XMFNC, XMFC

OUTPUT

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

NOTES
IF SOME FE EXACTLY 'SIZE' WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN
ERROR RETURN (A-REG = -32768) IS TAKEN
USES .EMTR
1 begin xmsrc
2 set index to appropriate chain head, i.e., pemhead(opth)
3 start search while pointer not = end-of-chain (-32768), and
4 while toc space fence has not been crossed
5 exit if fe size = max('size', 3)
6 decrement total free space by max('size', 3)
7 decrement fe
8 return address of area
9 exit if fe size > max('size', 3) + 3
10 decrement total free space by max(size, 3)
11 if allocating from head of space (opth = 0)
12 then
13 create chain pointers and size fields in bottom of space
14 rechain new fe
15 return address of area
16 else
17 change size field to fe size - max('size', 3)
18 compute and return address of area
19 endif
20 or else
21 index to next fe
22 end loop
23 set return code to 0(not found)
24 end search
25 if toc space was found at the toc space fence, then
26 increment toc space fence by 'size'
27 endif
28 end xmsrc
**CALLING PROCEDURE**

347 1 *00 PROVIDE AWA MANAGEMENT BASED ON REQUEST LIST (SEE FDS SDO
348 1 *01 TABLE 6.2 - III)
349 1 *01 TABLE 6.2 - III)
350 1 *01 TABLE 6.2 - III)
351 1 *01 TABLE 6.2 - III)
352 1 *01 TABLE 6.2 - III)
353 1 *01 TABLE 6.2 - III)
354 1 *01 TABLE 6.2 - III)
355 1 *01 TABLE 6.2 - III)
356 1 *01 TABLE 6.2 - III)
357 1 *01 TABLE 6.2 - III)
358 1 *01 TABLE 6.2 - III)
359 1 *01 TABLE 6.2 - III)
360 1 *01 TABLE 6.2 - III)
361 1 *01 TABLE 6.2 - III)
362 1 *01 TABLE 6.2 - III)
363 1 *01 TABLE 6.2 - III)
364 1 *01 TABLE 6.2 - III)
365 1 *01 TABLE 6.2 - III)
366 1 *01 TABLE 6.2 - III)
367 1 *01 TABLE 6.2 - III)
368 1 *01 TABLE 6.2 - III)
369 1 *01 TABLE 6.2 - III)
370 1 *01 TABLE 6.2 - III)
371 1 *01 TABLE 6.2 - III)
372 1 *01 TABLE 6.2 - III)
373 1 *01 TABLE 6.2 - III)
374 1 *01 TABLE 6.2 - III)
375 1 *01 TABLE 6.2 - III)
376 1 *01 TABLE 6.2 - III)
377 1 *01 TABLE 6.2 - III)
378 1 *01 TABLE 6.2 - III)

**NOTES**

379 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
380 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
381 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
382 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
383 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
384 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
385 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
386 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
387 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
388 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN
389 1 *05 USES EXE, XMFR, XMCT, XMPK, XMTN

**INPUT**

390 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
391 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
392 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
393 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
394 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
395 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
396 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
397 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
398 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST
399 1 *01 CLINO - CLASS I/O NUMBER CONTAINING REQUEST LIST

**OUTPUT**

400 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
401 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
402 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
403 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
404 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
405 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
406 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
407 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
408 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)
409 1 *01 ID SEGMENT PARAMETERS (SEE TABLE 6.2 - IV)

**REQUEST LIST FIELD 8**

410 1 *03 REQUEST LIST FIELD 8
411 1 *03 REQUEST LIST FIELD 8
412 1 *03 REQUEST LIST FIELD 8
413 1 *03 REQUEST LIST FIELD 8
414 1 *03 REQUEST LIST FIELD 8
415 1 *03 REQUEST LIST FIELD 8
416 1 *03 REQUEST LIST FIELD 8
417 1 *03 REQUEST LIST FIELD 8
418 1 *03 REQUEST LIST FIELD 8
419 1 *03 REQUEST LIST FIELD 8

**NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS**

420 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
421 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
422 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
423 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
424 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
425 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
426 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
427 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
428 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS
429 1 *05 NEITHER XEXEC NOR THE PROCESSORS WILL MAKE AWA MANAGEMENT REQUESTS

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

430 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
431 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
432 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
433 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
434 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
435 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
436 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
437 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA
438 1 *05 FOR DMA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DMA

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

439 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
440 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
441 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
442 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
443 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
444 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
445 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
446 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
447 1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO

**THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.**

448 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
449 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
450 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
451 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
452 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
453 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
454 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
455 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
456 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
457 1 *05 THE DMA AND COPY THEM BACK TO THE AWA AS NECESSARY.
1 BEGIN XNAMG
2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER
3 CLEAR RETURN PARM1
4 CLEAR RETURN PARM2
5 INITIALIZE TO FIRST REQUEST CODE
6 DO UNTIL END OF LIST (0), EIGHT REQUESTS PROCESSED OR PARM1 > ZERO
7 IF REQUEST FOR TOC (CODE 10)
8 THEN
9 IF REQUESTED SIZE .GT. TOC SIZE, THEN
10 CALL XNAPK TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
11 ENDIF
12 WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O
13 STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT
14 SET REQUESTED SIZE FROM TOC SIZE
15 SET REWRITE FLAG
16 ELSE
17 IF REQUESTS TO CLEAR (CODE 1?)
18 THEN
19 GET A($DWA) FROM XMDWA
20 SAVE HEADER AND DIRECTORY-SIZE
21 CLEAR XMDWA THRU XMDWO
22 BUILD AN FE AT XMDWA FOR AWA SIZE
23 CALL XNAMG TO ALLOCATE A TOC ENTRY FOR $DWA
24 CHAIN IN TOC ENTRY TO XMDWO
25 CALL XNAMG TO ALLOCATE SPACE FOR $DWA
26 SET LOCATION, SIZE, & KEY IN THE TOC
27 SET DIRECTORY ADDRESS AT XMDWA
28 CLEAR THE DIRECTORY
29 SET LU, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY
30 ELSE
31 CALL XMTFN TO SEARCH TOC FOR INDICATED ENTRY
32 CASE (:VERIFY:, :VERALO:, :VERALO:, :RNAME:, :DELVER:, :DELVER:, :STORE:)
33 (;RETRIEVE:, ;RETRIEVE:) REQUEST CODE
34 :VERIFY:
35 IF ENTRY NOT FOUND
36 THEN
37 SET RETURN PARM1 AND PARM2 (2 & INDEX)
38 ENDIF
39 :VERALO:
40 IF ENTRY ALREADY EXISTS
41 THEN
42 IF ALLOCATE REQUEST (3)
43 THEN
44 SET RETURN PARM1 AND PARM2 (3 & INDEX)
45 ELSE
46 IF TYPE, SIZE AND 1-DIM FIELDS DO NOT MATCH
47 THEN
48 SET RETURN PARM1 AND PARM2 (4 & INDEX)
49 ENDIF
50 ENDIF
51 ELSE
52 CALL XNAMG TO ALLOCATE TOC SPACE
53 IF CLASS EQ 3 OR 8, THEN
54 CHAIN IN NEW TOC ENTRY
55 SET DATA SPACE ADDRESS TO ZERO
56 ELSE
57 CALL XNAMG TO ALLOCATE DATA SPACE
IF SPACE NOT AVAILABLE
   THEN
   SET RETURN PARM1 AND PARM2 (1 & INDEX)
   ELSE
      IF DATA ELEMENT (CLASS 2)
         THEN
            IF CHARACTER STRING (TYPE 4 - 8)
               THEN
                  INITIALIZE AREA TO BLANKS
               ELSE
                  INITIALIZE AREA TO ZEROS
               ENDIF
            ELSE
               CHAIN IN NEW TOC ENTRY
            ENDIF
         ENDIF
      ELSE
         IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE
         THEN
            CALL XMNL DWA ALLOCATION
         ENDIF
      ENDIF
   ENDIF
ENDIF

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

:DELIVER:
IF ENTRY FOUND
   THEN
      GENERATE KEY 1 LESS THAN FOUND KEY
      CALL XMTPN FOR GENERATED KEY
      DECHAIN TOC ENTRY
      CALL XMAGF TO RETURN TOC ENTRY SPACE TO FE POOL
      CALL XMAGF TO RETURN DATA SPACE TO FE POOL
      IF CLASS EQ 4 OR 6, THEN
         CALL XMDCD DWA DEALLOCATION
      ENDIF
   ELSE
      IF DELETE REQUEST (5)
      THEN
         SET RETURN PARM1 AND PARM2 (2 & INDEX)
      ENDIF
   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 XMODST; DMA STORE DATA
      ENDIF
    ENDIF
  ENDIF

RETRIEVE:
  IF ENTRY NOT FOUND
  THEN
    SET RETURN PARM1 AND PARM2 (2 & INDEX)
  ELSE
    IF VALUES REQUESTED (8)
    THEN
      IF (TOC TYPE .NE. 0 AND .NE. REQUESTED TYPE) , OR
      DISPLACEMENT OR SPECIFIED SIZE < 0, OR
      DISPLACEMENT + SPECIFIED SIZE > ALLOCATED SIZE
      THEN
        SET RETURN PARM1 AND PARM2 (4 & INDEX)
      ELSE
        IF REQUESTED SIZE = ZERO
        THEN
          CALCULATE AMOUNT OF DATA TO RETRIEVE AS ACTUAL SIZE MINUS DISPLACEMENT
          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 XMODRT MOVE INTO AMA
        IF NO SPACE THEN
          SET RETURN PARM1 AND PARM2 TO(1, INDEX)
          EXIT TO :XMRK
        ENDIF
        ENDIF
        WRITE VALUES TO CLASS I/O
        STORE TYPE IN LOW BYTE OF REQUEST WORD 1
        STORE CLASS NUMBER IN REQUEST WORD 1:CHT
        SET WRITE FLAG
      ENDIF
    ELSE
      WRITE TOC ENTRY TO CLASS I/O
      STORE CLASS NUMBER IN REQUEST WORD 8:CHT
      SET WRITE FLAG
      :XMRK
      ENDIF
  ENDIF
550       6        ENDIF
551       5        END CASE
552       4        ENDIF
553       3        ENDIF
554       3        INCREMENT TO NEXT REQUEST CODE
555       2        ENDDO
556       2        STORE REWRITE FLAG IN RETURN PARM5
557       2        IF REWRITE FLAG SET
558       2          THEN
559       3          WRITE REQUEST LIST BACK TO COMMUNICATIONS CLASS I/O NUMBER
560       2          ENDF
561       2          PEND ASSOCIATED TASK WITH RETURN PARM5
562       1        ENW XHANG
564 1 *00      CALLING PROCEDURE
565 1 *00
566 1 *00      JSB XMTFF
567 1 *00      DEF ++2
568 1 *00
569 1 *00
570 1 *00
571 1 *00
572 1 *00      EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'
573 1 *00
574 1 *00
575 1 *00
576 1 *00
577 1 *00
578 1 *00      EXTERNAL SYMBOLS FROM XNANA
579 1 *00
580 1 *00
581 1 *00
582 1 *00
583 1 *00
584 1 *00
585 1 *00
586 1 *00
587 1 *00
588 1 *00
589 1 *00
590 1 *00
591 1 *00
592 1 *00
593 1 *00
594 1 *00
595 1 *00
596 1 *00
597 1 *00
598 1 *00
599 1 *00
600 1 *00
601 1 *00
602 1 *00
603 1 *00
604 1 *00
605 1 *00
606 1 *00
607 1 *00
608 1 *00
609 1 *00      END XMTFF

Calling Procedure
611 1 *D0   CALLING PROCEDURE
612 1 *DO
613 1 *DO JSB XMSST
614 1 *DO
615 1 ********
616 1 *D1   ADJUST SEQUENCE LOCATION COUNTER
617 1 *D1
618 1 ********
619 1 *D2   INPUT
620 1 *D2
621 1 *D2   A-REG - INTEGER VALUE OF SEQUENCE NUMBER TO BE LOCATED
622 1 *D2
623 1 *D2   EXTERNAL SYMBOLS
624 1 *D2   SRAD - ADDRESS OF EXECUTING SEQUENCE TABLE IN AWA
625 1 *D2   ENDB - DISPLACEMENT TO LAST ENTRY OF EXECUTING SEQUENCE TABLE
626 1 *D2
627 1 ********
628 1 *D3   OUTPUT
629 1 *D3
630 1 *D3   A-REG - DISPLACEMENT TO ENTRY IF SEQUENCE NUMBER IS FOUND, ELSE -32768
631 1 ********
632 1 *
633 1 *
634 1 *
635 1 *
636 1 BEGIN XMSST
637 2 SAVE RESET SEQUENCE NUMBER IN LOCAL VARIABLE SEGNO
638 2 SET DISPLACEMENT IN A-REG TO ZERO
639 2 GET ADDRESS OF FIRST ENTRY IN SEQUENCE TABLE INTO X-REG
640 2 START SEARCH UNTIL CURRENT ENTRY DISPLACEMENT > LAST ENTRY DISPLACEMENT
641 2 EXIT IF INDICATED NUMBER FOUND
642 2 OR ELSE
643 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 XPAY
650 2   *00 ENTRY: JSB XPAY OR CALL XPAYM
652 2   *00 DEF RETURN ADDRESS
653 2   *00 INPUTS: XMPPM HAS REPLY
653 2   *00 OUTPUTS: XMPPM HAS REQUEST,
655 2   *00 AND XVS8 IS UPDATED.
660 2   *00 DO UNTIL GOOD PARMS RECEIVED OR TOP AT TERMINATES
661 3     IF ABORT CURRENT Flag CN(- STB) THEN
662 4         SET ABORT FLAG IN XMPPM
663 4         TURN OFF ABORT CURRENT FLAG
664 3     ENDIF
665 3     GET CURRENT TOP ASSOCIATED TASK(AT)
666 3     SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
667 3     JSB XVPAM
668 3     DEF *+3 RETURN
669 3     DFC 0 MANAGER CALL
670 3     DEF XMPPM PARM FIELD
671 3 * THIS IS AN IMPLIED WAIT
672 3 :XMPMCH GET CURRENT XVS8 ENTRY (XUSTA)
673 3   *00 IF THERE HAS BEEN A CALL TO PRTN (P1 FIELD IS NOT TOP AT) OR
674 4     IL SEGMENT IS DORMANT OR
675 4     ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
676 4     IF PARM IS NOT 0,3,8,9, OR -32768 THEN
677 5         SET PARM TO -32768
678 5 ENDIF
679 4   * PRINT ERROR "INVALID REQUEST"
680 5 ELSE
681 5 SET PARM FIELD FROM MANAGER'S ID SEGMENT
682 5 ELSE
683 5   * TOP AT IS STILL ACTIVE AND RETURNED VIP PAM
684 5 IF PARM NOT 1 OR 2 THEN
685 5   * CALL XMPIL TO ABORT TOP AT
686 5 ELSE
687 5     PRINT ERROR "INVALID REQUEST"
688 5 ELSE
689 5     SET PARM FROM CURRENT ID SEGMENT
690 5 ENDIF
691 1 END XPAY
CALLING PROCEDURE

JSO XMDIN

FUNCTION

INITIALIZE TOC

& DMA FUNCTIONS

OUTPUT

BREG O=COMPLETE

MINUS= ERROR IN INITIALIZATION

NOTES

USES EXEC DISC ALLOCATION,

XMTFN, XNAGT

BEGIN XMDIN

GET NUMBER OF DMA TRACKS FROM P5

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

CALL XMTFN (EDWA)

CALL XNAGT (0,8) TOC ENTRY FOR EDWA

CALL XNAGT (1,SIZE) DATA AREA FOR EDWA

INITIALIZE & CHAIN EDWA TOC ENTRY

SET # OF TRACKS FOR EDWA

CLEAR EDWA

CALL EXEC (DISC TRACK ALLOCATION)

IF DISC ADDRESS .EQ. -1 TRACKS NOT AVAILABLE

THEN

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

ERROR RETURN

ELSE

SET DISC ADDRESS IN EDWA

SET ADDRESS OF EDWA FOR DMA MANAGEMENT

ENDIF

E0MDIN
1 BEGIN XNAL
2 * DMA ALLOCATION
3 * CALLING PROCEDURE
4 JSB XNAL
5 * FUNCTION
6 * ALLOCATE DMA SPACE
7 * INPUTS
8 A(TOC ENTRY OF THE DMA ELEMENT)
9 IN TREG
10 * OUTPUTS
11 UPDATE TOC ENTRY FOR "WA"
12 * RETURNS REG= ZERO- ALLOCATION COMPLETE
13 = minus- ERROR CONDITION
14 NOTES
15 CALLS XNRA,XNBD
16 IF (BDWA) .NE. 0, THEN
17 SET START-ADDRESS TO FIRST TRACK WORD IN DMA DIRECTORY
18 GET SIZE(IN WORDS FROM TOC ENTRY)
19 SIZE(IN SECTORS) = (SIZE+63/64)
20 DO FOR NUMBER OF TRACKS IN DMA OR DMA SIZE .GE. SIZE
21 SET BITNUM TO ZERO
22 DO UNTIL DMA SIZE .GE. SITE OR BITNUM .GE. 96
23 CALL XRDB (0,BITNUM,START-ADDRESS)
24 STARTBIT =BITNUM
25 CALL XRDB (1,BITNUM,START-ADDRESS)
26 DMA-SIZE =BITNUM-STARTBIT
27 END=O
28 IF BITNUM .GE. 96, THEN
29 START-ADDRESS+START ADDRESS+1 TRACK ADDRESS
30 ENDIF
31 END=O
32 IF 01: 'C AREA FOUND, THEN
33 START-A= 'F5' & BITNUM DEFINES THE TRACK & SECTOR
34 SET DISC ADDRESS & SIZE IN TOC ENTRY
35 CALL XPST (TOC-ENTRY)
36 SET NORMAL RETURN
37 ELSE
38 ENDIF
39 ELSE
40 ISSMESSAGE '***XNOS NO DMA SPACE REMAINING'
41 SET ERROR RETURN
42 ENDIF
43 ENDIF
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>778</td>
<td>BEGIN XMODA</td>
</tr>
<tr>
<td>779</td>
<td>** BEGIN ** XMODA</td>
</tr>
<tr>
<td>780</td>
<td>** BEGIN ** XMODA</td>
</tr>
<tr>
<td>781</td>
<td>BEGIN XMODA</td>
</tr>
<tr>
<td>782</td>
<td>CALLING PROCEDURE</td>
</tr>
<tr>
<td>783</td>
<td>JSB XMODA</td>
</tr>
<tr>
<td>784</td>
<td>CALLING PROCEDURE</td>
</tr>
<tr>
<td>785</td>
<td>JSB XMODA</td>
</tr>
<tr>
<td>786</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>787</td>
<td>DELETE DMA ELEMENT WHICH CORRESPONDS</td>
</tr>
<tr>
<td>788</td>
<td>TO THE DMA ELEMENT</td>
</tr>
<tr>
<td>789</td>
<td>INPUTS</td>
</tr>
<tr>
<td>790</td>
<td>TOC ENTRY ADDRESS OF DMA ELEMENT</td>
</tr>
<tr>
<td>791</td>
<td>IN TREG</td>
</tr>
<tr>
<td>792</td>
<td>NOTES</td>
</tr>
<tr>
<td>793</td>
<td>USES XMNST</td>
</tr>
<tr>
<td>794</td>
<td>IF AESWMA ) .NE. 0 , THEN</td>
</tr>
<tr>
<td>795</td>
<td>GET DISC ADDRESS</td>
</tr>
<tr>
<td>796</td>
<td>GET DATA SIZE</td>
</tr>
<tr>
<td>797</td>
<td>FREE THE DISK AREA</td>
</tr>
<tr>
<td>798</td>
<td>CALL XMNST (TOC ENTRY)</td>
</tr>
<tr>
<td>799</td>
<td>ENDIF</td>
</tr>
<tr>
<td>800</td>
<td>END XMODA</td>
</tr>
</tbody>
</table>
1 BEGIN XHOST
2 * DWA STORE DATA CALLING PROCEDURE
3 * JSB XHOST
4 * FUNCTION
5 * UPDATE DWA ELEMENT WHICH CORRESPONDS TO THE AWA ELEMENT
6 * INPUTS
7 * TBC ENTRY ADDRESS OF AWA ELEMENT
8 * IN TREG
9 * OUTPUTS
10 * UPDATE DWA ELEMENT ON DISC TRACKS
11 * NOTES
12 * USES WRITE
13 IF A(EQVA) .NE. 0, THEN
14 SET DISC ADDRESS FOR THE WRITE
15 SET DATA ADDRESS FOR THE WRITE
16 WRITE DATA
17 ENDIF
18 END XHOST
BEGIN XHRT
  10 DWA RETRIEVE CALLING PROCEDURE
  20 JSB XHRT
  30 FUNCTION
  40 RETRIEVE DWA DATA INTO AWA
  50 INPUTS
  60 A(TOC ENTRY) IN YREG
  70 OUTPUTS
  80 ADDRESS OF DATA IN THE TOC
  90 BREG=0, RETRIEVE SUCCESSFUL
 100 MINUS, ERROR NO AWA DATA
 110 NOTES
 120 USES XHRT,INDATA,EXEC(READ)
 130 IF NO DATA DIRECTORY, THEN
 140 SET ERROR CODE -5
 150 ELSE
 160 CALL XHRT GET DATA SPACE
 170 IF NO SPACE, THEN
 180 SET ERROR CODE -1
 190 ELSE
 200 SET DATA ADDRESS IN TOC
 210 GET DISC ADDRESS
 220 READ DATA INTO AWA
 230 SET RETURN CODE TO 0
 240 ENDIF
 250 ENDIF
 260 END XHRT
1 BEGIN XMAPK
2 *00 CALLING PROCEDURE
2 *00 JSB XMAPK
2 *00 DEF OPTION
2 *00 DEF PHASE 1 FLAG
2 *00 GENERAL COLLAPSE INTERFACE
2 *00 OUTPUT
2 *00 RETURNS AREA ADDRESS FOR AREA FOUND
2 *00 RETURNS -32768 AREA NOT FOUND
2 *00 NOTES
2 *00 USES XMPK1, XMPK2, XMPK3, XMSRC
2 *05
2 IF PHASE1 FLAG NOT SET, THEN
2 CALL XMPK1 PURGE DWI ELEMENTS FROM AWA
2 PERFORM XMSRC(OPTN, SIZE)
2 ELSE
2 SET RETURN CODE TO NOT FOUND
2 ENDIF
2 IF RETURN CODE IS NOT FOUND, THEN
2 IF OPTN=1 (BACKWARD CHAIN), THEN
2 CALL XMPK2 PACK TOC CHAIN
2 CALL XMSRC(OPTN, SIZE)
2 ENDIF
2 IF RETURN CODE IS NOT FOUND, THEN
2 CALL XMPK3 PACK AWA DATA AREAS
2 PERFORM XMSRC(OPTN, SIZE)
2 ENDIF
2 SET RETURN VALUE TO RETURN CODE
1 END XMAPK
1 BEGIN XPAR
2    CALLING PROCEDURE
3    JSB XPAR

PHASE 2 OF COLLAPSE(COLLAPSE TOC ENTRIES)

OUTPUT
REORDERS TOC ENTRIES
UPDATES XMFC
TOC FENCE ADDRESS

NOTES
USES XMFC, XMSCP, XMFTM, XMCP, XMFR, XPAR

INCREMENT XPAR   UPDATE PHASE 2 COUNT
DO WHILE XMFC .GE. XMSCP AND XMFC .NE. -32768
* ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
* WILL A TOC COMPRESSION BE DONE.
GET FIRST-FREE(FIRST-FREE) FROM XMFC
GET JODSZ FROM THE FREE ELEMENT
NWSIZ=OLDSZ-B
IF NWSIZ .GT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
GET NEXT-FREE FROM FIRST-FREE'S FCP
SET INTO XMFC
DELETE FROM THE FORWARD CHAIN
IF XMFC .NE. -32768, THEN IF NOT THE LAST FREE ELEMENT
SET NEXT-FREE'S BCP TO -32768
ELSE
SET XMCP TO -32768 DELETE FROM BACKWARD CHAIN:
ENDIF
ELSE
OLDSZ=NWSIZ   UPDATE LENGTH IN FREE ELEMENT
ENDIF
NWTFC=(FIRST-FREE)=NWSIZ   THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
OLTOC=XMFNC-B   OLD TOC IS THE ENTRY MOVE THE FENCE
TMPKEY=OLTOC'S KEY - 1   CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
CALL XMFTM(TMPKEY)   FIND PREVIOUS
SET PTROC FROM ARG
COPY TOC ENTRY FROM OLTOC TO NWTFC
STORE NWTFC ADDRESS INTO PTROC'S CHAIN
CALL XMFR(OLTOC, TOC SIZE)   FREE THE OLD TOC ENTRY
END IF
END XPAR
BEGIN XMPK3  
CALLING PROCEDURE
JSB XMPK3
FUNCTION
COMPRESS AWA DATA AREA BY MIGRATING DATA AREAS
TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
OUTPUT
UPDATES XMBCP, XMFCF, XMFCN, XMFC, XMPK3
NOTES
USES XMAFR

INCREMENT XMPK3 (NUMBER OF PHASE 3, J)

IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
CALL XMPK2, INSURE TOC IS COMPRESSED
DO WHILE XMBCP .GT. XMFCN UNTIL 1 FREE AREA ADJACANT TO THE FENCE
IF HCP OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
SET HIGH-WATER TO XMFCN
ELSE
MULTIPLE AREAS
SET HIGH-WATER TO HCP OF LAST-FREE
ENDIF
SET LOW-WATER TO A(LAST-FREE)
PERFORM XMPK2(LOW-WATER,HIGH-WATER)
EXIT IF TOC-ADDRESS .EQ. 0
PERFORM XMAFR(TOC-ADDRESS)
ENDDO
ENDIF
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 (NONE FOUND).
1069  2  * FIRST TOC ENTRY IS AT SYMBOL XMAWA, THE
1070  2  * LAST TOC-ENTRY IS AT XMFNC-8.
1071  2  TOC-ENTRY=XMAWA)
1072  2  TEST-AD=0; TEST-TOC=0
1073  2  DO UNTIL TOC-ENTRY .GE. XMFNC
1074  3  IF DATA ADDRESS IN TOC-ENTRY:
1075  4    NE 0, AND IS
1076  4    LT LOW-WATER, AND IS
1077  4    GT HIGH-WATER, AND IS
1078  4    GT TEST-AD,
1079  3  THEN
1080  4  TEST-AD=DATA ADDRESS
1081  4  TEST-TOC=TOC-ENTRY
1082  3  ENDF
1083  3  ADD 8 TO TOC-ENT
1084  2  ENDDO
1085  2  TOC-ADDRESS=TEST-TOC  RETURN 0 OR A TOC ADDRESS
1086  1 END XMTSC
1088 1 BEGIN XMANV  AWA MOVE
1089 2 * MOVE THE DATA DEFINED BY THE TOC(WHICH IS
1090 2 * IMMEDIATELY ABOVE THE LAST FREE AREA) INTO
1091 2 * THE BOTTOM OF THE LAST FREE
1092 2 * UPDATE THE LENGTH OF THE RESULTING FREE AREA.
1093 2 GET DATA ADDRESS FROM THE TOC
1094 2 SAVE THE FIRST THREE WORDS OF THE DATA AREA
1095 2 CALL XMAFR(DATA ADDRESS,SIZE)
1096 2 GET FREE AREA FROM XMBEP
1097 2 DECREASE FREE AREA LENGTH BY MAX(DATA SIZE,3)
1098 2 CALCULATE NEW ADDRESS FROM FREE AREA + FREE LENGTH
1099 2 MOVE DATA FROM DATA ADDRESS TO NEW ADDRESS
1100 2 MOVE SAVED FIRST THREE WORDS TO NEW ADDRESS
1101 2 UPDATE DATA ADDRESS IN TOC WITH NEW ADDRESS
1102 1 END XMANV
FORTRAN CALLING PROCEDURE

CALL XPATR (LU, INTBUF, OPTN, LBBUFF, IMUN, NAME, TYPE, SIZE,
IDIN, DSPRT)

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

INPUT
LU - LOGICAL UNIT NUMBER OF USER TERMINAL
INTBUF - INPUT/OUTPUT BUFFER OF 7*(0 PARAMETERS + 1) WORDS,
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTLN - LENGTH OF INTBUF
MBUFF - MANAGER REQUEST BUFFER (64 WORDS) USED TO COMMUNICATE
WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACCROSS PROCESSOR SERVICE CALLS.
IMUN - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE WHOSE
ATTRIBUTES ARE REQUESTED.

OUTPUT
NAME - AN ALPHANUMERIC NAME OF UP TO SIX CHARACTERS WHICH
IDENTIFIES THE DATA ELEMENT OR DUE 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
SUBSCRIBED OR ZERO IF NOT SUBSCRIBED,
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DDE OR
ZERO IF OUTPUT DDE
IDIN - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DDE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIBED OUTPUT.
DSPTY - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIBED DATA ELEMENTS ELSE ZERO.
FOR AN INPUT DDE THE RTE FILE MANAGER TYPE CODE IS
RETURNED IN DSPRT.

EXTERNAL SYMBOLS
(SEE XPEGT)

INTERNAL VARIABLES

NOTES
USES .ENH, XPE13(XPEGT), XPINI(XPEGT), XPMFN(XPEGT), XPREQ,
APSCI(XPEGT), XPST, XVSTD
XPATR IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
POINTS XPATR, XPEGT AND XPPUT
1 BEGIN XPATR
2 PERFORM XPINI(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
3 EXIT TO :XP:15: (XPGET) IF PARAMETER IS OUT OF RANGE
4 EXTRACT NAME FROM INTERFACE TABLE
5 SET DSPTT TO ZERO
6 IF LITERAL PARAMETER (NAME IS ZERO)
7 THEN
8 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
9 ELSE
10 IF SUBSCRIPTED (INTERFACE TABLE DISP OR S FIELDS ARE NON-ZERO)
11 THEN
12 PERFORM XPICE(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
13 STORE TYPE, SIZE, IDIM AND DSPTT
14 ELSE
15 ENDIF
16 THEN
17 PERFORM XPIMW(XPGET) TO QUALIFY FILE NAME
18 IF INPUT PARAMETER
19 THEN
20 CALL XPREG TO RETRIEVE TOC ENTRY
21 COPY TYPE, SIZE, IDIM AND DSPTT FROM TOC ENTRY
22 ELSE
23 ENDIF
24 END
25 END IF
26 ENDIF
27 END XPATR
FORTRAN CALLING PROCEDURE

CALL XPGET (LU, INTOBUF, INTOLEN, RBUF, N, IMSN, IM(1), ..., IN(R))

XGETT ALLICTS PROCESSORS TO OBTAIN DATA FROM THE FDS VIA A.
VERIFIES THE EXISTENCE OF THE DATA ELEMENTS AND ORDERS
REFERENCED BY THE INTERFACE TABLE AND RETRIEVES INPUT DATA
BY ONE OF THE FOLLOWING WAYS:

1) REQUESTED DATA FROM LITERAL DATA STORED IN THE
INTERFACE TABLE IS RETURNED OR
2) REQUESTED DATA FROM A DATA ELEMENT REFERENCED BY
THE INTERFACE TABLE IS RETURNED OR
3) THE NAME OF THE ORDER REFERENCED BY THE INTERFACE
TABLE WITH A / SYMBOL AS A PREFIX AND A ONE CHARACTER
USER CODE AS A SUFFIX IS RETURNED SO THE CALLING
PROGRAM MAY RETRIEVE THE DATA.

DATA ELEMENT AND ORDER NAMES ARE SEARCHED FOR IN THE AWA TOC.
IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING TERMINATED.

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

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL
INTBUF - INPUT/OUTPUT BUFFER FOR L/C PARAMETERS + 1 WORDS.
INTLEN - 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
RBUF - MANAGER REQUEST BUFFER (4 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCANNING
AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
N - NUMBER OF PARAMETERS TO BE RETRIEVED. IF N=0, ALL
PARAMETERS ARE REQUESTED AND IMSN IS NOT USED.
IMSN - 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 IM(1)

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

OUTPUT

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

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

INTERNAL VARIABLES

AMULT - ARRAY OF 10 BIT MULTPLIERS FOR USE BY THE
PROCEDURE SMART IN DETERMINING WORD DISPLACEMENTS FOR
VARIOUS TYPES OF SUBSCRIPTED ARRAYS
CLSF - ADDRESS OF THE CLASS FIELD (SECOND WORD) OF THE REQUEST
ARRAY REQUEST
STPT - FLAG INDICATING GET OR PUT PROCESSING. VALUES CORRESPOND
TO BIT SETS FOR INPUT OR OUTPUT FLAGS IN THE FIFTH
WORD OF INTERFACE TABLE ENTRIES

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

1 00
97 1 00
94 1 00
91 1 00
88 1 00
85 1 00
82 1 00
79 1 00
76 1 00
73 1 00
70 1 00
67 1 00
64 1 00
61 1 00
58 1 00
55 1 00
52 1 00
49 1 00
46 1 00
43 1 00
40 1 00
37 1 00
34 1 00
31 1 00
28 1 00
25 1 00
22 1 00
19 1 00
16 1 00
13 1 00
10 1 00
7 1 00
4 1 00
1 1 00

155 1 *04 PROCESSING LOOP
156 1 *04 MP = VALUE OF "P", WHERE P IS THE NUMBER OF PARAMETERS IN THE
157 1 *04 INTERFACE TABLE
158 1 *04 XNAME - ARRAY OF THE NAME FIELD (THIRD WORD) OF THE REQUEST
159 1 *04 ARACT - ARRAY REQUEST
160 1 *04 NARY = NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING
161 1 *04 SEQUENCE
162 1 *04 RECUR = EIGHT WORD ARRAY USED FOR CONSTRUCTING ANA MANAGEMENT
163 1 *04 REQUESTS FOR XPRED
164 1 *04 USRFD = USER FILE IDENTIFIER CHARACTER (I=PARM(3))
165 1 *04 EXTERNAL VARIABLES (SEE XPRED)
167 1 *04 XPLCS
168 1 *04 XPLU
169 1 *04 XPRG
170 1 *04
171 1 **********
172 1 NOTES
173 1 *05 USES .ENTRY, EXEC, XPRED, XPRIT, XUSTB
174 1 *05
175 1 *05 XPRED MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.
176 1 *05 EPM INTRUFT MUST BE USED BY XPRED, XPUTT, AND XPATH AND XEODS
177 1 *05 TO BE INITIALIZED ONLY ONCE BY ANY OF THE THREE ROUTINES.
178 1 *05 SINCE REQUESTS FROM INPUT DATA FROM THE ANA MAY BE PASSED FAPU UPTO
179 1 *05 EIGHT AT A TIME, XPRED RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE
180 1 *05 REQUESTED IN MULTIPLES OF EIGHT.
181 1 *05
182 1 *05
183 1 **********
FORTRAN CALLING PROCEDURE

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

XPPUT ALLOWS PROCESSORS TO STORE DATA INTO DATA ELEMENTS AND
ORDER'S REFERENCED IN THE INTERFACE TABLE AS FOLLOWS:

1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC,
   IF FOUND, DATA IS STORED
   IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING
   TERMINATES.

2) NAME IS SEARCHED FOR IN THE TOC. IF THE NAME IS
   NOT FOUND, IT IS ADDED TO THE TOC. IF THE NAME IS
   FOUND OR AFTER BEING ADDED TO THE TOC, THE NAME IS
   PREFixed WITH A / SYMBOL, SUFFixed WITH A ONE

CHARACTER USER CODE AND RETURNED SO THE CALLING
PROGRAM CAN STORE DATA.
BEGIN XPGET
SET FOR "GET"
PERFORM ACCESS TO RETRIEVE DATA
END XPGET
BEGIN XPGET
SET FOR "PUT"
PERFORM ACCESS TO STORE DATA
END XPGET
BEGIN XPGET
PERFORM SPNI TO INITIALIZE GLOBALS AND INTERFACE TABLE
DO FOR EACH PARAMETER REQUESTED
IF SELECTED PARAMETER IS "O" OF RANGE
CALL XPRED TO PURGE QUEUE REQUESTS
EXIT TO :XPE13;
ENDIF
IF INPUT/OUTPUT TYPE DOES NOT MATCH "GET"/"PUT" PROCESSING
THEN
CALL XPRED TO PURGE QUEUE REQUESTS
EXIT TO :XPE12;
ENDIF
IF OVER-RUNNING CALLING SEQUENCE
THEN
CALL XPRED TO PURGE QUEUE REQUESTS
EXIT TO :XPE13;
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 (XPFPF = -1)
THEN
IF SUBSCRIPTED (DISPLACEMENT > 0 OR DOUBLE SUBSCRIPT FLAG SET)
THEN
PERFORM XPSC TO COMPUTE DISPLACEMENT = (DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST WITH NAME AND DISPLACEMENT
ENDIF
CALL XPRED TO QUEUE RETRIEVAL AND STORAGE OF INPUT
ELSE PARAMETER IS DDE (CLASS 3)
BUILD REQUEST FOR DDE ENTRY
CALL XPRED TO IMMEDIATELY RETRIEVE DDE ENTRY
PERFORM XPFRM 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 (XPFPF = -1)
THEN
ENDIF
ENDIF
IF SUBSCRIPTED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING DRD WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRD
CALL XPREQ TO QUEUE ALLOCATION
ENDIF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
ENDIF
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AHA
CALL XPREQ TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS ORDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING ORDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD RF ST TO REALLOCATE ORDE WITH NEW ATTRIBUTES
EXIT TO :ERR14: IF FILE TYPE NOT 1-15, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
CALL XPREQ TO QUEUE REALLOCATION OF ORDE
ENDIF
ENDIF
ENDDO
CALL XPREQ TO COMPLETE QUEUED REQUESTS
EXIT ACCESS
:ERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:ERR13: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:ERR14: CALL XPREQ TO PURGE QUEUED REQUESTS
:ERR15: TERMINATE PROCESSOR FOR INVALID ORDE OR DRD FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
**FORTRAN CALLING PROCEDURE**

**CALL XPGET (LU, INBUF, INTLNG, MBBuff, INUM, IN, SIZE, DISP)**

**INPUT**

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

**INBUF** - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 1) WORDS,

**INTLANG** - LENGTH OF INBUF

**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**

**BDSMP** - 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

**JTPT** - DISPLACEMENT INTO JTPT DEPENDING ON WHETHER GET (G) OR PUT (P) PROCESSING IS INVOLVED

**JGPT** - TWO WORD VECTOR CONTAINING 'JSB XPGET' OR 'JSB XPPUT'

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

**NOTES**

**USES EXEC, .ENTR, XPATR, XPE13(XPATR), XPGET(XPATR), XPPTR(XPATR)**

**USES EXEC, .ENTR, XPATR, XPE13(XPATR), XPGET(XPATR), XPPTR(XPATR)**
FORTRAN CALLING PROCEDURE

CALL XPAR (LU, INBUF, INTLG, MRBUF, IN, OUT, SIZE, DISP)

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

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INBUFF - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 1) WORDS,

INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON

FIRST USE TO CAUSE INITIALIZATION

LENGTH OF INBUFF

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

RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE INTO

WHICH TO STORE DATA

ARRAY OF AT LEAST SIZE WORDS FROM WHICH OUTPUT DATA IS TO

BE TAKEN

TOTAL NUMBER OF WORDS TO BE STORED

DISP - DISPLACEMENT FROM THE BEGINNING OF THE OUTPUT PARAMETER

AT WHICH TO BEGIN DATA STORAGE. A VALUE OF ZERO

INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER

IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT

BE SPECIFIED, I.E., STORAGE INTO AN AREA PREVIOUS TO THE

USER'S SUBSCRIPTS IS NOT SUPPORTED.

OUTPUT

NONE

SEE XPAR

NOTES

XPARTI IS AN ENTRY POINT INTO XPAR

************
CALLING PROCEDURE

JSB XPREQ

WHERE N IS THE NUMBER OF ACTUAL ARGUMENTS

DEF +N+1

REQUIRED OPTION

DEF OPIN

REQUIRED REQUEST

DEF ADRES

OPTIONAL ADDRESS (SEE OPTIN)

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

XPGET/XPPUT BUFFERED AXA MANAGEMENT SERVICE

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

IN-UT

OPTIN - OPTION WORD

SIGN BIT - 0 QUEUE REQUEST AND RETURN

DQUE REQUEST AND CLOSE BUFFER

DQUE REQUEST AND TRANSFER DATA TO/FROM

ADRES

DQUE REQUEST ONLY

'ERST - EIGHT WORD REQUEST (SEE SDD 6.2.6.2)

ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT

VALUES USED ONLY WHEN OPTIN(RIGHT BIT) = 0

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

EXTERNAL SYMBOLS

XPLU - TERMINAL LOGICAL UNIT NUMBER

XPROB - ADDRESS OF 64 WORD MANAGER REQUEST BUFFER

XPCLS - ADDRESS OF I/O NUMBER FOR MANAGER COMMUNICATIONS

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

OUTPUT

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

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

LOCAL VARIABLES

ADD - ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDs)

ADD - TABLE OF ADDRESSES FOR STORAGE OF FETCHED DATA

APRAM - ADDRESS OF PARS AREA

CLASS - CLASS NUMBER USED TO TRANSMIT DATA TO MANAGER

PIARS - PARAMETER RETURN AND SCRATCH AREA

PIR - SAVE AREA FOR ADD and XPROB POINTER

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

NOTES

USES .SENTR, EXEC, XPIIT, XPAR

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

************
1 BEGIN XPREG
2 MOVE REQUEST INTO BUFFER
3 IF OPTION IS TO TRANSFER DATA
4 THEN
5 IF REQUEST IS TO RETRIEVE DATA (B)
6 THEN
7 STORE ADDRESS IN TABLE
8 ELSE SHOULD BE A REQUEST TO STORE DATA (7)
9 OUTPUT DATA TO CLASS I/O
10 STORE CLASS NUMBER IN REQUEST WORD B
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 (B, 9 OR 16)
28 THEN
29 RETRIEVE AND STORE DATA IN ADDRESS CONTAINED IN TABLE
30 ENDIF
31 ENDDO
32 CLEAR POINTER AND LOCAL CLASS NUMBER
33 ELSE
34 OUTPUT FAILURE MESSAGE (XP11)
35 IF REQUEST TO RECEIVER REQUEST IN BUFFER
36 THEN
37 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
38 FREE CLASS I/O NUMBER AND SAM BUFFER
39 THEN
40 ENDIF
41 ENDIF
42 ENDDO
43 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
44 ENDIF
45 ENDIF
46 END XPREG
47 1 END XPREG
SUBROUTINE FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE

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

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

INPUTS FROM CALLING SEQUENCE:

1. LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
2. PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
3. PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
4. TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE
   a. 0 - FREE
   b. 1 - INTEGER
   c. 2 - REAL
   d. 3 - DOUBLE PRECISION
   e. 4 - CHARACTER STRING LENGTH 2
   f. 5 - CHARACTER STRING LENGTH 6
   g. 6 - CHARACTER STRING LENGTH 10
   h. 7 - CHARACTER STRING LENGTH 36
   i. 8 - CHARACTER STRING LENGTH 72
5. DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN
   THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE
   NUMBER OF WORDS.
6. IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA
   AREA. IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0
   IF IT IS A VECTOR.

OUTPUTS FROM CALLING SEQUENCE:

1. DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE)
   AREA TO CONTAIN THE USER'S RESPONSE
2. RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO
   CALLER:
   a. 0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
   b. 1 - USER ENTERED 1 CONTENTS OF BUFFER UNPREDICTABLE.
   c. 2 - USER ENTERED A CR. THERE IS NO RESPONSE.
   d. 3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
   e. 4 - PARAMETER LIST IS INVALID. EITHER:
       a. IDIM LESS THAN ZERO
       b. DATLEN LESS THAN ONE
       c. TYPE VALID TYPE SPECIFIED.
DATPTR — INDEX IN WORDS INTO DATA ARRAY WHERE NEXT ELEMENT IS TO BE STORED

EFFTP — THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER STRING RESPONDER. IF TYPE IS FREE, IT IS THE CLOSEST SUPPORTED LENGTH, ELSE IT IS THE LENGTH SPECIFIED BY TYPE.

IND — CONTAINS THE INDEX INTO MSGS ARRAY OF THE CURRENT ERROR MESSAGE.

LEN — LENGTH OF CURRENT ERROR MESSAGE.

LENGTH — 9 WORD ARRAY REPRESENTING THE LENGTH IN WORDS OF THE 9 DATA TYPES 0-8 RESPECTIVELY.

STKIND — COUNT OF NUMBER OF NESTED REPEATS AND INDEX TO THE CURRENT TOP OF THE STACK.

STKREP — 4 WORD ARRAY FOR STACK OF REPEAT COUNTERS.

STKTOE — 4 WORD ARRAY FOR STACK OF TOKEN POINTERS WHERE EACH REPEAT GROUP BEGINS.

TOKPTR — POINTER TO CURRENT TOKEN BEING PROCESSED.

TYPE — INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE.

SUBROUTINES AND FUNCTIONS CALLED:

EXEC, XPRDS, XPRMS

POL ROUTINES INCLUDED:

XPRDS, STRING, SUBSCR, REPET, ERMSG
1 BEGIN XPDRS
2    IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3        CALL XPDRS TO PROMPT USER AND RECEIVE RESPONSE
4        IF XPDRS RETURN CODE IS ZERO THEN
5            DO UNTIL EOS TOKEN IS FOUND
6    SET MESSAGE NUMBER TO XP02
7        CASE TOKEN (:NUMBER,:NUMBER,:NUMBER,:STRING,:NULL,:
8            :SUBSCR,:REPET):
9          :NUMBER:
10         ERREXIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERRMSG
11        SET MESSAGE NUMBER TO XP02
12         ERREXIT IF THERE IS NO ROOM IN DATA AREA PERFORM ERRMSG
13        SET PREVIOUS TOKEN TO DATA
14        CALL XPDRS TO MOVE DATA INTO DATA AREA
15        INCREMENT POINTER IN DATA AREA
16        INCREMENT POINTER TO NEXT TOKEN
17          :STRING:
18          PERFORM STRING
19          :NULL:
20         SET MESSAGE NUMBER TO XP02
21         ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERRMSG
22        SET PREVIOUS TOKEN TO DATA
23        INCREMENT POINTER IN DATA AREA
24        INCREMENT POINTER TO NEXT TOKEN
25          :SUBSCR:
26          PERFORM SUBSCR
27          :REPET:
28          PERFORM REPET
29          ENDCASE
30    DO UNTIL TOKEN IS NOT A CLOSED PARENTHESIS
31        IF STACK IS NOT EMPTY AND
32            (PREVIOUS TOKEN IS DATA AND TOP OF STACK PAREN FLAG =0) OR
33            (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG =1) THEN
34            DECREMENT TOP OF STACK REPEAT COUNT BY 1
35            IF TOP OF STACK REPEAT COUNT > 0 THEN
36                RESET TOKEN POINTER TO TOP OF STACK INDEX
37            ELSE
38                POP TOP ENTRY ON STACK
39                IF TOKEN IS A CLOSED PAREN THEN
40                    INCREMENT TO NEXT TOKEN
41                    ENDIF
42            ENDIF
43            ENDIF
44        ENDIF
45        ENDIF
46        ENDIF
47        SET XPDRS RETURN CODE = XPDRS RETURN CODE
48        ELBS
49        SET XPDRS RETURN CODE TO SY error IN I2D PARAMETER 1ST
50        ENDIF
51        ENDIF
52    END XPDRS
1 BEGIN STRING
2 ERREXIT IF TYPE IS NOT CHARACTER OR FREE PERFORM ERREMSG
3 DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
4 ERREXIT IF TYPE OF RESPONSE > TYPE REQUESTED AND
5 IF TYPE IS NOT FREE THEN
6 SET EFFECTIVE LENGTH = LENGTH REQUESTED
7 END IF
8 SET MESSAGE NUMBER TO XPO2
9 ERREXIT IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERREMSG
10 CALL XMOV TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH
11 CALL XMOV TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH
12 SET PREVIOUS TOKEN IS DATA
13 INCREMENT POINTER IN DATA AREA
14 INCREMENT TO NEXT TOKEN
15 END STRING
16 1
17 1
18 1
19 BEGIN SUBSCR
20 INCREMENT POINTER TO NEXT TOKEN
21 SET MESSAGE NUMBER TO XPO7
22 ERREXIT IF TOKEN IS NOT AN INTEGER TO PERFORM ERREMSG
23 IF 1-DIMENSION > 1 THEN
24 SET 1 TO INTEGER VALUE
25 INCREMENT POINTER TO NEXT TOKEN
26 ERREXIT IF TOKEN IS NOT AN INTEGER OR NUMBER TO XPO3
27 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERREMSG
28 SET MESSAGE NUMBER TO XPO3
29 ERREXIT IF INTEGER > 1-DIMENSION TO PERFORM ERREMSG
30 ERREXIT IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERREMSG
31 ELSE
32 ERREXIT IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERREMSG
33 SET MESSAGE NUMBER TO XPO3
34 ERREXIT IF SUBSCRIPT IS OUT OF RANGE
35 END IF
36 ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT
37 INCREMENT POINTER BY 3 TOKENS
38 SET PREVIOUS TOKEN = SUBSCRIPT
39 SET MESSAGE NUMBER TO XPO4
40 ERREXIT IF TOKEN IS AN EBS OR
41 ERREXIT IF TOKEN IS A REPEAT OR
42 ERREXIT IF TOKEN IS A CLOSE PAREN OR
43 ERREXIT IF TOKEN IS A SUBSCRIPT TO PERFORM ERREMSG
44 END SUBSCR
45 1
FORTAN CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE

CALL XPRDM (LU, PRMLM, PROMPT, COMLEN, COMBUF, RETC)

WRITE "PRMLM" WORDS OF "PROMPT" TO USER READS THE USER'S
RESPONSE, CONVERTS RESPONSE TO TOKENS IN "COMBU" AND PASSES
BACK A RETURN CODE "RETC" INDICATING THE SUCCESS OF XPRDM
AND THE USEFULNESS OF "COMBU".

INPUTS FROM CALLING SEQUENCE:

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

OUTPUTS FROM CALLING SEQUENCE:

COMBUF - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE
ENCODED USER RESPONSE
RETC - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO THE
CALLING PROGRAM AS FOLLOWS:
0 - NORMAL RETURN BUFFER CONTAINS USERS RESPONSE
1 - USER ENTERED $. 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.
DBLMT - 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
T - LAST CHARACTER WAS A COMMA
FLGCOM - CONTINUE FLAG
O - THIS IS NOT A RESPONSE TO A CONTINUE
T - THIS IS A CONTINUE RESPONSE
FLENUM - NUMBER FLAG
O - POSITIVE NUMBER
T - NEGATIVE NUMBER
FLGPW - POWER FLAG
O - POSITIVE POWER
1 - NEGATIVE POWER

FLAGTYPE - TYPE OF REAL NUMBER
0 - SINGLE PRECISION
1 - DOUBLE PRECISION

SUBROUTINES AND FUNCTIONS USED:
EXEC, CVT, XMNOV, XRNC, XRUPK, OVF

PDL ROUTINES USED:
XPRM, TOKENS, QUOTE, DIGIT, DCOL, DECP, EORD
INTEGR, REAL, DBL, INVAL, COMPUL
1 BEGIN XPRDM
2 IF PROMPT IS NOT TOO LONG THEN
3 CALL XMOV "$" MOVE PROMPT INTO OUTPUT AREA
4 CALL EXEC TO WRITE PROMPT
5 SET XPRDM RETURN CODE TO NORMAL RETURN
6 SET CONTINUE FLAG OFF
7 INITIALIZE COMMUNICATIONS BUFFER
8 :COMLOP:
9 CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
10 CALL EXEC TO READ RESPONSE
11 CALL XUPK TO CONVERT A2 RESPONSE TO R1 FORMAT
12 IF NUMBER OF WORDS READ IS NOT ZERO THEN
13 SET COMMA FLAG ON
14 DO WHILE CONTINUE FLAG IS OFF
15 IF INPUT BUFFER IS COMPLETELY SCANNED THEN
16 SET CONTINUE FLAG ON
17 IF COMMA FLAG IS ON THEN
18 CALL EXEC TO WRITE CONTINUE
19 GO TO :COMLOP:
20 ELSE
21 IF COMMA IS FULL PERFORM COMFULL
22 IF INPUT CHARACTER IS A COMMA THEN
23 ERREXIT IF COMBUF IS NOT HOLD PERFORM COMFUL
24 STORE NULL FIELD TOKEN IN COMBUF
25 INCREMENT #WORDS IN COMBUF BY 1
26 INCREMENT #TOKENS IN COMBUF BY 1
27 ENDIF
28 SET COMMA FLAG ON
29 IF NEXT INPUT CHARACTER THEN
30 SET COMMA FLAG OFF
31 PERFORM TOKENS
32 ENDIF
33 ENDIF
34 ELSE
35 IF CONTINUE FLAG IS OFF THEN
36 SET XPRDM RETURN CODE TO SAY USER ENTERED CR
37 ENDIF
38 ENDIF
39 ELSE
40 SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
41 ENDIF
42 END XPRDM
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1019 1 BEGIN DIGIT
1020 2 PERFORM O/COL IF INPUT CHARACTER IS NOT EXHAUSTED THEN
1021 3 PERFORM DIGIT
1022 4 ELSE PERFORM DEPOP
1023 5 IF INPUT CHARACTER IS A "E" OR A "D" THEN
1024 6 IF INPUT CHARACTER IS A "E" THEN
1025 7 EXIT IF INPUT CHARACTER IS A "D" PERFORM INVERSE FOR THIS TOKEN PERFORM CONVOL
1026 8 ELSE IF INPUT CHARACTER IS "E" THEN
1027 9 PRINT ERROR MESSAGE IN GROUP OF 2
1028 10 ENDIF
1029 11 ELSE PERFORM INTEGER
1030 12 ELSE PERFORM INTEGER
1031 13 ENDIF
1032 14 ENDIF
1033 15 END DIGIT
1034 16 ENDIF
1035 17 ELSE PERFORM INTEGER
1036 18 ELSE PERFORM INTEGER
1037 19 ENDIF
1038 20 ENDIF
1039 21 END DIGIT
1040 22 END IF
1041 23 BEGIN DIGIT
1042 24 SET INTEGER = 0
1043 25 DO WHILE INPUT BUFFER IS NOT EXHAUSTED
1044 26 IF INPUT CHARACTER IS A DIGIT AND
1045 27 SET INTEGER = INTEGER + INPUT CHARACTER
1046 28 ELSE IF INPUT CHARACTER IS NOT A DIGIT AND
1047 29 SET INTEGER = INTEGER + 3
1048 30 IF INTEGER > 63 THEN
1049 31 RESET INTEGER TO 0
1050 32 END IF
1051 33 END DO
1052 34 END DIGIT
1053 35 ENDIF
1054 36 END O/COL
1055 37 END
1056 38 END
1057 39 END
1058 40 REPRODUCIBILITY OF THE
1059 41 ORIGINAL PAGE IS POOR
1060 42 5-217
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 EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067 3 ENDF
1068 3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069 4 PERFORM EORD
1070 3 ELSE
1071 4 PERFORM REAL
1072 3 ENDF
1073 2 ELSE
1074 3 PERFORM REAL
1075 2 ENDF
1076 1 END DECPT
1078 1 BEGIN EORD
1079 2 IF INPUT CHARACTER IS AN "E" THEN
1080 3 SET TYPE FLAG TO "E"
1081 4 ELSE
1082 5 SET TYPE FLAG TO "D"
1083 6 ENDIF
1084 7 GET NEXT CHARACTER
1085 8 ERREXIT IF INPUT BUFFER IS EXHAUSTED PERFORM INVAL
1086 9 IF INPUT CHARACTER IS A - THEN
1087 10 SET NEGATIVE POWER FLAG ON
1088 11 GET NEXT CHARACTER
1089 12 ELSE
1090 13 IF INPUT CHARACTER IS A + THEN
1091 14 SET NEGATIVE POWER FLAG OFF
1092 15 GET NEXT CHARACTER
1093 16 ENDIF
1094 17 ENDIF
1095 18 ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
1096 19 ERREXIT IF INPUT CHARACTER IS NOT A DIGIT PERFORM INVAL
1097 20 PERFORM DCOL
1098 21 IF NEGATIVE POWER FLAG IS ON THEN
1099 22 SET POWER = -POWER
1100 23 ENDIF
1101 24 IF TYPE FLAG IS "E" THEN
1102 25 PERFORM REAL
1103 26 ELSE
1104 27 BEGIN DBL
1105 28 ERREXIT IF NEXT TOKEN IS NOT & COMMA AND
1106 29 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
1107 30 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
1108 31 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1109 32 SET DOUBLE = DOUBLE * 10 ** POWER
1110 33 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1111 34 IF NEGATIVE POWER FLAG IS ON THEN
1112 35 SET DOUBLE = -DOUBLE
1113 36 ENDIF
1114 37 STORE DOUBLE TOKEN IN COMBUF
1115 38 INCREMENT #WORDS IN COMBUF BY 4
1116 39 INCREMENT #TOKENS IN COMBUF BY 1
1117 40 END DBL
1118 41 ENDIF
1119 42 1 END EORD
1 BEGIN INTEGER
2 ERROR: IF NEXT TOKEN IS NOT A COMMA AND
3 ERROR: IF NEXT TOKEN IS NOT A CLOSING PAREN AND
4 ERROR: IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
5 ERROR: IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
6 CONVERT DOUBLE TO INTEGER
7 ERROR: IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
8 IF NEGATIVE NUMBER FLAG IS ON THEN
9 SET INTEGER = -INTEGER
10 ENDIF
11 STORE INTEGER TOKEN IN COMBUF
12 INCREMENT #WORDS IN COMBUF BY 2
13 INCREMENT #TOKENS IN COMBUF BY 1
14 END INTEGER
15 *
16 *
17 BEGIN REAL
18 ERROR: IF NEXT TOKEN IS NOT A COMMA AND
19 ERROR: IF NEXT TOKEN IS NOT A CLOSING PAREN AND
20 ERROR: IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
21 ERROR: IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
22 SET REAL = DOUBLE = 10 ** POWER
23 ERROR: IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
24 IF NEGATIVE NUMBER FLAG IS ON THEN
25 SET REAL = -REAL
26 ENDIF
27 STORE TOKEN IN COMBUF
28 INCREMENT #WORDS IN COMBUF BY 3
29 INCREMENT #TOKENS IN COMBUF BY 1
30 END REAL
1153 1 BEGIN INVAL
1154 2 CALL XCVT TO CONVERT OCTAL CHARACTER NUMBER TO ASCII
1155 2 CALL EXEC TO WRITE ERROR MESSAGE
1156 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1157 1 END INVAL
1158
1159
1160 1 *
1161 1 BEGIN CMOVF
1162 2 CALL EXEC TO WRITE ERROR MESSAGE
1163 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1164 1 END CMOVF
1165
1166 1 *
1167 1 *
1168 1 BEGIN OVFLOW
1169 2 CALL XCVT TO CONVERT OCTAL TO ASCII
1170 2 CALL EXEC TO WRITE ERROR MESSAGE
1171 2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN
1172 1 END OVFLOW
FORTRAN CALLING PROCEDURE

CALL XPXIT (LU, RPARGS)

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

INPUT

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

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

OUTPUT

RPARGS IS RETURNED TO THE FATHER TASK

NOTES

USES EXEC, PRTN

XPXIT DOES NOT RETURN TO THE CALLER.

BEGIN XPXIT

IF LU IS NON-ZERO THEN

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

ENDIF

RETURN PARAMETERS TO FATHER

TERMINATE PROGRAM

END XPXIT
FORTRAN CALLING PROCEDURE

CALL XRBIT (BIT, BITHUM, STRING)

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

#01

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

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

#02

INPUT

#02

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

#02

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

#02

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

#02

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

OUTPUT

#03

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

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

NOTES

#05

USES .ENTR, XRSET

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

BEGIN XRBIT

TRANSFER CALLING SEQUENCE

COMPUTE ADDRESS OF WORD CONTAINING BITHUM

COMPUTE BIT NUMBER WITHIN WORD

CALL XRSET TO SET/CLEAR BIT

END XRBIT
INTEGER FUNCTION

100  XRCPR(LENGTH, ARRAY1, ARRAY2)

301 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)

501rette------

901 INPUT

100  LENGTH - POSITIVE INTEGER NUMBER OF WORDS TO BE COMPARED

120  ARRAY1 - ARRAY OF ENTERS OR CHARACTERS TO BE COMPARED TO ARRAY2

130  ARRAY2 - ARRAY OF ENTERS OR CHARACTERS TO BE COMPARED TO ARRAY1

801 print------

901 FUNCTION VALUE - 0, ARRAYS EQUAL

100  -1, ARRAYS NOT EQUAL, ARRAY1 < ARRAY2

110  +1, ARRAYS NOT EQUAL, ARRAY1 > ARRAY2

401 RETURN

501 BEGIN XRCPR

701 RETURN RESULT FLAG

1 END XRCPR
FORTRAN CALLING PROCEDURE.

CALL XR018 (DOUBLE, ASCII)

CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN
IP018.YY FORMAT

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

ASCII - NINE ASCII CHARACTER STRING REPRESENTATION OF
'DOUBLE'

LOCAL

- WORKING LOCATION FOR ABSOLUTE VALUE OR 'DOUBLE'
- REPEATEDLY MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

USES DOUBLE, FLOAT, IA4D, IDINT, IOR, KCVT, XREXT

BEGIN XR018

SET SIGN FIELD
MOVE ABSOLUTE VALUE OF 'DOUBLE' INTERNAL
COMPUTE EXPONENT
SET SIGN AND VALUE OF EXPONENT FIELD
REDUCE VALUE TO RANGE OF 1 < VALUE < 10
EXTRACT FIRST DIGIT, MERGE WITH SIGN AND STORE FIELD
EXTRACT SECOND DIGIT, MERGE WITH DECIMAL AND STORE FIELD
DO FOR NEXT FIVE PAIRS OF DIGITS
MULTIPLY BY 100 TO EXTRACT PAIR
EXTRACT DIGITS AND STORE FIELD
END XR018
1   CD0        FORTRAN CALLING PROCEDURE
2   CD3
3   CD0        CALL XREQ
4   CD0
5   **********
6   CD1        MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE
7   CD1
8   **********
9   CD1        INPUT
10  CD2        COMMON XE - CLASNO, FLAGS, LU, REQBUF, REQPTR
11  CD2        ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER
12  CD2
13  **********
14  CD3        OUTPUT
15  CD3        COMMON XE - REQPTR
16  CD3        CLASS I/O WRITE/READ TO CLASS 'CLASNO'
17  CD5        REQBUF AND RESPONSE IS PRINTED IF REQUESTED
18  CD3
19  **********
20  CD5        NOTES
21  CD5        COMMUNICATES WITH FDS MANAGER FATHER TASK
22  CD5        USES EXEC, IAND, XRMOV, XRMSG, XR6G, XRSL, XDPL, XVPW
23  CD5
24  **********
25  *           *
26  *           *
27  *           *
28  *           *
29  *           *
30  *           *
31  BEGIN XREQ
32      BEGIN TRACE
33      OUTPUT TRACES TO MANAGER
34      REQUEST AWA MANAGEMENT AND WAIT FOR RESPONSE
35      RETRIEVE MANAGER RESPONSE
36      BEGIN TRACE
37      RETURN RESPONSE IN REQPTR
38  END XREQ
39  BEGIN TRACE
40      IF TRACE REQUESTED
41      THEN
42      DO FOR EACH REQUEST
43      CALL XDPL TO FORMAT LINE
44      OUTPUT LINE
45      ENDDO
46      OUTPUT PARM1 AND PARM2
47      ENDDO
48  END TRACE
INTEGER FUNCTION
XREV(START, LENGTH, SOURCE)
********
** EXTRACT 'LENGTH' BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
** AND RIGHT ADJUST
********
** START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
** (SIGN BIT = 0)
** LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
** SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED
********
** NOTES
USES .ENTR
**
BEGIN XREV
TRANSFER CALLING SEQUENCE
IF START NOT = 0
THEN
CONSTRUCT SHIFT
LOAD A WITH SOURCE
SHIFT BA LEFT START BITS
ELSE
LOAD A WITH SOURCE
ENDIF
SAVE A
CLEAR B
CONSTRUCT SHIFT
RESTORE A
SHIFT BA LEFT LENGTH BITS
MOVE RESULT FROM B TO A
END XREV
FORTRAN CALLING PROCEDURE.

CALL XRE14 (REAL, ASCII)

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

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

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

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

NOTES USES FLOAT, IAND, IIFIX, IOR, KCVT, XREST, XRSFL

BEGIN XRE14

SET SIGN FIELD
MOVE ABSOLUTE VALUE OF 'REAL' INTERNAL
COMPUTE EXPONENT
SET SIGN AND VALUE OF EXPONENT FIELD
REDUCE VALUE TO RANGE OF 1 <= VALUE < 10
EXTRACT FIRST DIGIT AND STORE FIELD
SET DECIMAL FIELD
DO FOR NEXT THREE PAIRS OF DIGITS
MULTIPLY BY 100 TO EXTRACT PAIR
EXTRACT DIGITS AND STORE FIELD

END XRE14
FORTRAN CALLING PROCEDURE

CALL XR16 (INTGR, ASCII)

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

INPUT
INTGR - SIXTEEN BIT INTEGER TO BE CONVERTED

OUTPUT
THREE WORD CHARACTER STRING REPRESENTATION OF 'INTGR'

LOCAL
I - INTERNAL LOCATION FOR 'INTGR' REPEATEDLY MODIFIED TO

PRODUCE 'ASCII'

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

NOTES
USES XRMV AND XRCK

BEGIN XR16
BLANK WORKING SPACE

CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING

SET SIGN OF 'INTGR' IN 'ASCII'

CALL XRCK TO CONVERT FROM R1 TO A2 FORMAT

END XR16
302 1 *00 FORTRAN CALLING PROCEDURES
303 1 *00
304 1 *00 CALL XRLCK (RCODE)
305 1 *00 CALL XRULK (RCODE)
306 1 *00
307 1 ********
308 1 *01
309 1 *01 XRLCK AND XRULK PROVIDE A MECHANISM FOR SERIALIZING THE UPDATE
310 1 *01 OF FDS GLOBAL SYSTEM TABLES AND FILES. THE RESOURCE NUMBER
311 1 *01 STORED IN THE XVS TB RESIDENT STATUS TABLE IS USED AS THE
312 1 *01 LOCKING MECHANISM
313 1 *01
314 1 ********
315 1 *02 INPUT
316 1 *32 XVS TB RESOURCE NUMBER
317 1 *02
318 1 ********
319 1 *03 OUTPUT
320 1 *03 RCODE - INTEGER RETURN CODE (0 - SUCCESSFUL, 1 - FAILURE)
321 1 *03
322 1 ********
323 1 *04 LOCAL
324 1 *04 STAT - STATUS OF THIS COPY OF XEXEC USE OF XVS TB RN
325 1 *04 1 - RN LOCKED
326 1 *04 4 - RN UNLOCKED
327 1 *04
328 1 ********
329 1 *05 NOTES
330 1 *05 USES .ENTR, RNR.
331 1 *05
332 1 *05 THIS ROUTINE MAY NOT BE OVERLayed
333 1 *05
334 1 ********
1 BEGIN XLCK
2 SET REQUEST FOR LOCK FUNCTION
3 PERFORM RLOCK
4 END XLCK
5 BEGIN XRULK
6 SET REQUEST FOR UNLOCK FUNCTION
7 PERFORM RLOCK
8 END XRULK
9 BEGIN RLOCK
10 IF REQUEST IS CONSISTENT WITH STATUS
11 THEN
12 SET NEW STATUS
13 IF RM IN STB IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBK
14 THEN
15 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
16 ENDIF
17 CLEAR RETURN CODE
18 ELSE
19 SET RETURN CODE
20 ENDIF
21 END RLOCK
22 END XLCK
INTEGER FUNCTION

360 1 *00 XRLOC(A)

361 1 *00

362 1 *00

363 1 *00

364 1 *01

365 1 *01

366 1 *01

367 1 *01

368 1 *02

369 1 *02 A - VARIABLE, ROUTINE, ETC. FOR WHICH THE ADDRESS IS DESIRED

370 1 *02

371 1 *03

372 1 *03

373 1 *03 XRLOC - 16-BIT ADDRESS OF A

374 1 *03

375 1 *03

376 1 *05

377 1 *05 NO EXTERNAL REFERENCES

378 1 *05

379 1 *05

380 1 *

381 1 *

382 1 *

383 1 *

384 1 BEGIN XRLOC

385 2 TRANSFER CALLING SEQUENCE

386 2 LOAD THE ADDRESS OF THE CALLING PARAMETER

387 1 END XRLOC
1 *NO FORTRAN CALLING PROCEDURE
390 1 *NO
391 1 *NO CALL XRMOV (LENGTH, SOURCE, OBJECT)
392 1 *NO
393 1 *********
394 1 *D1 MOVE 'LENGTH' WORDS FROM 'SOURCE' TO 'OBJECT'
395 1 *D1
396 1 *********
397 1 *********
398 1 *D2 INPUT
399 1 *D2  LENGTH - POSITIVE INTEGER INDICATING NUMBER OF WORDS TO MOV.
400 1 *D2  SOURCE - ARRAY OF WORDS TO BE MOVED
401 1 *D2
402 1 *********
403 1 *D3 OUTPUT
404 1 *D3  OBJECT - ARRAY RECEIVING MOVED WORDS
405 1 *D3
406 1 *********
407 1 *D5 NOTES
408 1 *D5  USES .ENTR
409 1 *D5
410 1 *********
411 1 *
412 1 *
413 1 *
414 1 *
415 1 BEGIN XRMOV
416 1 END XRMOV
417 1 TRANSFER CALLING SEQUENCE
418 1 INITIALIZE MOVE
419 1 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
FORTRAN CALLING PROCEDURE

CALL XMSG (NUMBER, LOCATE, LENGTH, SOURCE)

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

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

TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL

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

A - AREA INDICATOR AS FOLLOWS

1 - AS
2 - XD
3 - XE
4 - XI
5 - XS
6 - YT
7 - XX
8 - XL
9 - DF
10 - SC

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

WORDS OF 'SOURCE' TO BE OUTPUT

LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED

'SOURCE' (NOT USED IF 'MM' OF 'NUMBER' IS ZERO)

LENGTH - INTEGER NUMBER OF WORDS OF 'SOURCE' TO BE INSERTED INTO

MESSAGE. ZERO INDICATES NO INSERTION

SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED

IF 'LENGTH' IS ZERO)

LU - USERS LOGICAL UNIT NUMBER

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

COMMON

OUTPUT

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

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

NOTES

USES FD SYSTEM MESSAGE FILE XMSG

USES CLOSE, EXEC, IAMS, KCUT, OPEN, READ, XMOV, XUDOG

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

XSTRIP(Bit, BitNum, String)

!!! SEARCH BIT STRING "STRING" BEGINNING AT BIT NUMBER "BITNUM" FOR THE NEXT OCCUREANCE OF BIT SETTING "BIT"

***********

** INPUT **

** BIT ** - INTEGER VALUE THE LAST BIT OF WHICH IS TO BE COMPARED TO BIT NUMBERS OF "STRING" FOR A MATCH

** BITNUM ** - UN-SIGNED 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 OCCUREANCE OF "BIT" >= "BITNUM"

***********

** NOTES **

USES .ENTR

** CD5 ** SEARCH WILL NOT TERMINATE UNTIL A VALUE OF "BIT" IS DETECTED OR ALL OF MEMORY HAS BEEN EXAMINED. THIS APPROPRIATE STEPS SHOULD BE TAKEN TO FORCE A MATCH AT THE END OF THE STRING.

** CD5 ** A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4K96 WORDS) CAN BE MEANINGFULLY ACCOMMODATED.

***********
BEGIN XRRKB
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. ENDDO
15. CLEAR BIT COUNT
16. ENDIF
17. ENDDO
18. RETURN VALUE OF MATCHING BIT NUMBER
19. END XRRKB
542 1 *00 FORTRAN CALLING PROCEDURE
543 1 *00 CALL X06 (BINARY, OCTAL)
544 1 *00 *00
545 1 *00 *00
546 1 *00 *00
547 1 *00 *00
548 1 *00 CONVERT A WORD FROM BINARY TO SIX CHARACTER OCTAL REPRESENTATION
549 1 *00
550 1 *00
551 1 *00
552 1 *00
553 1 *00
554 1 *00
555 1 *00
556 1 *00
557 1 *00
558 1 *00
559 1 *00
560 1 *00
561 1 *00
562 1 *00
563 1 *00
564 1 *00
565 1 *00
566 1 *00
567 1 *00
568 1 *00
569 1 *00
570 1 *00
571 1 *00
572 1 *00
573 1 *00
574 1 *00
575 1 *00
576 1 *00 OCTAL - THREE WORD ARRAY CONTAINING OCTAL REPRESENTATION OF
577 1 *00 'BINARY' IN ASCII FORMAT (66)
578 1 *00
579 1 *00
580 1 *00 NOTES
581 1 *00
582 1 *00
583 1 *00
584 1 *00
585 1 *00
586 1 *00
587 1 *00
588 1 *00 BEGIN X06
589 1 *00 TRANSFER CALLING SEQUENCE
590 1 *00 FORM 18-BIT WORD USING B AND 2 MOST SIGNIFICANT BITS OF A
591 1 *00 SET BYTE FLAG HIGH
592 1 *00 DO FOR EACH PAIR OF OCTAL DIGITS (3)
593 1 *00 SET PREFIX BITS (0000000)
594 1 *00 SHIFT IN DIGIT
595 1 *00 IF BYTE FLAG SET HIGH
596 1 *00 THEN
597 1 *00 SNIFT FOR ACCOMODATION OF LOW BYTE
598 1 *00 ELSE
599 1 *00 STORE PAIR OF DIGITS IN OCTAL(1)
600 1 *00 CLEAR FOR NEXT PAIR OF DIGITS
601 1 *00 ENDIF
602 1 *00 FLIP BYTE FLAG
603 1 *00 END X06
604 1 *00
FORTRAN CALLING PROCEDURE

CALL XRPACK (LENGTH, UNPKED, PACKED)

### INPUT
1. LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED
2. UNPKED - ARRAY OF CHARACTERS IN R1 FORMAT

### OUTPUT
3. PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS ODD, THE LAST WORD WILL BE BLANK FILLED.

### NOTES
- USES .ENTR

BEGIN XRPACK
TRANSFER CALLING SEQUENCE
SET BYTE FLAG FOR HIGH BYTE
INITIALIZE PACKED POINTER
DO FOR EACH CHARACTER IN UNPKED
IF BYTE FLAG SET HIGH THEN
LOAD A WITH NEXT WORD OF UNPKED
SHIFT CHARACTER INTO HIGH BYTE
ELSE
INCCLUSIVE OR NEXT WORD OF UNPKED INTO A
STORE A IN PACKED
INCREMENT POINTER
ENDIF
FLIP BYTE FLAG
ENDDO
IF BYTE FLAG SET LOW THEN
INCCLUSIVE OR BLANK INTO LOW BYTE
ENDIF
STORE A IN PACKED
END XRPACK
FORTRAN CALLING PROCEDURE

1 100 CALL XRFIN (PREFIX, NAME4, NAME6)

XRFIN BUILDS A QUALIFIED FILE NAME OF UPTO SIX CHARACTERS IN
LENGTH BY PREFIXING THE INPUT ONE TO FOUR CHARACTER NAME WITH
THE PREFIX CHARACTER AND APPENDING A USER QUALIFIER CODE TO THE
INPUT.

1 101 IN

1 102 PREFIX - FILE TYPE PREFIX STORED IN R1 FORMAT
1 103 NAME4 - ONE TO FOUR CHARACTER PACKED NAME TO BE QUALIFIED
1 104 COMMON XE - QUAL

XRFIN

1 105 OUTPUT

1 106 NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME

XRFIN

1 107 NOTES

XRFIN

1 108 USES .ENTR

XRFIN

1 109

XRFIN

1 110

XRFIN

1 111

XRFIN

1 112 BEGIN XRFIN

XRFIN

2 668 STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING
2 669 MOVE NAME4 INTO NEXT FOUR POSITIONS
2 670 STORE BLANK IN SIXTH POSITION
2 671 LOCATE FIRST BLANK CHARACTER
2 672 REPLACE BLANK WITH USER ID CHARACTER (QUAL)
2 673 MOVE QUALIFIED NAME TO NAME6
2 674 END XRFIN

XRFIN
FORTRAN CALLING PROCEDURE

CALL XRSET (START, LENGTH, SOURCE, OBJECT)

*********

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

*********

INPUT

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

LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED

SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED

*********

OUTPUT

OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED

*********

NOTES

USES .ENTR

*********

BEGIN XRSET

TRANSFER CALLING SEQUENCE

CONSTRUCT SHIFT INSTRUCTIONS

SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER

SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD

CONSTRUCT MASK AND CLEAR FIELD OF OBJECT

INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT

END XRSET
INTEGER FUNCTIONS

*0
*1 XRSFL(COUNT, SOURCE)
*2 XRSFR(COUNT, SOURCE)
*3
*4 1 #D1
*5 1 #D1
*6 1 #D1
*7 1 #D2
*8 1 #D2
*9 1 #D2
*10 1 #D5
*11 1 #D5
*12
*13 1 BEGIN XRSFL
*14 2 SET FOR LEFT SHIFT
*15 2 PERFORM SHIFT (FLAG, COUNT, SOURCE)
*16 2 END XRSFL
*17 1 BEGIN XRSFR
*18 2 SET FOR RIGHT SHIFT
*19 2 PERFORM SHIFT (FLAG, COUNT, SOURCE)
*20 2 END XRSFR
*21 1 BEGIN SHIFT
*22 2 TRANSFER CALLING SEQUENCE
*23 2 CONSTRUCT SHIFT INSTRUCTION
*24 2 LOAD A WITH SOURCE
*25 2 CLEAR B
*26 2 SHIFT BA AS SPECIFIED
*27 1 END SHIFT
FORTRAN CALLING PROCEDURE

CALL XRUNG (NAME6, NAME4)

XRUNG 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

NOTES

USES .ENTR

LOCATE LAST NON-BLANK CHARACTER OF NAME4

BLANK THAT CHARACTER

END IF

END XRUNG
**FORTRAN CALLING PROCEDURE**

1  00  CALL XRUPK (LENGTH, PACKED, UNPKED, COUNT)

1  00  **********

1  01  CONVERT 'LENGTH' WORDS OF 'PACKED' FROM A2 FORMAT TO R1 FORMAT,

1  01  REMOVING BLANKS NOT DELIMITED BY QUOTE MARKS, AND RETURN IN

1  01  'UNPKED' WITH THE NON-BLANK CHARACTER COUNT IS 'COUNT'.

1  01  **********

1  02  INPUT

1  02  LENGTH - POSITIVE INTEGER NUMBER OF WORDS IN PACKED

1  02  PACKED - ARRAY OF CHARACTER DATA IN A2 FORMAT

1  02  **********

1  03  OUTPUT

1  03  UNPKED - ARRAY OF NON-BLANK CHARACTERS IN R1 FORMAT

1  03  COUNT - NUMBER OF CHARACTERS IN UNPKED

1  03  **********

1  05  NOTES

1  05  USES .ENTR

1  05  IF PACKED AND UNPKED ARE THE SAME ADDRESS SPACE UNPKED MAY OVERLAY

1  05  PACKED

1  05  **********

1  05  ** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER

1  06  STRINGS.
839 1 BEGIN XRUPK
840 2 TRANSFER CALLING SEQUENCE
841 2 TURN ON BLANK REMOVAL
842 2 INITIALIZE COUNT
843 2 DO FOR EACH WORD OF PACKED
844 3 LOAD A WITH NEXT WORD
845 3 ROTATE A 8 BITS
846 3 DO FOR EACH BYTE OF WORD
847 4 AND OFF HIGH BYTE
848 4 IF A = QUOTE MARK
849 4 THEN
850 5 CHANGE BLANK REMOVAL OPTION
851 4 ENDF
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 ENDF
859 4 ELSE
860 5 INCREMENT COUNT
861 5 STORE A IN UNPACKED
862 4 ENDF
863 4 RELOAD A WITH WORD
864 3 ENDDO
865 2 ENDDO
866 2 RETURN VALUE OF COUNT
867 1 END XRUPK
FORTRAN CALLING PROCEDURE

CALL XRISP (CHSTR, LNGTH)

**********

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

**********

INPUT

CHSTR - CHARACTER STRING OF LNGTH WORDS IN A2 FORMAT

LENGTH - LENGTH OF CHSTR IN WORDS

**********

OUTPUT

CHSTR - CHARACTER STRING OF LNGTH WORDS IN A2 FORMAT WITH:
DUPLICATE BLANKS REMOVED

LENGTH - NEW LENGTH OF CHSTR IN WORDS

**********

EXTERNAL REFERENCES

ENTR

**********

SPECIAL REMARKS

THIS ROUTINE ASSUMES:
1. THE INPUT CHARACTER STRING HAS A POSITIVE LENGTH > 0
2. SPECIAL HANDLING OF STRINGS WITHIN QUOTATION MARKS DOES NOT
BEGIN UNTIL THE FIRST OCCURRENCE OF QUOTATION MARKS ON EACH
ENTRY
3. THE FIRST/LAST CHARACTER IN A CHARACTER STRING HAS THE
HIGH ORDER BIT OF THE QUOTE CHARACTER 'ON' TO INDICATE
BEGIN/END OF A CHARACTER STRING. THIS BIT IS TURNED 'OFF'
BY THIS PROGRAM PRIOR TO EXIT. THIS CHANGE MADE TO
INCORPORATE QUOTE MARKS WITHIN STRING.

OTHER PROCESSORS CHANGED FOR THIS MODIFICATION WERE:
A. XILAN, XPAR
B. XILSS/XILSS

**********
1 BEGIN XR15P
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'S END:
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 PRECEDING BLANK), THEN
14 SKIP THIS CHARACTER (EXIT TO ENDDD)
15 ELSE
16 SET BLANK FLAG
17 ENDIF
18 ELSE
19 CLEAR BLANK FLAG
20 ENDIF
21 ENDDD
22 STORE CHARACTER AT INDEXED POSITION
23 INCREMENT STORE INDEX
24 ENDDD
25 IF NUMBER OF CHARACTERS STORED'S END:
26 STORE ONE MORE BLANK
27 ENDIF
28 CONVERT STORE INDEX TO WORD COUNT AND RETURN
942 1 END XR15P
FORTRAN CALLING PROCEDURE

CALL XRLBS(XSERE)

XSERE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR

COMMON XE - LU
COMMON XB - DEBUG, IRETCD, NEWTAB, NUMENT, PRMLEN
PROMPT, PRNTND, WBUF

OUTPUT
COMMON XE - RERBUF, RERPTR

COMMON XB - NUMENT, PRMLEN, PRNTND, PROMPT, WBUF

NOTES
USES Routines
EXEC
XEREM
XREX
XRMOV
XRMSE
XMPY
XRPCX
XRPRM
XRCON
XRJOB
BEGIN XSERE
  DO UNTIL A % OR 'EXIT' IS ENTERED
  CALL XSPMN TO BUILD A PROMPT BASED ON PROMPT MODE FOR THE
  NEXT TABLE ENTRY (INDICATED BY TABINDX)
  CALL ITCOM TO ISSUE THE PROMPT AND RETURN RESPONSE
  ERREXIT IF ERROR IN ITCOM :ERR10:
  EXIT XSERE IF X WAS ENTERED
  IF NOTHING (ONLY CR) ENTERED, THEN
  IF PROMPT MODE IS NOT 'ALL', THEN
    CALL XRMSE - 'INVALID INPUT'
    ENDF
  ELSE
    CALL XSNPT TO PROCESS INPUT BASED ON PROMPT MODE,
    CURRENT TABLE ENTRY (TABINDX), AND PROMPTED SEQUENCE
    NUMBER (PRNUM)
  ENDF
  ENDDO
  BUILD ANA REQUEST TO DELETE/VERIFY ABSENCE OF ME,TAB
  CALL XSPEX TO PACK THE TABLE BUFFER (REMOVE DELETED ENTRIES)
  BUILD ANA REQUEST TO ALLOCATE NEWTAB
  IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
  BUILD ANA REQUEST TO STORE NEWTAB
  CALL XREN TO PROCESS THE REQUESTS
  IF THE ALLOCATE REQUEST FAILED, THEN
    CALL XRMSE - 'ANA/DWA FULL, SEQUENCE TABLE NOT STORED'
    CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
  ELSE
    CALL XRMSE - 'VACUOUS TABLE -- NOT STORED'
  ENDF
  ENDF
  EXIT XSERE
  ERR10: CALL XRMSE - 'SYSTEM ERROR'
  END XSERE
BEGIN XSPRM
  IF PROMPT MODE IS ALL, THEN
    IF NUMBER OF ENTRIES (NUMENT) > 0, AND
      ENTRIES EXIST BEYOND TABLE ENTRY INDEX (TABNDX), THEN
        DO UNTIL A NON-DELETED ENTRY IS FOUND
          INCREMENT TABLE ENTRY INDEX (TABNDX) TO NEXT ENTRY (+7)
        ENDU
        BUILD PROMPT OF THE FORM 'XSPRM=PROC,INT'
        SET PROMPT SEQUENCE NUMBER (SEINO) TO SEQUENCE NO. OF ENTRY
      ELSE
        SET PROMPT NO. TO CREATE
      ENDIF
    ENDIF
  ENDIF
  IF PROMPT MODE IS CREATE, THEN
    SET TABLE ENTRY INDEX (TAPID) TO NEXT ENTRY (NUMENT * 7 + 1)
    IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
      IF SEQUENCE NO. OF LAST ENTRY > 32690, THEN
        CALL XPRNG - 'UNABLE TO BUILD SEQUENCE NO. > 32700'
      ENDIF
      SET PROMPT MODE TO UPDATE
    ELSE
      SET PROMPT SEQUENCE NUMBER (SEINO) TO NEXT MULTIPLE OF 100
      BEYOND SEQUENCE NUMBER OF LAST TABLE ENTRY
    ENDIF
  ENDIF
  ELSE
    SET PROMPT SER. NO. (SEINO) TO BE 100
  ENDIF
  IF PROMPT MODE IS NOT UPDATE, THEN
    BUILD PROMPT OF THE FORM 'XSPRM=
  ENDIF
  IF PROMPT MODE IS UPDATE, THEN
    BUILD PROMPT OF THE FORM
  ENDIF
  SET PROMPT LENGTH TO 0 CAUSING # PROMPT TO BE ISSUED
ENDIF
END XSPRM
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>156</td>
<td>CALL XSNPT</td>
</tr>
<tr>
<td>157</td>
<td>XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE</td>
</tr>
<tr>
<td>158</td>
<td>TABLE EDITION</td>
</tr>
<tr>
<td>159</td>
<td>COMMON XE - COMBUF, CMPTR, LU, TOKENS</td>
</tr>
<tr>
<td>160</td>
<td>COMMON XB - DEBUG, DIRECT, NUMED, P-MTHO</td>
</tr>
<tr>
<td>161</td>
<td>COMMON XE - COMPTN</td>
</tr>
<tr>
<td>162</td>
<td>COMMON XB - INSERT, INETC, NUMED, P-MTHO, SEQNO, TABNOX, TABSTX, WBUF</td>
</tr>
<tr>
<td>163</td>
<td>XSNPT</td>
</tr>
<tr>
<td>164</td>
<td>USES ROUTINES</td>
</tr>
<tr>
<td>165</td>
<td>XSNPT</td>
</tr>
<tr>
<td>166</td>
<td>XSNPT</td>
</tr>
<tr>
<td>167</td>
<td>XSNPT</td>
</tr>
<tr>
<td>168</td>
<td>XSNPT</td>
</tr>
<tr>
<td>169</td>
<td>XSNPT</td>
</tr>
<tr>
<td>170</td>
<td>XSNPT</td>
</tr>
<tr>
<td>171</td>
<td>XSNPT</td>
</tr>
<tr>
<td>172</td>
<td>XSNPT</td>
</tr>
<tr>
<td>173</td>
<td>XSNPT</td>
</tr>
<tr>
<td>174</td>
<td>XSNPT</td>
</tr>
<tr>
<td>175</td>
<td>XSNPT</td>
</tr>
<tr>
<td>176</td>
<td>XSNPT</td>
</tr>
<tr>
<td>177</td>
<td>XSNPT</td>
</tr>
<tr>
<td>178</td>
<td>XSNPT</td>
</tr>
<tr>
<td>179</td>
<td>XSNPT</td>
</tr>
<tr>
<td>180</td>
<td>XSNPT</td>
</tr>
<tr>
<td>181</td>
<td>XSNPT</td>
</tr>
<tr>
<td>182</td>
<td>XSNPT</td>
</tr>
<tr>
<td>183</td>
<td>XSNPT</td>
</tr>
<tr>
<td>184</td>
<td>XSNPT</td>
</tr>
<tr>
<td>185</td>
<td>XSNPT</td>
</tr>
<tr>
<td>186</td>
<td>XSNPT</td>
</tr>
<tr>
<td>187</td>
<td>XSNPT</td>
</tr>
<tr>
<td>188</td>
<td>XSNPT</td>
</tr>
<tr>
<td>189</td>
<td>XSNPT</td>
</tr>
<tr>
<td>190</td>
<td>XSNPT</td>
</tr>
<tr>
<td>191</td>
<td>XSNPT</td>
</tr>
<tr>
<td>192</td>
<td>XSNPT</td>
</tr>
<tr>
<td>193</td>
<td>XSNPT</td>
</tr>
<tr>
<td>194</td>
<td>XSNPT</td>
</tr>
<tr>
<td>195</td>
<td>XSNPT</td>
</tr>
<tr>
<td>196</td>
<td>XSNPT</td>
</tr>
<tr>
<td>197</td>
<td>XSNPT</td>
</tr>
<tr>
<td>198</td>
<td>XSNPT</td>
</tr>
</tbody>
</table>

5-254
1 BEGIN XSNPT
2 IF PROMPT MODE IS UPDATE, THEN
3 IF TOKEN INPUT IS AN INTEGER, THEN
4 ERREXIT IF INTEGER < 1 :ERROR:
5 RETAIN INTEGER AS SEQUENCE NO. (SEQNO)
6 INCREMENT TO NEXT TOKEN
7 ERREXIT IF TOKEN IS NOT "#" :ERROR:
8 INCREMENT TO NEXT TOKEN
9 START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED
10 EXIT IF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SEQNO.)
11 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
12 SET INSERT FLAG TO ZERO INDICATING REPLACEMENT OF ENTRY
13 EXIT IF SEQUENCE NO. OF ENTRY > SEQUENCE NO. INPUT (SEQNO)
14 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
15 SET INSERT FLAG TO 1 INDICATING INSERT NECESSARY
16 ORELSE
17 INCREMENT TO NEXT TABLE ENTRY
18 ENDDO
19 SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE
20 SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE
21 CALL XSEMT TO "REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG"
22 ELSE
23 ERREXIT IF TOKEN IS NOT A NAME :ERROR:
24 START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED
25 EXIT IF NAME INPUT IS DIRECTIVE
26 SET INDEX TO DIRECTIVE LIST ENTRY
27 ORELSE
28 INCREMENT TO NEXT DIRECTIVE
29 ENDDO
30 ERREXIT :ERROR:
31 ENDSZARCH
32 CASE (IXLIS, IXDEL, IXSNPT, IXSNM, IXSMRG), INDEX
33 ENDIF
34 ELSE, PROMPT MODE IS NOT UPDATE
35 IF TOKEN IS #, THEN
36 SET PROMPT MODE TO UPDATE
37 ELSE
38 IF AN $ IS INPUT, THEN
39 ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR:
40 MARK THIS TABLE ENTRY AS DELETED
41 DO FROM END OF TABLE UNTIL A NONDELETED ENTRY IS FOUND
42 IF TABLE ENTRY IS MARKED FOR DELETION, THEN
43 DECREMENT NUMBER OF TABLE ENTRIES BY ONE
44 ENDF
45 ENDDO
46 ELSE
47 IF PROMPT MODE IS ALL, THEN
48 SET INSERT FLAG TO TWO TO INDICATE EXTENSION OF TABLE
49 ELSE
50 SET INSERT FLAG TO ZERO TO INDICATE REPLACEMENT OF TABLE ENTRY
51 ENDF
52 CALL XSEMT TO BUILD ENTRY BASE ON INSERT FLAG
53 ENDF
54 ENDF
55 ENDF
56 ENDF
57 EXIT XSNPT
58 :ERROR: CALL XRMG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
258  3
259  2
260  3
261  2
262  END ISMP
263
266
5-256
FORTRAN CALLING PROCEDURES

CALL XSENT

****

XSENT CONSTRUCTS SEQUENCE TABLE ENTRIES

COMMON XE - COMBUF, COMPTR, TOKENS, LU

COMMON XB - DEBUG, LIBIDZ, NUMENT, PRMTMD,
SEGNO, TABMDX, WBUF

******

OUTPUT

COMMON XE - COMPTR

COMMON XB - INTNAM, NUMENT, PRCNAM, PRMTMD,
SEGNO, TABMDX, WBUF

******

NOTES

USES ROUTINES

XRCPN
XREXC
XRESF
XRMOV
XRMSG
XRESET
XRPOK
XUE9G

*******

BEGIN XSENT

ERRSIT IF TOKEN INPUT IS NOT A NAME :ERR01:

RETAIN THIS NAME AS PROC. NAME

INCREMENT TO NEXT TOKEN

START SEARCH UNTIL ALL ENTRIES OF XLIBD SEARCHED

EXIT IF XLIBD ENTRY = PROC. NAME

ORELSE

INCREMENT TO NEXT ENTRY

END LOOP

ERRSIT :ERR03:

END SEARCH

IF COMMA IS NEXT TOKEN, THEN

EXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERR04:

INCREMENT TO NEXT TOKEN

EXIT IF NEXT TOKEN IS NOT A NAME :ERR01:

RETAIN THIS NAME AS INTERFACE TABLE NAME

INCREMENT TO NEXT TOKEN

ELSE

SET INTERFACE TABLE NAME TO ZERO
322 3 ELDIF
323 3 ERREXIT IF NEXT TOKEN IS NOT EOF; :ERROR:
324 3 IF INSERT FLAG DOES NOT INDICATE REPLACE, THEN (I.E. INSERT OR ADD)
325 4 IF NUMBER OF TABLE ENTRIES (NUMENT) = 150, THEN
326 5 CALL XSPCK TO PACK TABLE BUFFER (REMOVE DELETED ENTRIES)
327 5 IF NUMBER OF TABLE ENTRIES STILL = 150, THEN
328 6 SET PROMPT MODE TO UPDATE
329 6 ERREXIT :ERROR5;
330 5 ELDIF
331 4 ELDIF
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 ELDIF
344 5 ELDIF
345 4 ELDIF
346 4 SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SEQNO) INPUT/PROMPTED
347 3 ELDIF
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 ELDIF
352 2 EXIT XSEN T
353 3 :ERROR: CALL XRMG - 'SYNTAX ERROR - MISSING OR EXTRANE OUS FIELD'
354 3 :ERROR: CALL XRMG - '.... IS NOT A VALID PROCESSOR NAME'
355 3 :ERROR: CALL XRMG - '.... DOES NOT USE AN INTERFACE TABLE'
356 3 :ERROR: CALL XRMG - 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'
357 2 ELDXSEN T
359 2 CD0      FORTRAN CALLING PROCEDURE
360     CALL XPCK
361     CD0      TABLE ENTRIES MARKED FOR DELETION
362     CD0      CD0      XPCK COMPACTS THE WORKING BUFFER BY REMOVING ALL SEQUENCE
363     CD0      CD0      INPUT
364     CD0      CD0      COMMON XE - LU
365     CD0      CD0      COMMON XB - DEBUG, NUMENT, TABNOX, WBUF
366     CD0      CD0      CD0      CD0      CD0      CD0      NOTES
367     CD0      CD0      CD0      CD0      CD0      CD0      CD0      USES ROUTINES
368     CD0      CD0      CD0      CD0      CD0      CD0      CD0      XRMOW
369     CD0      CD0      CD0      CD0      CD0      CD0      CD0      XRMSP
370     CD0      CD0      CD0      CD0      CD0      CD0      CD0      XUDP5
371     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
372     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
373     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
374     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
375     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
376     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
377     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
378     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
379     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
380     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
381     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
382     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
383     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
384     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
385     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
386     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
387     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
388     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
389     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
390     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
391     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
392     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0
393     CD0      CD0      CD0      CD0      CD0      CD0      CD0      CD0

FURTHER DETAILS OF THE
GENERAL LAY IS PAPER
BEGIN XSPCK
IF THE TABLE IS NOT EMPTY, THEN
DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED
IF THIS ENTRY IS MARKED DELETED, THEN
SET MOVE LENGTH (MOVLEN) TO 7
DO UNTIL A NON-DELETED ENTRY IS FOUND
INCREMENT MOVLEN BY 7
ENDDO
MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO
THE DELETED ENTRY
DECREMENT NUMENT BY MOVLEN/7
IF TABLE INDEX (TABNOX) > INDEX TO DELETED ENTRY, THEN
DECREMENT TABLE INDEX (TABNOX) BY MOVLEN
ENDIF
ENDDO
ENDIF
END XSPCK
414       CD0  FORTRAN CALLING PROCEDURE
415       CD0
416       CD0
417       CD0  CALL XSLST
418       CD0
419       CD0  **********
420       CD1  CD1  XSLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
421       CD1  CD1  CONTAINED IN THE WORKING BUFFER
422       CD1
423       CD1  **********
424       CD2
425       CD2  INPUT
426       CD2
427       CD2  COMMON XE - LU, REBUF, SUBSTA
428       CD2
429       CD2
430       CD2  COMMON XB - BEGIN, DEBUG, ENDMO, LISTLU,
431       CD2
432       CD2
433       CD2  **********
434       CD5
435       CD5  NOTES
436       CD5
437       CD5  USES Routines
438       CD5
439       CD5
440       CD5  XRIG
441       CD5  XRMOV
442       CD5  XRSET
443       CD5  XUDBG
444       CD5  **********
445       BEGIN XSLST
446       IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
447       DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
448       SET LIMITS (BEGIN AND ENDMO) OF SEQ. ENTRIES LISTED
449       SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
450       ENDF
451       WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
452       IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
453       DO FROM BEGIN TO ENDMO
454       MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
455       IF INT. NAME = 0, THEN
456       SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARS.)
457       ELSE
458       SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARS.)
459       ENDIF
460       CALL XRIG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
461       CALL XRDBG TO PRINT BUFFER
462       ENDF
463       ENDS
464       ENDM
BEGIN XSCAN
SET LIST LIMITS (BEGIN AND ENDMO) 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 :ERROR6:
SET BEGIN LIMIT (BEGIN) 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 :ERROR6:
SET END LIMIT (ENDMO) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
ERREXIT IF NEXT TOKEN IS NOT EOS :ERROR1:
IF BEGIN LIMIT (BEGIN) = 0, THEN
SET BEGIN LIMIT (BEGIN) TO 1 (BEGIN IS NOW A TABLE INDEX)
ELSE
START SEARCH FROM FIRST TO LAST SEQ. TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = BEGIN LIMIT (BEGIN)
SET BEGIN LIMIT (BEGIN) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERROR6:
ENDIF
ELSE
ENDIF
IF END LIMIT (ENDMO) = 0, THEN
SET END LIMIT (ENDMO) TO INDEX OF LAST TABLE ENTRY
ELSE
START SEARCH FROM BEGIN LIMIT (BEGIN) TO LAST TABLE ENTRY
EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDMO)
SET END LIMIT (ENDMO) TO INDEX OF THIS ENTRY
ORELSE
INCREMENT INDEX TO NEXT TABLE ENTRY
ENDLOOP
ERREXIT :ERROR6:
ENDIF
ENDIF
EXIT XSLICE
:ERROR1: CALL XMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
:ERROR6: CALL XMSG - 'INVALID SEQUENCE NUMBER'
:ERROR8: CALL XMSG - 'INVALID SEQUENCE NUMBER RANGE'
END XSCAN
FORTRAN CALLING PROCEDURE

CALL XPMT

********

XPMT PROCESSES THE SEQUENCE TABLE EDITOR PROMPT DIRECTIVE

********

INPUT

COMMON AE - COMBUF, COMPTB, LU, TOKENS

COMMON XB - DEBUG

********

OUTPUT

COMMON XB - PRMTBH, TABNOX

********

NOTES

USES ROUTINES

XRMSG

XUDMG

********

BEGIN XPMT

ERR01: IF TOKEN IS NOT COMMA :ERR01:

ERR01: IF TOKEN IS NOT A NAME :ERR01:

ERR01: IF TOKEN IS NOT EOS :ERR01:

IF NAME IS 'M', THEN

SET PROMPT MODE TO CREATE

ELSE

ERR01: IF NAME IS NOT 'A':ERR01:

SET PROMPT MODE TO ALL

SET TABLE ENTRY INDEX (TABNOX) TO 0

ENDIF

EXIT XPMT

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

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

END XPMT

602 2  CD0      FORTRAN CALLING PROCEDURE
604 2  CD0      CALL XSDEL
606 2  CD0      ********
607 2  CD0      ********
608 2  CD0      XSDel IS THE SEQUENCE TABLE EDITOR DELETE DIRECTIVE PROCESSOR
609 2  CD0      ********
610 2  CD0      ********
611 2  CD0      INPUT
612 2  CD0      ********
614 2  CD0      COMMON XE - Lu
615 2  CD0      ********
617 2  CD0      COMMON XB - BEGIN, DEBUG, ENDNO, IRETC, NUMENT
618 2  CD0      ********
619 2  CD0      OUTPUT
620 2  CD0      ********
622 2  CD0      COMMON XB - NUMENT, WBUF
623 2  CD0      ********
625 2  CD0      NOTES
626 2  CD0      ********
627 2  CD0      USES ROUTINES
628 2  CD0      XSCAN
629 2  CD0      XDBG
630 2  CD0      ********
633 2  CD0      BEGIN XSDel
634 3  CD0      CALL XSCAN TO SCAN AND INTERPRET SEQ. LIMITS ON THE DIRECTIVE
635 3  CD0      IF NO ERROR INDICATED, THEN
636 4  CD0      DO FROM THE BEGIN LIMIT (BEGINO) TO THE END LIMIT (ENDNO)
637 5  CD0      MARK THIS SEQ. TABLE ENTRY AS DELETED
638 4  CD0      ENDDO
639 4  CD0      DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR
640 4  CD0      UNTIL A NON-DELETED ENTRY FOUND
641 5  CD0      IF ENTRY IS MARKED DELETED, THEN
642 6  CD0      DECREMENT NUMBER OF TABLE ENTRIES (NUMENT) BY ONE
643 5  CD0      ENDIF
644 4  CD0      ENDDO
645 4  CD0      ENDIF
646 4  CD0      END
647 3  CD0      END XSDEL
648 2  CD0
657   2  CD0  FORTRAN CALLING PROCEDURE
658   2  CD0
659   2  CD0
660   2  CD0
661   2  CD0
662   2  CD0
663   2  CD1  XSLTS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR
664   2  CD0
665   2  CD0
666   2  CD0
667   2  CD0
668   2  CD0
669   2  CD0
670   2  CD0
671   2  CD0
672   2  CD0
673   2  CD0
674   2  CD0
675   2  CD5  NOTES
676   2  CD5
677   2  CD5
678   2  CD5
679   2  CD5
680   2  CD5
681   2  CD5
682   2  CD5
683   2  CD5
684   2  CD5
685   2  CD5
686   2  CD5
687   2  CD5
688   2  CD5
689   2  CD5
690   2  CD5
691   2  CD5
692   2  CD5
693   2  CD5
694   2  CD5
695   2  CD2
696   2  CD2
697   2  CD2
698   2  CD2
699   2  CD2
700   2  CD2
701   2  CD2
702   2  CD2
703   2  CD3
704   2  CD3
705   2  CD3
706   2  CD3
707   2  CD3
708   2  CD3
709   2  CD3
710   2  CD3
711   2  CD3
712   2  CD3
713   2  CD3
714   2  CD3
715   2  CD3
FORTRAN CALLING PROCEDURE FOR TERMINAL COMMUNICATIONS:

CALL XICOM (PROMPT, PRMLN, RETCOD)

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

INPUTS FROM CALLING SEQUENCE:

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

OUTPUTS IN CALLING SEQUENCE:

RETCOD - (INTEGER, 1 WORD) IS A COMPLETION CODE DEFINED
AS FOLLOWS:
0 - NORMAL RETURN, BUFFER CONTAINS RESPONSE
1 - USER RESPONDED '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 - CHARACTERS (38 WORDS)
5 - USER REQUESTED A CONTINUATION

INTERNAL VARIABLES

CONMSG - 6 WORD ARRAY CONTAINING THE CONTINUATION
MESSAGE
PREFIX - 4 CHARACTERS USED AS PREFIXES TO PROMPT,
CORRESPONDING TO CODES IN XE(5)
RETCP - RETURN CODE FROM XIPRM EXTENDED PROMPTING
RETLA - RETURN CODE FROM XILAN LEXICAL ANALYSIS
SUFFIX - 1, CHARACTERS APPENDED TO END OF PROMPT

XE COMMON USED:

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

XS COMMON USED:


131 1 BEGIN READSEG
132 2      CALL XORMOV TO INITIALIZE INPUT BUFFER TO BLANKS
133 3      CALL XRECAL TO READ RESPONSE TO PROMPT
134 4      CALL XRPK ROUTINE TO CONVERT A2 FORMAT BUFFER TO R1
135 5      IF NUMBER OF WORDS READ IS NOT ZERO THEN
136 6      CALL XTLAM ROUTINE TO BUILD COMMUNICATIONS BUFFER
137 7      ELSE
138 8      IF LAST LA RETURN CODE WAS A CONTINUE THEN
139 9      REMOVE TRAILING COMMAS FROM COMBUF
140 0      SET LA RETURN CODE TO NORMAL RETURN
141 1      ELSE
142 2      SET LA RETURN CODE TO SAY USER ENTERED CR
143 3      ENDIF
144 4      ENDIF
145 5      END READSEG
FORTRAN CALLING PROCEDURE FOR LEXICAL ANALYSIS:

CALL XTLAM (RETC)

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

OUTIITS IN CALLING SEQUENCE:

RETC - (INTEGER, 1 WORD) IS A COMPLETIN 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.
1X - ERROR IN RESPONSE AT OR BEYOND CHARACTER XX.
2X - OVERFLOW/UNDERFLOW DETECTED AT OR BEYOND CHARACTER XX.

INTERNAL VARIABLES

COMLEN - LENGTH IN WORDS OF COMBUF = 256
DBLINT - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER VALUE
DBLW - 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
LAST - LAST CHARACTER WAS A COMMA
FLGENG - END LOOP FLG
0 - CONTINUE LOOP
1 - END LOOP
FLGNEG - NEGATIVE EXPONENT FLG
0 - EXPONENT WAS POSITIVE
1 - EXPONENT WAS NEGATIVE
FLGTYP - TYPE OF REAL VALUE
0 - SINGLE PRECISION
1 - DOUBLE PRECISION
POWER - EXPONENT PART OF A REAL NUMBER
RELDU - SINGLE PRECISION LOCATION FOR REAL VALUE
SPCHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING THE 8! FORMAT REPRESENTATION FOR:
4-9/0#%&@#*()'
X IS AN EXCLAMATION POINT
Y IS AN OPEN BRACKET
Z IS A BACK SLÁSH
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>206</td>
<td>CD4</td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>CD5</td>
<td><strong>XE COMMON USED:</strong></td>
</tr>
<tr>
<td>208</td>
<td>CD5</td>
<td><strong>EQUIVALENCE (XE(85), TOKENS), (XE(145), COMBUF),</strong></td>
</tr>
<tr>
<td>209</td>
<td>CD5</td>
<td><strong>(XE(145), NOTOK), (XE(146), NOWDS),</strong></td>
</tr>
<tr>
<td>210</td>
<td>CD5</td>
<td><strong>XS COMMON USED</strong></td>
</tr>
<tr>
<td>211</td>
<td>CD5</td>
<td><strong>EQUIVALENCE (XS(1), INBUF), (XS(81), NOSCHAR),</strong></td>
</tr>
<tr>
<td>212</td>
<td>CD5</td>
<td><strong>(XS(122),FLAG5), (XS(186), SCRATCH)</strong></td>
</tr>
<tr>
<td>213</td>
<td>CD5</td>
<td><strong>SUBROUTINES AND FUNCTIONS CALLED</strong></td>
</tr>
<tr>
<td>214</td>
<td>CD5</td>
<td><strong>XRPCX, XRMOV</strong></td>
</tr>
<tr>
<td>215</td>
<td>CD5</td>
<td><strong>PDL ROUTINES INCLUDED:</strong></td>
</tr>
<tr>
<td>216</td>
<td>CD5</td>
<td><strong>XTILAN, COMMA, ALPHA, DIGIT, DCOL, DECP, EORD</strong></td>
</tr>
<tr>
<td>217</td>
<td>CD5</td>
<td><strong>INTEGRA, REAL, DBL, REPET, INVAL, SCHARS,QUOTE</strong></td>
</tr>
</tbody>
</table>
BEGIN XILAN
SET END FLAG OFF
SET LAST CHARACTER WAS A COMMA ON
INITIALIZE INDEX INTO INPUT BUFFER
INITIALIZE RETURN CODE TO NORMAL RETURN
DO WHILE END FLAG IS OFF
IF INPUT BUFFER HAS BEEN COMPLETELY SCANNED THEN
IF LAST CHARACTER WAS A COMMA OR A SYMBOLIC STRING IS STILL OPEN THEN
SET RETURN CODE TO SAY CONTINUATION REQUESTED
ENDIF
SET END FLAG ON
ELSE
EXIT 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
ENDO
STORE END OF BUFFER TOKEN IN COMBUF
INCREMENT #TOKENS BY 1
END XILAN

BEGIN COMMA
SET LAST CHARACTER WAS A COMMA ON
ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
STORE COMMA TOKEN IN COMBUF
INCREMENT #WORDS IN COMBUF BY 1
INCREMENT #TOKENS BY 1
GET NEXT INPUT CHARACTER
END COMMA
274 1 BEGIN ALPHA
275 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
276 2 CALL XRMONEY TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
277 2 DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
278 3 INPUT CHARACTER IS A NUMERIC OR
279 3 INPUT CHARACTER IS AN EXCLAMATION POINT) AND
280 3 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
281 3 MOVE CHARACTER INTO TEMPORARY BUFFER
282 3 GET NEXT INPUT CHARACTER
283 2 ENDDO
284 2 SET CHARACTER COUNT = 6
285 2 STORE CHARACTER NAME TOKEN IN COMBUF
286 2 CALL ZPCK ROUTINE TO PACK CHARACTERS INTO COMBUF
287 2 INCREMENT #WORDS IN COMBUF BY 4
288 2 INCREMENT #TOKENS BY 1
289 1 END ALPHA
290 1 *
291 1 *
292 1 *
293 1 BEGIN DIGIT
294 2 INITIALIZE POWER TO ZERO
295 2 PERFORM DCOL
296 2 IF INPUT BUFFER IS NOT EXHAUSTED THEN
297 3 IF INPUT CHARACTER IS A 0 THEN
298 4 PERFORM DECPT
299 3 ELSE
300 4 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
301 5 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
302 5 PERFORM EOMB
303 4 ELSE
304 5 IF INPUT CHARACTER IS AN "R" THEN
305 6 PERFORM REPEET
306 5 ELSE
307 6 PERFORM INTEGR
308 5 ENDF
309 4 ENDF
310 3 ENDF
311 2 ELSE
312 3 PERFORM INTEGR
313 2 ENDF
314 2 END DIGIT
1 BEGIN DCOL
2   SET INTEGER = 0
3   SET COUNTER = 0
4   DO WHILE CHARACTER IS A DIGIT AND
5      INPUT BUFFER IS NOT EXHAUSTED
6       SET INTEGER = (INTEGER + 10) + CURRENT CHARACTER - 48
7       ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
8       INCREMENT COUNTER BY 1
9   GET NEXT CHARACTER
10  ENDDD
11 1 END DCOL
12
131 1 BEGIN DECPT
14   CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
15   GET NEXT INPUT CHARACTER
16   IF INPUT BUFFER IS NOT EXHAUSTED THEN
17      IF INPUT CHARACTER IS A DIGIT THEN
18         PERFORM DCOL
19         ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
20         ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
21      ELSE
22         IF INPUT CHARACTER IS AN "E" OR A "D" THEN
23            PERFORM EORD
24         ELSE
25            PERFORM REAL
26         ENDIF
27      ELSE
28         PERFORM REAL
29      ENDIF
30   ELSE
31      PERFORM REAL
32   ENDIF
33 1 END DECPT
349 1 BEGIN EORD
350 2 IF INPUT CHARACTER IS AN "E" THEN
351 3 SET TYPE FLAG TO "E"
352 4 ELSE
353 5 SET TYPE FLAG TO "D"
354 6 ENDIF
355 7 GET NEXT CHARACTER
356 8 IF INPUT BUFFER IS EXHAUSTED
357 9 SET NEGATIVE FLAG OFF
358 10 IF CHARACTER IS A - THEN
359 11 SET NEGATIVE FLAG ON
360 12 GET NEXT CHARACTER
361 13 ELSE
362 14 IF CHARACTER IS A + THEN
363 15 GET NEXT CHARACTER
364 16 ENDIF
365 17 ENDIF
366 18 EXIT IF INPUT BUFFER IS EXHAUSTED OR
367 19 EXIT IF CHARACTER IS NOT A DIGIT PERFORM INVALID
368 20 PERFORM BCOL
369 21 IF NEGATIVE FLAG IS ON THEN
370 22 SET POWER = -POWER
371 23 ENDIF
372 24 IF TYPE FLAG IS "E" THEN
373 25 PERFORM REAL
374 26 ELSE
375 27 PERFORM DBL
376 28 ENDIF
377 29 1 END EORD
1 BEGIN INTEGER
2   ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
3   CONVERT NUMBER TO INTEGER
4   ERROR IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
5   STORE INTEGER TOKEN IN COMBUF
6   INCREMENT WORDS IN COMBUF BY 2
7   INCREMENT TOKENS BY 1
8 END INTEGER
9
10 BEGIN REAL
11  ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
12  SET REAL = DOUBLE PRECISION * 10 ** POWER
13  ERROR IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
14  STORE REAL TOKEN IN COMBUF
15  INCREMENT WORDS IN COMBUF BY 3
16  INCREMENT TOKENS BY 1
17 END REAL
18
19 BEGIN DDBL
20  ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
21  SET DOUBLE = DOUBLE PRECISION * 10 ** POWER
22  ERROR IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
23  STORE DOUBLE TOKEN IN COMBUF
24  INCREMENT WORDS IN COMBUF BY 4
25  INCREMENT TOKENS BY 1
26 END DDBL
27
28 BEGIN REPEAT
29  ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
30  ERROR IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL
31  STORE REPEAT TOKEN IN COMBUF
32  INCREMENT WORDS IN COMBUF BY 2
33  INCREMENT TOKENS BY 1
34  GET NEXT CHARACTER
35 END REPEAT
BEGIN SCARS

CHARACTER TABLE:

"-#80#80=#"

NH ARE INVALID CHARACTERS HERE

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 INVALID - NO RETURN

END SEARCH

SET NEXT FLAG ON


A: Z, E, Y, X, N


A: Z, E, Y, X, N

SET NEXT FLAG OFF

PERFORM QUOTE

SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED

SET END FLAG ON

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

SET RETURN CODE TO SAY X ENTERED

SET END FLAG ON

IF FOLLOWING CHARACTER IS A DIGIT THEN

SET NEXT FLAG OFF

SET INTEGER = 0

SET POWER = 0

PERFORM DECPT

ENDIF

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
BEGIN QUOTE
GET NEXT CHARACTER
SET #CHARACTERS = 0
DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
(INPUT CHARACTER IS A QUOTE AND
NEXT CHARACTER IS A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
INCREMENT #CHARACTERS BY 1
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
ENDIF
ENDDO
ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
ERROR IF LENGTH OF CHARACTER STRING IS 0 OR
ERROR IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVAL
STORE CHARACTER STRING TOKEN IN COMBUF
CALL XRPCX TO PACK CHARACTERS INTO COMBUF
INCREMENT #WORDS IN COMBUF BY 2+((#CHARACTERS+1)/2)
INCREMENT #TOKENS BY 1
GET NEXT CHARACTER
END QUOTE
FORTAN CALLING PROCEDURE

CALL XTPRM

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

INPUT

XE COMMON - CARTRG, COMBUF, FLAGS, LU, MOPROC, PRCKAM, SUBSTA,

TOKENS

XB COMMON - AREPT, VKBUF (FROM INTERFACE TABLE EDITOR)

VARIOUS FDS PROMPT FILES (SEE INTERNAL VARIABLE TABLE)

VARIABLES

USED FOR SCRATCH SPACE

LISTING OF APPROPRIATE EXTENDED PROMPTS

INTERNAL VARIABLES

CONTIN - CONTINUATION INDICATOR (1) FOR CURRENT TABLE ENTRY

FILE - FILE NAME OF CURRENT TABLE ENTRY

INDEX - INDEX TO CURRENT TABLE ENTRY

L - RECORD NUMBER OF TEXT OR SYNTAX BLOCK CORRESPONDING TO

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

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

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

SECTION 6.2.4.3)

RECORD 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

BEGIN SETUP
EXIT TO :ERO2: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
CASE (:X:, :S:, :F:, :/) SUBSTA

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

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

:Fi
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

:Ti
FORM TENTH ENTRY FILE NAME FROM > AND IT EDITOR CURRENT PROCESSOR NAME
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 NINETH ENTRY
ENDIF
ENDIF
END CASE
EXIT SETUP

:ERO2: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
END SETUP
FORTRAN CALLING PROCEDURE

CALL XUDBG (I,U, ID)

* * *

CD1

XUDBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY

CD2

DUMP AND/OR MODIFICATION FACILITY

CD3

INPUT (CALLING SEQUENCE)

CD4

LU - LOGICAL UNIT TO INTERACT WITH IN INVOKING XUDBG OPTIONS

CD5

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

CALLER

CD6

INPUT (INTERACTIVE)

CD7

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

CD8

S = SNAP OUT (DUMP) MEMORY

CD9

M = MODIFY MEMORY

CD10

E = EXIT XUDBG

CD11

START: - OCTAL MEMORY ADDRESS IN THE USERS MAP OF FIRST

CD12

WORD TO BE SNAPED OR MODIFIED

CD13

STOP: - OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPED

CD14

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

CD15

WRITTEN

CD16

VALUES: - ARRAY OF OCTAL VALUES TO BE STORED IN MEMORY

CD17

BEGINNING AT LOCATION "START", WHEN INPUT

CD18

"VALUES" MUST NOT EXCEED 50 CHARACTERS. NULL

CD19

FIELDS; I.E., SUCCESSIVE COMMAS, INDICATE WORDS

CD20

OF ZERO TO BE STORED.

CD21

* * *

CD22

OUTPUT (TO 'LU')

CD23

HEADER - "*** XUDBG FROM IDIDID"

CD24

PROMPTS - (SEE INPUT)

CD25

* * *

CD26

OUTPUT (TO 'OUTPUT UNIT')

CD27

HEADER - "*** XUDBG FROM IDIDID"

CD28

SNAP - 102 WORD DUMP FORMATTED LINE (SEE XUDPL)

CD29

* * *

CD30

BASE - NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE

CD31

CLASS - CLASS I/O NUMBER FOR TERMINAL INPUT

CD32

LENGTH - LENGTH OF CHARACTER STRING BEING MANIPULATED

CD33

LINE - EIGHT WORD BUFFER OF WORD TO BE SNAPED

CD34

LUI - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET

CD35

LUG - LOGICAL UNIT FOR SNAP OUTPUT

CD36

OPTN - ONE CHARACTER EXECUTION OPTION CODE

CD37

ORIGIN - REFERENCE POINT FOR MEMORY ACCESS OFFSET COMPUTATION

CD38

ORG - ADDRESS OF 'ORIGIN'

CD39

RST - 'ORIGIN' RELATIVE END OF DATA TO BE SNAPED

CD40

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

CD41

CONVERSION -

CD42

* * *

CD43

NOTES

CD44

* * *
1 BEGIN XU86
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 XU86 TO FORMAT LINE
17 OUTPUT LINE
18 END DO
19 ENDIF
20 END DO
21 BEGIN XU86
22 ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
23 RETRIEVE RESPONSE
24 CLEAR SCREEN
25 DO FOR EACH CHARACTER
26 SHIFT SCREEN AND ADD NEXT DIGIT
27 END DO
28 END PROMPT AND CONVERSION
GENERAL FILE DUMP PROGRAM FOR FILE MANAGER FILES

NAME - NAME OF FM FILE TO BE DUMPED
INREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED
(FIRST RECORD IS RECORD NUMBER ONE)
NBREC - 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, MAXO, OPEN, POINT, READF, RMAPR, XPRES, XNOW, XUPL

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

DO FOREVER
READ FILE NAME
EXIT XUDPF 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
ELSE
READ LF 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
ELSEDO :ERROR: OUTPUT MESSAGE
ENDDO
ELSE
END XUDPF
158 1 C00  FORTRAN CALLING PROCEDURE
159 1 C00
160 1 C00  CALL XUDPL (ADDRESS, LINE, BUFFER)
161 1 C00
162 1 C00********
163 1 C01
164 1 C01  PRODUCE AN OCTAL AND ASCII PRINT FORMATTED MEMORY DUMP LINE IMAGE
165 1 C01
166 1 C01********
167 1 C02  INPUT
168 1 C02  ADDRESS - TWO WORD INTEGER ARRAY CONTAINING THE ABSOLUTE AND
169 1 C02  RELATIVE ADDRESS TO BE FORMATTED WITH THE LINE
170 1 C02  LINE - EIGHT WORD ARRAY TO BE CONVERTED TO OCTAL AND ASCII
171 1 C02  BUFFER - FIFTY-ONE WORD BUFFER TO HOLD FORMATTED PRINT LINE. MUST
172 1 C02  BE BLANKED PRIOR TO FIRST CALL TO XUDPL AND NOT STORED
173 1 C02  INTO BETWEEN CALLS TO XUDPL.
174 1 C02
175 1 C02********
176 1 C03
177 1 C03  OUTPUT
178 1 C03  BUFFER - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE
179 1 C03  COLUMNS CONTENTS
180 1 C03  3-8 FIRST ADDRESS
181 1 C03  11-16 SECOND ADDRESS
182 1 C03  21-26 OCTAL REPRESENTATION OF 'LINE'
183 1 C03  87-102 ASCII REPRESENTATION OF 'LINE'
184 1 C03
185 1 C03********
186 1 C05  NOTES
187 1 C05  USES XREX, XRO6, XRSET
188 1 C05
189 1 C05********
190 1 *
191 1 *
192 1 *
193 1 *
194 1 BEGIN XUDPL
195 2 CALL XRO6 TO CONVERT EACH WORD OF ADDRESS TO OCTAL
196 3 DO FOR EACH WORD OF LINE
197 4 CALL XRO6 TO CONVERT WORD TO OCTAL
198 5 DO FOR EACH BYTE OF WORD
199 6 IF BYTE < 40 OR BYTE > 1368
200 7 THEN
201 8 REPLACE BYTE WITH ASCII PERIOD
202 9 ENDIF
203 10 END DB
204 11 ENDDB
205 1 END XUDPL
CD1 FORTRAN MAIN PROGRAM XUFMT IS SCHEDULED BY XUDMP TO PRINT
CD1 A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD1

CD2 INPUT
CD2 ICTL - CONTROL WORD FOR EXEC CALL READS. CONTAINS THE
CD2 LU NUMBER OF WHERE DUMP IS ON DISK
CD2 TRACK - TRACK NUMBER OF A 4 TRACK GLOBALLY ALLOCATED AREA
CD2 CONTAINING THE DUMP

CD2 OUTPUT
CD2 FORMATTED DUMP TO LU 6

CD2 EXTERNAL REFERENCES
CD5 EXEC
CD5 XICPR
CD5 XIMOV
CD5 XUDPL

BEGIN XUFMT

CALL XICPR TO GET LU AND STARTING TRACK NOS.

READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER

.COUNT OF ID-SEGS TO BE DUMPED

.UP TO 7 ID-SEGMENT ADDRESSES

.LOW AND HIGH BASE PAGE ADDRESSES

.LOW AND HIGH MAIN MEMORY ADDRESSES

.POINT TO 2ND SECTOR OF DUMP DATA

.DO UNTIL ALL ID-SEGS PRINTED

.DO UNTIL A EIGHT-WORD LINES PRINTED

.PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT

ENDO

_INCREMENT TO NEXT SECTOR OF DUMP DATA

ENDO

COMPUTE N, THE NO. OF 8-WORD LINES IN THE BASE PAGE DUMP

PERFORM COMPARE AND PRINT FUNCTION

COMPUTE NTKS, NO. OF DISK TRACKS OF MAIN MEMORY TO BE READ

.DO UNTIL NTKS ARE READ

.READ NEXT TRACK FROM DISK

COMPUTE N, THE NUMBER OF 8-WORD LINES TO DUMP

PERFORM COMPARE AND PRINT FUNCTION

ENDO

RELEASE THE GLOBALLY ALLOCATED TRACKS

EXIT XUFMT

BEGIN COMPARE AND PRINT FUNCTION

.DO UNTIL N LINES PROCESSED

IF NOT 1ST LINE, THEN

CALL XICPR TO COMPARE WITH PREVIOUS LINE

IF LINES ARE IDENTICAL, THEN

IF THIS IS 1ST OF A SERIES, THEN

WRITE 'DUPLICATE LINE'

ENDIF

ELSE

CALL XUDPL TO FORMAT THE DUMP LINE

WRITE FORMATTED DUMP LINE

ENDIF

ELSE

CALL XUDPL TO FORMAT THE DUMP LINE

WRITE FORMATTED DUMP LINE

ENDIF

ENDO
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
1 #01 ENTRY XVADB
2 #02 TYPE 1A ROUTINE TO ABORT CURRENT ID AND
3 #03 REQUEST A DUMP FOR THE ID.
4 #01 ENTRY XVADB
5 #01 TYPE 1A ROUTINE TO DUMP FROM CURRENT ID
6 #01 THE REQUESTED DATA AREAS AND THE CURRENT
7 #01 ID SEGMENT AND ITS FATHER CHAIN SEGMENTS
8 #02 INPUTS: NONE
9 #03 OUTPUTS: SETS CURRENT ID TO ABORTED STATE
10 #03 EXTERNALS: $ABRT, $ABRE, $XERN, $LIBR
11 CALL $LIBR. REENTRANT ROUTINE CALL
12 GET CURRENT ID FROM XERT (LOCATION 1717)
13 INCREMENT TO ADDRESS BOUNDARIES
14 GET ADDRESS BOUNDARIES
15 STORE := XVADB PARAMETER LIST
16 PERFORM XVADB DUMP THIS PARTITION
17 CALL $ABRT ABORT THIS PARTITION
18 CALL $ABRE FREE CURRENT REENTRANT DATA BLOCK
19 #01 EXIT TO :$XERN GO TO DISPATCHER
20 #01 XVADB
21 #01 XVADB
22 #01 XVADB
23 #01 XVADB
24 #01 XVADB
25 #01 XVADB
26 #01 XVADB
27 #01 XVADB
28 #01 XVADB
29 #01 XVADB
30 #01 XVADB
31 #01 ENTRY XVADB
32 #02 INPUTS: ADDRESS OF DOUBLE WORD CONTAINING START AND END ADDRESSES
33 #02 DEFINING AREA TO BE DUMPED
34 #03 OUTPUTS: OUTPUTS TO DISK
35 #03 HEADER DATA CONTAINING NO. OF ID SEGMENTS TO FOLLOW
36 #03 CURRENT ID SEGMENT AND ALL FATHER CHAIN ID SEGMENTS
37 #03 MEMORY IMAGE OF BASE PAGE SEGMENT FOR THIS PARTITION
38 #03 MEMORY AS SPECIFIED BY INPUT ADDRESS RANGE
39 #03 SCHEDULES XVFM TO FORMAT DATA FROM DISK TO PRINT
40 #03 EXTERNALS: $LIBR, $LIBX, EXEC, XVFM
41 CALL $LIBR TO NOTIFY OF REENTRANCY AND BECOME PRIVILEGED
42 RUN FATHER ID SEGMENT CHAIN SAVING ADDRESS OF EACH
43 CALL $LIBX TO BECOME NON-PRIVILEGED
44 CALL EXEC FOR A GLOBAL ALLOCATION OF DISK
45 CALL EXEC TO WRITE HEADER DATA USING CLASS 1/0
46 DO UNTIL ALL ID SEGMENTS PROCESSED
47 WRITE ID SEGMENT TO DISK
48 #02 ENDO
49 WRITE BASE PAGE SEGMENT TO REMAINING PORTION OF THIS TRACK
50 DO UNTIL END ADDRESS REACHED
51 WRITE ONE SECTOR OF DATA
52 #02 ENDO
53 #02 SCHEDULE XVFM WITH LU AND STARTING TRACK NOS.
54 CALL $LIBX TO RETURN FROM REENTRANT ROUTINE
55 #01 XVADB
56 #01 XVADB
57 #01 XVADB
58 #01 XVADB
59 #01 XVADB
60 #01 XVADB
01 TYPE 14 ROUTINE TO CONTROL COMMUNICATION BETWEEN AND
02 EXECUTION OF FDS MANAGER AND IT'S ASSOCIATED TASKS
03 (EXECUTIVE, PROCESSORS, AND UTILITIES).
04 ENTRY XVPAM AND XVSUB
05 INPUTS
06 FROM AN ASSOCIATED TASK
07 CALL XVPAM(PARMS)
08 ASSEMBLY FORM
09 JSB XVPAM
10 DEF #2 RETURN ADDRESS
11 DEF PARMS A(PARMS)
12 WHERE PARMS ARE P1,P2,P3,P4,P5
13 P1 IS THE SERVICE REQUEST
14 0 = NORMAL TERMINATION (P2-P5 NOT USED)
15 1 = WORK AREA REQUEST (P2-P5 NOT USED)
16 2 = EXECUTE A SEQUENCE TABLE
17 (P2-P4 HAS TABLE NAME)
18 (P5 INDICATES EXECUTION CONTROL IN CLASS I/O BUFFER)
19 3 = RESET SEQUENCE POINTERS
20 (P2 HAS SEQUENCE NUMBER)(P3-P5 NOT USED)
21 8 = TERMINATE SEQUENCE (P2-P5 NOT USED)
22 9 = TERMINATE FDS FUNCTION (P2-P5 NOT USED)
23 -32767 = ABNORMAL TERMINATION OF ASSOCIATED TASK
24 FROM AN FDS MANAGER
25 ASSEMBLY FORM
26 JSB XVPAM
27 DEF (RETURN POINT)
28 OCT 0
29 DEF PARMS A(FDS MANAGER RESPONSE)
30 OUTPUTS
31 TO AN FDS MANAGER
32 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
33 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
34 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
35 MANAGER IS ACTIVATED VIA SLIST
36 TO AN ASSOCIATED TASK
37 RESPONSE PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
38 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
39 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
40 ASSOCIATE TASK IS ACTIVATED VIA SLIST
103 1 BEGIN XVPAM
104 2 CALL BLIBN, BECOME PRIVILEGED
105 2 SET STOP-ID FROM XRT (OCT 1717)
106 2 IF THIS IS A MANAGER RESPONSE
107 2 THEN SET UP TO ACTIVATE ASSOCIATED TASK AND SUSPEND MANAGER
108 3 CALL SEQUENCE IS RETURN.O,A(PARMS)
109 2 PERFORM XVPAM POST REQUESTOR AND WAIT
110 2 ELSE SET UP TO ACTIVATE MANAGER AND SUSPEND ASSOCIATED TASK
111 3 CALL SEQUENCE IS RETURN,A(PARMS)
112 2 PERFORM XVPAM POST MANAGER AND WAIT
113 2 EIND
114 2 * STOP-ID HAS ID-SEGMENT TO BE SUSPENDED,
115 2 * AWAKEN-ID HAS ID-SEGMENT TO BE ACTIVATED,
116 2 CALL SLIST(SCHEDULE,AWAKEN-ID)
117 2 MOVE RETURN ADDRESS TO $SUSP OF STOP-ID.
118 2 CALL SLIST(WAIT,STOP-ID)
119 1 EXIT TO $XEN RTE DISPATCHER
120 1 END XVPAM
122 BEGIN XVPMAW
123 1 * DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
124 2 * FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
125 3 SET TARGET-ID FROM CURRENT-ID-SEGMENT
126 4 DO WHILE FATHER-ID-NUMBER NE 0 OR FATHER IS WAITING
127 5 PERFORM MGRFNDFATHER-ID-SEGMENT,COUNT)
128 6 EXITIF COUNT .GT. 0
129 7 SET TARGET-ID TO FATHER-ID-SEGMENT
130 8 ENDDO
131 9 IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
132 10 CALL $SYM (12,*XVO3,SEGMENT-NAME) "XVO3,NAME" REQUESTING PROG
133 11 PERFORM PUMP
134 12 EXIT TO $LIBX TO ENABLE AND REDISPATCH
135 13 ENDF
136 14 SET AWAKEN-ID FROM FSD-ENTRY $SYM
137 15 GET REQUEST PARRS MOVE INTO ID-SEGMENT
138 16 SET STBRT FROM CURRENT-ID
139 17 FND XVPMAW
140 1 *
141 1 *
142 1 *
143 1 *
144 1 *
145 1 DETERMINE IF CALLER IS A VALID FDS MANAGER
146 1 *
147 1 * PERFORM MGRFNDFCURRENT-ID,COUNT)
148 1 * COUNT WILL BE 0 FOR NO MATCH.
149 1 * COUNT NOT EQUAL ZERO IMPLIES A MATCH
150 1 * AND FSD-ENTRY HAS MATCHING FSDTAB ENTRY ADDRESS
151 1 * IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
152 1 * CALL $SYM (12,*XVO1,SEGMENT-NAME) "XVO1 PROGRAM" REQUESTING PROG.
153 1 PERFORM PUMP
154 1 EXIT TO :EXEC THE DISPATCHER
155 1 ENDF
156 1 *
157 1 FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
158 1 *
159 1 SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
160 1 IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
161 1 CALL $SYM (12,*XVO2,SEGMENT-NAME) "XVO2 PROGRAM" ASSOCIATED PROG.
162 1 PERFORM PUMP
163 1 EXIT TO :EXEC THE DISPATCHER
164 1 ENDF
165 1 AWAKEN-ID-SEGMENT=STAB
166 1 IF MANAGER HAS REQUEST FOR ABORT, THEN
167 1 CALL UABRT FOR CURRENT AF
168 1 ENDF
169 1 *
170 1 * MOVE FDS MANAGERS INPUT PARRS TO ASSOCIATED TASK ID SEGMENT
171 1 MOVE PARRS TO ID-SEGMENT WORDS 2-6
172 1 END XVPMAW

...
172 1 BEGIN PDUMP
173 2 * SET A CALL TO XVABM AT CALLER'S SUSPEND POINT
174 2 * END PDUMP
175 1 *
176 1 *
177 1 *
178 1 *
179 1 *
180 1 *
181 1 *
182 1 BEGIN MGRFND INPUT IS TARGET-ID
183 2 SET XVSTB TABLE OF ACTIVE FDS MANAGERS
184 2 SET COUNT TO NUMBER IN TABLE NUMBER MANAGERS
185 2 SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
186 2 DO WHILE COUNT .GT. 0
187 2 EXIT IF TARGET-ID .EQ. MGRFND-ID-SEGMENT
188 3 SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
189 3 SET COUNT = COUNT - 1
190 2 ENDDO
191 2 * FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT .GT. 0
192 2 * COUNT = D MEANS NO MATCH
193 1 END MGRFND
194 1 *
195 1 *
196 1 *
197 1 *
198 1 BEGIN XVSTB
199 2 N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSTB
200 2 STBNS DEF N NUMBER-IN-TABLE
201 2 STBNC DEF O NUMBER-ACTIVE
202 2 STBNR DEF O STB RESOURCE NUMBER
203 2 STBRE EQU * ENTRY START
204 2 STBLU DEF O LU(OCTAL)
205 2 STBLA EQU 0 LU(ASCII)
206 2 STBGL DEF O AFDS-MANAGER-ID-SEGMENT
207 2 STBEX DEF O AFDS-EXECUTIVE-ID-SEGMENT
208 2 STBEC DEF O CLASS NUMBER FOR EXEC
209 2 STBAT DEF O (CURRENT-ASSOCIATED-TASK)
210 2 STBPC DEF O CLASS-NUMBER FOR PROCESSOR
211 2 STBGR DEF O ENTRY SAVE AREA
212 2 STBEE EQU * ENTRY "N"
213 2 STBEL EQU STBEE-STBES ENTRY LENGTH
214 2 STBEN EQU N NUMBER-OF-ENTRIES
215 2 ORG STBES
216 2 REP N DO FOR N ENTRIES
217 2 REP STBEL DEFINE STORAGE WORDS FOR AN ENTRY
218 2 DEF O
219 1 END XVSTB
CD***************
CD
CD**** FORTRAN CALLING PROCEDURE:
CD
CD** CALL XIAUT
CD
CD***************
CD
CD** XIAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE
CD
CD***************
CD
CD** INPUTS IN COMMON:
CD
CD** XE(5) MASSTA, XE(10) SERSTR, XE(11) SEGEND, XE(12) SEOPTR,
CD
CD** XE(140)TABEND, XB(1) NPROC, XB(2) LIBD,
CD
CD** XB(249) SENNO, XB(250) SENLEN, XB(251) SERTAB
CD
CD***************
CD
CD** OUTPUTS IN COMMON:
CD
CD** XE(5) MASSTA, XE(1) FLGTAB
CD
CD***************
CD
CD** COMMON USED:
CD
CD** EQUIVALENCE (XE(5), MASSTA)
CD
CD** +XE(10), +SERSTR, +XE(11), +SEGEND
CD
CD** +XE(12), +SEOPTR, +XE(140), +TABEND,
CD
CD** +XB(1), +NPROC, +XB(2), +LIBD,
CD
CD** +XB(249), +SENNO,
CD
CD** +XB(250), +SENLEN, +XB(251), +SERTAB,
CD
CD** +XE(1), +FLGTAB
CD
CD***************
CD
CD** FBS Routines Called:
CD
CD** XECPR, XEXIT, XREMV, XMSG,
CD
CD** XECXE, XISTO, XITMP
CD
CD***************
CD
CD** RTE Routines Called:
CD
CD** IOE
CD
CD***************
49 1 BEGIN XXAUT
50 1 IF ENTRY IS FROM A DIRECTIVE THEN
51 2 SET MASTER STATE TO INDICATE REENTRY
52 3 DO FOR EACH ENTRY IN THIS SEQUENCE TABLE
53 4 SEARCH LIBRARY DIRECTORY FOR THIS PROCESSOR
54 5 ERREIT IF PROCESSOR NOT FOUND TO :ERR1:
55 6 STUFF INTERFACE TABLE BIT AND VERSION INTO SEQUENCE TABLE ENTRY
56 7 END
57 8 CALL XXSPO TO STORE REVISED SEQUENCE TABLE IN APA AS $SEQTB
58 9 ELSE (I AM BEING REENTERED FROM INT.)
59 10 CALL XXSPO TO SET UP TEMPORARY ENTRY $ENMB
60 11 CALL $XXSEX TO EXECUTE FROM TEMPORARY ENTRY
61 12 IF SEQUENCE NUMBER IS NOT REQUESTED THEN
62 13 EXIT XXAUT IF TERMINAL ENTRY WAS JUST EXECUTED
63 14 SET STARTING ENTRY TO NEXT ENTRY
64 15 ENDIF
65 16 ENDDO
66 17 DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST FOR RESET
67 18 CALL $XXSEX TO EXECUTE REMAINDER OF TABLE
68 19 END
69 1 EXIT XXAUT
70 2 :ERR1:
71 2 CALL XXMSG TO DISPLAY INVALID PROCESSOR NAME
72 1 END XXAUT
FORTRAN CALLING PROCEDURE FOR EXECUTION CONTROLLER:

CALL XELS (XICNT)

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

Inputs from calling sequence:

XICNT - (INTEGER, 3 WORDS) array containing the name "XICNT" used by XELS to call EXEC to load the execution controller segment.

Internal variables:

MODE - (INTEGER, 1 WORD) mode in which the execution controller was called

0 - MANUAL
1 - SEMI-AUTOMATIC
2 - AUTOMATIC WITH TRACE
3 - AUTOMATIC

Common used:

EQUIVALENCE (XICNT, XICNTA)

FBS routines used

XETM, XEXT, XRMSG, XXMAP

Note: contains dummy call to XEXEC
119 1 BEGIN: NAME TO KEPT OF BITS 12 AND 13 OF OPST
121 3 NAME: CALL REANT
122 5 :SEM: CALL KEEN TO DISPLAY INVALID OPTION
123 3 :AUT: CALL REANT
124 3 :AUTO: CALL REANT TO RETURN TO SELECT LONG
125 2 CALL HUM TO RETURN TO SELECT LONG
126 2 END STICK
**FORTRAN CALLING SEQUENCE:**

```
132  CD************
134  CD          CALL XDEC (RETC)
138  CD************
140  CD          XDEC DECODES A RESPONSE OF PROCESSOR NAME (INT TABLE NAME)
141  CD          INTO A SEQUENCE TABLE ENTRY.
143  CD************
145  CD          INPUTS IN COMMON:
146  CD          XE85) TOKENS, XE145) COMBUF, XB(1) NPROC, XB(2) LIBD
150  CD************
151  CD          OUTPUTS IN CALLING SEQUENCE:
153  CD          RETC - RETURN CODE (O IS NORMAL RETURN)
156  CD          COMMON USED:
162  CD          EQUivalence
164  CD          + (XE85), TOKEHS), (X(144),TOKPTR),
166  CD          + (XB(1), NPROC), (XB(2), LIBD),
168  CD          + (XB(251), SEQTAB)
169  CD          FDS ROUTINES USED:
172  CD          XRCPR, XEXEC, XRMVO, XRMSS
173  CD          RTE ROUTINES USED:
176  CD          IAND
```

**REPORTIBILITY OF THE ORIGINAL TASK IS POOR**
1 BEGIN XXDEC
2 INITIALIZE RETURN CODE TO ZERO
3 SET SEQUENCE ENTRY TO ZEROS
4 ERREXIT IF FIRST TOKEN IS NOT A PROCESSOR NAME TO :ERR1:
5 SEARCH LIBRARY DIRECTORY FOR PROCESSOR
6 ERREXIT IF NAME NOT FOUND TO :ERR1:
7 MOVE PROCESSOR NAME, IT BIT AND VERSION INTO SEQUENCE ENTRY
8 IF AN INTERFACE TABLE NAME WAS ENTERED THEN
9 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
10 ENDIF
11 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
12 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
13 IF AN INTERFACE TABLE IS REQUIRED BUT NOT SPECIFIED THEN
14 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'SINTAB'
15 ENDIF
16 EXIT XXDEC
17 :ERR1:
18 CALL XRRUG TO DISPLAY ERROR
19 SET RETURN CODE TO SAY ERROR
20 END XXDEC
200     1 CD************
201     1 CD
202     1 CD0 FORTRAN CALLING PROCEDURE:
203     1 CD0
204     1 CD0
205     1 CD0 CALL XXDEF
206     1 CD0
207     1 CD0************
208     1 CD0
209     1 CD0 XXDEF READS IN THE DEFAULT INTERFACE TABLE FOR A PROCESSOR
210     1 CD0 AND STORES IT IN THE AWN AS GITAB
211     1 CD0
212     1 CD0************
213     1 CD2 INPUTS FROM COMMON:
214     1 CD2
215     1 CD2 PRNAME - (INTEGER, 3 WORDS) NAME OF PROCESSOR IN SERTAB
216     1 CD2 FOR WHICH NO INTERFACE TABLE WAS SUPPLIED
217     1 CD2
218     1 CD2************
219     1 CD4 INTERNAL VARIABLES:
220     1 CD4
221     1 CD4
222     1 CD4 DEFTAB - (INTEGER, 1200 WORDS) ARRAY WHERE MAXIMUM SIZE
223     1 CD4 DEFAULT INTERFACE TABLE CAN BE READ INTO
224     1 CD4 DEFNAME - (INTEGER, 3 WORDS) NAME OF INTERFACE TABLE
225     1 CD4 NAME IS CREATED FROM PROCESSOR NAME
226     1 CD4
227     1 CD4************
228     1 CD5 COMMON USED:
229     1 CD5
230     1 CD5
231     1 CD5 EQUIVALENCE (XK(5), MASSTA), (XK(6), SUBSTA),
232     1 CD5 (XK(13), INTMAP), (XK(16), PRNAME),
233     1 CD5 (XK(19), REPTR), (XK(20), REGBUF),
234     1 CD5 (XK(145), ICN),
235     1 CD5 (XK(15), TMPTAB), (XK(14), DEFNAME),
236     1 CD5 (XK(18), LEN1), (XK(19), LEN2),
237     1 CD5 (XK(20), IDC1), (XK(200), RILC)
238     1 CD5
239     1 CD5
240     1 CD5 RTE ROUTINES USED:
241     1 CD5
242     1 CD5
243     1 CD5 CLOSE, EXEC, KCVT, OPEN, READF
244     1 CD5
245     1 CD5
246     1 CD5 FDS ROUTINES USED:
247     1 CD5
248     1 CD5
249     1 CD5 XERIV, XREQ, XREX, XRMOV, XRMIG, XRPRC, XRPUR
BEGIN XXE
   INITIALIZE RESET INDEX TO ZERO
   IF TABLE FLAG SAYS SETTAB IN XB THEN
      CALL XVPAM WITH SEQ# START AND END
   ELSE - AM BEING CALLED TO EXECUTE A TEMPORARY SEQUENCE TABLE
      CALL XVPAM WITH SEQUENCE pointer AS START AND END
   ENDIF
   CALL RMPAR TO RETRIEVE RETURN PARAMETERS
   IF THE MANAGER DETECTED +1 ERROR TRYING TO EXECUTE THE SEQUENCE THEN
      SET SEQUENCE POINTER TO SEQUENCE # IN ERROR
      FIND ENTRY IN ERROR AND SAVE IN XE
   CASE ERROR (:INT1,:INT2,:INT3,:INT4,:INT5,:INT6:)
      :INT1: INTERFACE TABLE NOT SPECIFIED
      CALL XXDEF TO READ UP DEFAULT TABLE
      CALL XXTMP TO SET UP A TEMPORARY ENTRY
      PERFORM XEXE TO EXECUTE **NO RETURN**
      :INT2: SET MESSAGE TO INTERFACE TABLE NOT FOUND
      :INT3: INTERFACE TABLE NOT COMPLETE
      SET SUBSTATE TO INTERFACE TABLE EDITOR
      CALL XRTAB TO RETURN TO EXEC **NO RETURN**
      :INT4: SET MESSAGE TO INT TABLE NOT FOR PROCESSOR
      :INT5: SET MESSAGE TO VENDORS DO NOT MATCH
      CONVERT SEQUENCE # TO INDEX
      SAVE INDEX AND SEQUENCE NUMBER IN COMMON
      EXIT XXE
      :INT6: SET MESSAGE TO RESET SEQ# NOT FOUND
      :INT7: SET MESSAGE TO PROCESSOR ABENDED
      :INT8: SET MESSAGE TO AVA OVERFLOW
   ENDCASE
   CALL XREMS TO DISPLAY ERROR
   FORMAT SEQUENCE ENTRY INTO ASKII
   CALL XREMS TO DISPLAY SEQUENCE TABLE ENTRY IN ERROR
   IF MODE IS AUTO AND ENTRY IS NOT AN OVERRIDE THEN
      RESET OLD INDEX TO RE-EXECUTE THIS ENTRY
   ENDIF
   IF MODE IS AUTO THEN
      SET MASTER STATE TO ZERO
      CALL XRTAB TO ABORT SEQUENCE *** NO RETURN ***
   ENDIF
   RETURN
1 END XXE
**PROCEDURE FOR CALLING XIMAN:**

CALL XIMAN

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

**OUTPUTS TO COMMON:** XE(10) SEQI; XE(11) SEQI; XE(21) SEQI; XE(25) SEQI; XE(29) SEQI; XE(13) SEQI

**COMMON USED:**

- EQUIVALENCE (XST, XST; XSTM)
- XE(10) SEQI; XE(11) SEQI; XE(21) SEQI; XE(25) SEQI; XE(29) SEQI; XE(13) SEQI

**FNS ROUTINES USED:**

- XECE, XIMSN, XTRO

**RTE ROUTINES USED:**

- XCTC, XDEF, XISTO, XSTM
BEGIN XTMAN
IF ENTRY IS FROM A DIRECTIVE THEN
SET HASSTA TO INDICATE RE-ENTRY
DO UNTIL PERCENT IS ENTERED
  :PROMPT: CALL/XICOM TO PROMPT FOR PRNAME, "ITMNAME"
  IF PERCENT IS NOT ENTERED THEN
    ENREXIT IF CR ENTERED TO :PROMPT:
  CALL XDEC TO DECODE RESPONSE
  ENREXIT IF INVALID RESPONSE TO :PROMPT:
  SET SER #5 IN .E ; J ZEROS
  SET #ENTRIES IN SERTAC IN XE TO 1
  CALL XXSTO TO STORE SEQUENCE TABLE
  IF IT NAME IN SENTAB IS &INTAB THEN
    PROCESSOR REQUIRES AN IT THEN
    CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
  ENDF
  CALL XXEXE TO EXECUTE SENTAB
ENDDO
ELSE
  AM BEING REENTERED FROM INTERFACE TABLE EDITOR
  CALL XXMP TG SET UP TO EXECUTE A TEMPORARY TABLE
  CALL XXEXE TO EXECUTE ENTRY
  PERFORM XTMAN **NO RETURN**
ENDF
RETURN
  :PRMERR: CALL XRMSG TO DISPLAY ERROR
  PERFORM XTMAN **NO RETURN**
END XTMAN
FORTRAN CALLING SEQUENCE:

CALL XSEX

INPUTS IN COMMON:

XX(5) NASSTA, XX(10) SERSTA, XX(11) SESEND, XX(12) SEQTR,

XX(140) TABEND, XX(3) NOPROC, XX(2) LISO,

XX(246) SERNO, XX(250) SERLEN, XX(251) SERTAB

OUTPUTS IN COMMON:

XX(5) NASSTA, XX(13) FLGTA

INTERNAL COMMON USED:

XX(139) EXEND - ENDING SEQUENCE NUMBER USED TO

TERMINATE SEQUENCE

XX(14) CURIND - INDEX TO THE CURRENT ENTRY BEING EXECUTED

XX(235) RESIND - INDEX TO RESET CURRENT INDEX TO

XX(246) OLIND - INDEX TO THE LAST ENTRY EXECUTED IN THE

SEQUENCE TABLE

COMMON USED:

(XE(5), MASSTA),

(XE(10), SERSTA), (XE(11), SESEND), (XE(12), SEQTR),

(XE(13), SESEND), (XE(14), TABEND), (XE(15), CURIND),

(XE(16), CURIND), (XE(17), LISO), (XE(18), NOPROC),

(XE(19), SERTAB), (XE(20), SERNO), (XE(246), OLIND),

(XE(249), SERNO), (XE(250), SERTAB), (XE(251), SERTAB),

(XE(13), FLGTA)

FDS ROUTINES USED:

XXEP, XXEXT, XXISO, XXMOD,

XXINS, XXSET, XXDEC, XXDEF, XXEXF,

XXSTO, XXTMP, XXCM
BEGIN XXSEM:
  IF ENTRY IS FROM A DIRECTIVE THEN
  SET MASTER STATE TO SAT REENTRY
  DO FOR # ENTRY IN SEQUENCE TABLE
      SEARCH LIDC FOR PROCESSOR NAME
      ERREXIT IF NOT FOUND TO :ERR1:
  MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
  ENDIF
  SAVE ORIGINAL ENDING SEQUENCE #
  SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
  SET FLAST TO SAT EXECUTE ENTIRE SEQUENCE
  CALL XISTO TO STORE REQUEST IN AWA
  ELSE (I AM BEING REENTERED FROM INTE)
      CALL XETXP TO SET UP TEMPORARY EXECUTI*N
      CALL XERE TO EXECUTE ONLY THE FIRST ENTRY OF SEQU
      IF RESET WAS REQUESTED THEN
          RESET CURRENT SEQUENCE # TO NEW SEQUENCE #
      ELSE
      ENDIF
      EXIT XXRAUT IF TERMINAL ENTRY WAS JUST EXECUTED
  ENDIF
  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 XIDEF TO READ UP DEFAULT TABLE
  ENDIF
  SET BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
  SET ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
  CALL XERE TO EXECUTE
  SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
  ELSE
      IF RESPONSE WAS A SEQUENCE # THEN
          ERREXIT IF NUMBER IS ZERO TO :ERR1:
          SEARCH SEQUENCE TABLE FOR SEQUENCE #
          ERREXIT IF NUMBER IS NOT FOUND TO :ERR1:
          SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
      ELSE
          IF RESPONSE IS AN AMPERAND THEN
              SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
          ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
              CALL XECOC TO DECODE PROCESSOR NAME, IT NAME RESPONSE
              IF RSPONSE IS VALID THEN
                  CALL XISTO TO STORE OVERRIDING ENTRY
                  SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
                  SET CURRENT SEQUENCE # TO FIRST ENTRY
              ENDIF
          ENDF
      ENDF
  ENDF
ENDIF
ENDO
EXIT XXSEM
ERR1:
CALL XXMSG TO DISPLAY ERROR MESSAGE
END XXSEM
CALLING PROCEDURE FOR XXSTO:

CALL XXSTO

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

INPUTS FROM COMMON:

XBO(250) SERLEN, XBO(251) SEOTAB, X5(13) FLGTAB

OUTPUTS TO COMMON:

XE(5) MASSTA

COMMON USED:

EQUIVALENCE (XBO(5), MASSTA),

XBO(19) REOPTB, XBO(20) RENBU7,

XBO(250), SECLEN, XBO(251), SEOTAB,

XBO(13), FLGTAB

FDS ROUTINES USED:

XREQ, XRMVR, XRMSG, XERNX

RTE ROUTINES USED:

EXEC
600 1 BEGIN X3STO
610 2 SET CLASS NUMBER TO ZERO
611 3 IF TABLE FLAG SAYS STORE ENTIRE TABLE THEN
612 4 CALL EXEC TO WRITE ENTIRE TABLE
613 5 SET LENGTHS IN REQUEST BUFFER TO LENGTHS IN KB
614 6 CALL XMOV TO MOVE DELETE, ALLOCATE AND STORE INTO REQUEST BUFFER
615 7 ELSE
616 8 CALL EXEC TO WRITE ONLY FIRST ENTRY
617 9 SET LENGTHS IN REQUEST BUFFER TO 7 WORDS
618 0 CALL XMOV TO MOVE STORE REQUEST INTO REQUEST BUFFER
619 1 ENDBF
620 2 CALL XFER TO REQUEST MANAGER TO STORE X3STO
621 3 IF RETURN CODE IS NOT ZERO THEN
622 4 CALL XMSG TO WRITE SPACE ERROR
623 5 CALL EXEC TO RELEASE CLASS #
624 6 SET MASTER STATE TO ZERO
625 7 CALL XERTH TO RETURN TO EXEC **NO RETURN**
626 8 ENDBF
627 9 RETURN
628 0 1 END X3STO
CALL XXTMP

** Fortran Calling Sequence: **

** Inputs from common: **
- XE(12), SERPTR, XB(250), SELLEN, XB(251), SERTAB

** Outputs to common: **
- XE(6), SUBSTA, XB(249), SERNNO, XB(250), SELLEN,
  XB(251), SERTAB, XE(13), FLGTAB

** Common used: **
- EQUIVALENCE (XE(6), SUBSTA), (XE(12), SERPTR),
  (XB(19), SERPTR), (XB(20), REBUF),
  (XB(249), SERNNO), (XB(250), SELLEN),
  (XB(251), SERTAB), (XE(13), TMPTAB),
  (XE(13), FLGTAB)

** FDS routines used: **
- XREA, XMOV, XSTO

** RTE routines used: **
- EXEC
```
2 1  CD************
3 1  CD0
4 1  CD0
5 1  CD0
6 1  CD0
7 1  CD0
8 1  CD0
9 1  CD0
10 1  CD0
11 1  CD0
12 1  CD0
13 1  CD0
14 1  CD0
15 1  CD0
16 1  CD0
17 1  CD0
18 1  CD0
19 1  CD0
20 1  CD0
21 1  CD0
22 1  CD0
23 1  CD0
24 1  CD0
25 1  CD0
26 1  CD0
27 1  CD0
28 1  CD0
29 1  CD0
30 1  CD0
31 1  CD0
32 1  CD0
33 1  CD0
34 1  CD0
35 1  CD0
36 1  CD0
37 1  CD0
38 1  CD0
39 1  CD0
40 1  CD0
41 1  CD0
42 1  CD0
43 1  CD0
44 1  CD0
45 1  CD0
46 1  CD0
47 1  CD0
48 1  CD0
49 1  CD0
50 1  CD0
51 1  CD0
52 1  CD0
53 1  CD0
54 1  CD0
55 1  CD0
56 1  CD0
57 1  CD0
58 1  CD0
59 1  CD0
60 1  CD0

ASSGN - DATA ASSIGNMENT PROCESSOR
- SCHEDULED BY FDS

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

INPUTS FROM THE MANAGER:

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

INPUTS FROM THE INTERFACE TABLE:

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

OUTPUTS TO THE ANA:

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

INTERNAL VARIABLES:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIMENSION</th>
<th>START</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARMS</td>
<td>5</td>
<td>1</td>
<td>PARMS(1) = LU, PARMS(2) = DEBUG FLGS</td>
</tr>
<tr>
<td>TIKERS</td>
<td>32</td>
<td>2</td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STMIDE</td>
<td>1</td>
<td>33</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 SYMB. TAB ENTRY DEFINED</td>
</tr>
</tbody>
</table>
| SYMTAB | 12,81    | 41    | SYMBOL TABLE (WORDS 1-8 = TOK ENTRY OR
|              |          |       | APPLICABLE INFORMATION, WORDS 9-11 = |
|              |          |       | VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR |
|              |          |       | SUBSCRIPTED DATA ELEMENT) |
| SSTING | 247      | 1013   | RESULT STACK USED DURING POST- |
| RESULT | 4,35     | 1260   | FIX STRING EVALUATION (EACH EN- |
|        |          |        | TRY: WORDS 1-3 CONTAIN VALUE; |
|        |          |        | WORD 4 = DATA TYPE). DATA TYPE= |
|        |          |        | 1,2,3 FIXED DATA |
|        |          |        | -1 SYMBOL TABLE INDEX |
```
<table>
<thead>
<tr>
<th>ASSGN</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>1 CD4</th>
<th>2 CD4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTRNS</td>
<td>4,9</td>
<td>1400</td>
<td>-2 DISPLACEMENT</td>
<td>-3 CHARACTER STRING INDEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCRATCH</td>
<td>4,9</td>
<td>1436</td>
<td>SCRATCH AREA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGST</td>
<td>1</td>
<td>1472</td>
<td>MANAGER REQUEST FOR XPRES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPINFO</td>
<td>7</td>
<td>1480</td>
<td>CONTENTS OF FNCBL OR SYNTAX FOR FUNCTION OR MATHEMATICAL OPERATIONS BEING EVALUATED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMWORDS</td>
<td>1</td>
<td>1487</td>
<td>NUMBER OF WORDS TO BE STORED IN OBJECT DATA ELEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATYPES</td>
<td>9</td>
<td>1488</td>
<td>DATA TYPES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDLTPY</td>
<td>1</td>
<td>1497</td>
<td>RESULT STACK POINTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPTR</td>
<td>1</td>
<td>1498</td>
<td>POST-FIX STRING (POLISH) POINTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLSREN</td>
<td>1</td>
<td>1499</td>
<td>XPRES OPTION WORD FOR QUEUE REQUEST AND CLOSE BUFFER - NO DATA TRANSFERED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLSTRN</td>
<td>1</td>
<td>1500</td>
<td>XPRES OPTION WORD FOR QUEUE REQUEST, CLOSE BUFFER AND TRANSFER DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAPWDS</td>
<td>9</td>
<td>1501</td>
<td>NUMBER OF WORDS PER LOGICAL UNIT OF DATA FOR EACH DATA TYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECLAS</td>
<td>1</td>
<td>1510</td>
<td>DATA ELEMENT CLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNGSTR</td>
<td>4,4</td>
<td>1511</td>
<td>BASIC RANGE, END RANGE, INCREMENT AND SYMBOL INDEX FOR EACH RANGE SPECIFICATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POLISH</td>
<td>161</td>
<td>1527</td>
<td>POST-FIX REPRESENTATION OF EXPRESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNTAX</td>
<td>7,40</td>
<td>1688</td>
<td>SYNTAX TABLE FOR VALIDITY TESTS ON EXPRESSION (SEE BELOW)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNCBL</td>
<td>7,36</td>
<td>1968</td>
<td>FUNCTION TABLE CONTAINING DATA REQUIREMENTS FOR EACH FUNCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRBUFF</td>
<td>64</td>
<td>2220</td>
<td>XPRES BUFFER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

TABLE DEFINITIONS:
1 CD4

**SYNTAX - 7x40 TABLE CONTAINING INFORMATION FOR PROCESSING EACH TOKEN**

<table>
<thead>
<tr>
<th>CD4</th>
<th>TOKEN</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>TOKEN #PRNDS-1</th>
<th>DATA TYPE</th>
<th>PRECEDENT</th>
<th>VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD4</td>
<td>(EOS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>INTEGER</td>
<td>-</td>
<td>2</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>REAL</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>DOUBLE</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>MINUS</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>NAME</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>EOF</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>=</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>&lt;</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>&gt;</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>(</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>)</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>[</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
<tr>
<td>CD4</td>
<td>]</td>
<td>-</td>
<td>4</td>
<td></td>
<td>4</td>
<td>OP(LBRKT=</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:**
- **OP** = Operator
- **LB** = Left Bracket
- **RB** = Right Bracket
- **CN** = Constant
- **SAME** = Same Type
- **FUNCTION** = Function

REPRODUCTION OF THE ORIGINAL PAGE IS POOR
Routines referenced: RRPAR, XRGET, XRMOV, XI.S1, XIIS2, XPIXIT

Backus-Naur Language Definition

<assignment> ::= <replacement> <range> //
<evaluation> ::= <range>

<replacement> ::= <non-numeric de> = <non-numeric de> //
<replacement> ::= <non-numeric de> = "character string" //
<variable> ::= free data element
<variable> ::= free data element<subscript list> //
<expression> ::= <variable> = <expression> //
<expression> ::= free data element = <expression> //
<expression> ::= free data element<subscript list> = <expression>

<variable> ::= fixed data element //
<variable> ::= fixed data element<subscript list>

<subscript list> ::= <subscript list>, <expression> //
<subscript list> ::= <expression>
<expression> ::= <expression> <additive operator> <term> //
<expression> ::= <expression> <additive operator> <term> //
<expression> ::= unary operator <term> //
<expression> ::= unary operator <factor> //
<factor> ::= <power>**<power> //
<factor> ::= <power>
<power> ::= <expression> //
<power> ::= <expression> //
<operand> ::= <additive operator> //
<operand> ::= <additive operator> //
<operand> ::= * //
<operand> ::= function name <lb> <function list> <rb> //
<operand> ::= variable //
<operand> ::= constant
<operand> ::= <constant>
<operand> ::= left bracket
<operand> ::= left bracket
<operand> ::= right bracket
<operand> ::= right bracket
<function list> ::= <function list>, <expression> //
<function list> ::= <expression> //
BEGIN ASSGN
CALL XIPST TO BUILD POST-FIX STRING
CALL XPS2 TO EVALUATE EXPRESSION AND STORE VALUE(S)
CALL XPSIT TO EXIT PROFESSOR
END ASSGN
DBDSP - DATA BOX DISPLAY PROCESSOR

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:

XIDIM - READS AND EDITS INTERFACE TABLE
XIDP1 - READS ORDE A, VALIDATES NAMES IN INTERFACE TABLE AGAINST NAMES IN DATABOX SCAN, WITH SUBROUTINE XIDMK, IT DEVELOPS CONSTRAINT MASKS FOR ARRAYS
XIDP2 - PROMPTS USER IF REQUIRED AND OUTPUTS REQUESTED PAGE ARRAYS OF UP TO TWO VARIABLES TO THE SPECIFIED LU DEIVCE WITH SUBROUTINE XIDOT

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

INPUTS TO DBDSP FROM INTERFACE TABLE:
- DATABOX - DATA BOX FILE NAME
- NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER
- KEP - DISPLAY VARIABLE SCALE LIST SET UP BY USER
- VODEF - CONSTRAINT VARIABLE DEFINITION LIST

INPUTS TO DBDSP FROM ORDE FILE:
- RECORD 1
  (1) - NAME OF FDS PROCESSOR CREATING FILE
  (4) - INTERFACE TABLE VARIABLE NAME FOR THIS FILE
  (7) - NAME OF FDS PROCESSOR UPDATING FILE
  (10) - INTERFACE TABLE VARIABLE NAME FOR THIS U-DATE

- RECORD 2
  (1) - NO OF ENTRIES IN SUMMARY TABLE
  (2) - X SCAN VARIABLE (6 CHAR)
  (3) - XFIRST SUBSCRIPT (INT OR ZERO)
  (4) - XSECOND SUBSCRIPT (INT OR ZERO)
  (7) - X UNITS (6 CHAR)
  (10) - X CENTROID (REAL)
(12) - X INC.(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) - Y INC。(REAL)
(27) - Y NUMBER OF STEPS (INTEGER 1-5)

DEPENDANT VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

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

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

CD 4 COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPRM
CD 4 DBX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
CD 4 DATABU - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
CD 4 ID1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
CD 4 ID2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR
CD 4 ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)
CD 4 IISCM1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD 4 IISCM2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
CD 4 IFFBU - NAME OF COMMON AREA USED FOR INTERFACE TABLE
CD 4 ITSCM1 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
CD 4 ITSCM2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
CD 4 LENBUF - LENGTH OF COMBUF IN 16-BIT WORDS - XPRM
CD 4 LENV - LENGTH IN WORDS OF CHAR STRING USED FOR USER PROMPT - XPRM
CD 4 M - ARRAY CONTAINING MASKS FOR UP TO 9 CONSTRAINTS (4 X 121 SIZE)
CD 4 NAME - NAME LIST FOR VARIABLES SCANNED BY SCANENDSCM
CD 4 NCR - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
CD 4 NCRL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
CD 4 NCVL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
CD 4 NDISP V - LIST OF DEP DISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16PR)
CD 4 NVAVL - NUMBER OF DEP DISP VAR IN NOVAR LST (INTEGER)
CD 4 NVB - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR OF/P
CD 4 NWE - SET OF INDICATORS FOR CONSTRAINTS VIOLATED-D/NOT D
CD 4 NSTEPS - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
CD 4 XCO - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
CD 4 XCOORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
CD 4 XSV - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
397 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
398 1 CD 4 YCORD - LIST OF X VAR VALUES FOR Y COORDINATES (1-11 REAL)
399 1 CD 4 YSCAN - 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 NAMVAL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN
403 1 CD 4 SUMTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
404 1 CD 4 PARDS - COMMUNICATION BUFFER FOR RPAR - LU, USER ID, FLAGS
405 1 CD 4 LU - LOGICAL UNIT # FOR XPRM CALLING SEQUENCE - USER LOCATM
406 1 CD 4 LUOUP - DDDSP 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 =O PROMPT ; SELECT NOT O RUN ALL DISPLAYS TO O/P
410 1 CD 4 SELECT - SELECT =O PROMPT ; SELECT NOT O RUN ALL DISPLAYS TO O/P
411 1 CD 4 CARTGC - CARTRIDGE USED TO LOCATE DATA BOX
412 1 CD 4 CVARGE -
413 1 CD 5 ******
414 1 CD 5 ******
415 1 CD 5 ******
416 1 CD 5 ******
417 1 CD 5 XPRM, XELBS, XPXiT, RMPAR
418 1 CD 5 ******
419 1 CD 5 ******
420 1 CD 5 ******
421 1 CD 5 ******
DEFIN DEFINES AND MANAGES DATA ELEMENTS IN THE ANA THAT ARE SPECIFIED BY THE PARAMETER KEYWORD DEFINE. IF THE DATA ELEMENT ALREADY EXISTS, IT IS DELETED AND REALLOCATED. DATA ELEMENTS ARE INITIALIZED TO ZERO (CHARACTER STRINGS TO BLANKS).

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

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

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

INTERNAL VARIABLES:
- INBUF - INTERFACE TABLE HEADER
- ISLENG - LENGTH OF SYMBOLIC STRING
- IZTOK - POSITION WITHIN THE SYMBOLIC STRING
- MWARN - BUFFER FOR VALID NAMES TO BE ALLOCATED
- MBUFF - BUFFER AREA FOR XGET AND XPER USE
- MEY - NUMBER OF ENTRIES IN THIS ALA REQUEST
- MNAM - NUMBER OF NAMES IN SYMBOLIC STRING
- NEXTHN - TOKEN POSITION FOR NEXT NAME
- NAM - STRING - SYMBOLIC STRING INPUT TO DEFINE

EXTERNAL ROUTINES USED:
- EXEC, IAMP, KCVT, KMPAR, XPIET,
- XPREX, XPXIT, XUP06, XIOFT, XIMSG
1 BEGIN DEFIN
2 CALL RMAR TO GET LU AND DEBUG FLAGS
3 CALL APGET TO RETRIEVE SYMBOLIC STRING
4 GET STRING LENGTH FROM INTERFACE TABLE HEADER
5 START SEARCH WHILE TOKEN-POSITION .LT. STRING-LENGTH, OR
6 WHILE CURRENT-TOKEN .NE. END-OF-STRING
7 PERFORM ZDPM TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
8 PERFORM ZDPM TO PROCESS THE CURRENT NAME
9 EXIT IF THERE WAS AN ERROR RETURN FROM ZDPM
10 CALL ZNDN TO DISPLAY SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
11 SET PROCESSOR RETURN CODE TO ABEND
12 ORElse
13 INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
14 ENDDO
15 SET PROCESSOR RETURN FOR NORMAL EXIT
16 ENDSNAR
17 SET OPTION SO XPREN WILL DO A QUEUE REQUEST
18 DO FOR NUMBER OF REQUESTS IN REQUEST BUFFER (MANARR)
19 IF THIS IS LAST REQUEST, THEN
20 SET OPTION TO CLOSE REQUEST BUFFER
21 ENDIF
22 COMPUTE INDEX TO THIS REQUEST
23 CALL XPREN TO QUEUE THIS REQUEST
24 ENDDO
25 CALL XPRT TO RETURN TO THE MANAGER
26 1 END DEFIN
527 1 BEGIN XIDPM
528 2 SET IDIM AND JDIM TO 1
529 3 IF TOKEN IS NOT A NAME THEN
530 4 SET ERROR CODE
531 5 ELSE
532 6 MOVE NAME INTO REQUEST
533 7 INCREMENT TO NEXT TOKEN
534 8 IF THERE ARE SUBSCRIPTS (TOKEN IS A LEFT PAREN) THEN
535 9 INCREMENT TO NEXT TOKEN
536 10 IF TOKEN IS NOT AN INTEGER OR
537 11 TOKEN IS NOT ZERO THEN
538 12 CALL XIMSG TO DISPLAY ERROR "INVALID IDIM"
539 13 EXIT TO :PERR1:
540 14 ENDIF
541 15 SET IDIM TO THIS TOKEN
542 16 INCREMENT TO NEXT TOKEN
543 17 IF THERE ARE TWO SUBSCRIPTS (TOKEN IS A COMMA) THEN
544 18 INCREMENT TO NEXT TOKEN
545 19 IF TOKEN IS NOT AN INTEGER OR
546 20 TOKEN IS NOT ZERO THEN
547 21 CALL XIMSG TO DISPLAY ERROR "INVALID JOIN"
548 22 EXIT TO :PERR1:
549 23 ENDIF
550 24 SET JDIM TO THIS TOKEN
551 25 INCREMENT TO NEXT TOKEN
552 26 ENIF
553 27 IF TOKEN IS NOT A RIGHT PAREN THEN
554 28 CALL XIMSG TO DISPLAY ERROR "INVALID SUBSCRIPT DELIMITER"
555 29 EXIT TO :PERR1:
556 30 ENDIF
557 31 INCREMENT TO NEXT TOKEN
558 32 IF TOKEN IS NOT A NAME THEN
559 33 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
560 34 EXIT TO :PERR1:
561 35 ENIF
562 36 INCREMENT TO NEXT TOKEN
563 37 IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
564 38 CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
565 39 EXIT TO :PERR1:
566 40 ENDIF
567 41 INCREMENT TO NEXT TOKEN
568 42 STARTSEARCH FOR ALL VALID TYPES
569 43 EXIT IF TYPE MATCHES THE TYPE IN THE SYMBOLIC STRING
570 44 SET TYPE AND CLASS IN REQUEST
571 45 COMPUTE SIZE AS IDIM + JDIM + LENGTH OF TYPE
572 46 IF SIZE IS TOO LARGE (>1200 WORDS) THEN
573 47 CALL XIMSG TO DISPLAY ERROR "DATA ELEMENT IS TOO LARGE"
574 48 EXIT TO :PERR1:
575 49 ENDIF
576 50 ENDLOOP
577 51 CALL XIMSG TO DISPLAY ERROR "INVALID OR MISSING TYPE FIELD"
578 52 EXIT TO :PERR1:
579 53 ENDSEARCH
580 54 INCREMENT TO NEXT TOKEN
581 55 IF TOKEN IS NOT END OF TYPE FIELD SLASH OR
582 56 NEXT TOKEN IS NOT A COMMA THEN
583 57 CALL XIMSG TO DISPLAY WARNING "TYPE NOT TERMINATED BY A SLASH"
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:

XXRET - RETURN CODE TO MANAGER

OUTPUTS TO AWA:

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

RTE ROUTINES USED:

CLOSE, EXEC, FLOAT, KCVT, POINT,
READ, RNPAR, WRIT

FDS ROUTINES USED:

XPXIT, XREX, XRMOV, XUDBG, XVPW,
XIFCL, XIMSB
641 1 BEGIN ENDS
642 2 CALL RMPAR TO GET INPUTS FROM MANAGER
643 3 SAVE REQUEST TO RETRIEVE $SCMTB FROM $AMT
644 4 CALL XPVAM TO REQUEST $AMT MANAGEMENT
645 5 ERREXIT IF $SCMTB NOT FOUND TO :ERR4:
646 6 CALL EXEC TO READ IN $SCMTB
647 7 SAVE REQUEST TO RETRIEVE SMTAB IN RQB
648 8 CALL XPVAM TO REQUEST $AMT MANAGEMENT
649 9 ERREXIT IF NOT FOUND TO :ERR4:
650 2 CALL EXEC TO READ IN SMTAB
651 3 IF SUMMARY TABLE IS LARGER THAN 32 ENTRIES THEN
652 4 SET SIZE OF SMTAB TO 32 ENTRIES
653 5 ENDIF
654 6 CALL WRFTF TO WRITE SMTAB TO DATBOX
655 7 ERREXIT IF WRFTF ERROR TO :ERR4
656 8 IF THERE IS 1 VARIABLE AND XRMT IS CENTROID OR
657 9 THERE ARE 2 VARIABLES AND XRMT IS CENTROID AND YCUR IS CENTROID THEN
658 10 CALL READ TO READ HEADER RECORD
659 11 ERREXIT IF READF ERROR TO :ERR4.
660 12 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
661 13 CALL WRFTF TO WRITE UPDATED HEADER
662 14 ERREXIT IF WRFTF ERROR TO :ERR4:
663 15 CALL CLOSE TO CLOSE DATBOX
664 16 SAVE REQUEST TO DELETE ABS $SCMTB IN RQB
665 17 IF THERE ARE REMAINING SCANS I; $SCMTB THEN
666 18 CALL EXEC TO WRITE REMAINING $SCMTB
667 19 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCMTB
668 20 ENDIF
669 21 SET RETURN PARAMETER TO NORMAL RETURN
670 2 ELSE
671 3 PERFORM SETTY
672 4 ENDIF
673 5 CALL XPVAM TO REQUEST $AMT MANAGEMENT
674 6 CALL XPVMT TO TERMINATE WITH RETURN PARAMETERS
675 7 EXIT ENDS
676 8 :ERR4;
677 9 CALL XIMSG TO DISPLAY ERROR
678 10 CALL XPMT TO ABEND PROCESSOR
679 1 END ENDS
BEGIN SETXY
IF XCUR IS END STEP THEN
  IF THERE IS 1 VARIABLE THEN
    SET XCUR TO CENTER
    CALL POSN TO POSITION FILE TO CENTERED RECORD
    ERREXIT IF POSN ERROR TO :ERR4:
  ELSE
    IF YCUR IS END STEP THEN
      SET YCUR TO CENTER
      SET YCUR TO CENTER
      CALL POSN TO POSITION FILE TO CENTERED RECORD
      ERREXIT IF POSN ERROR TO :ERR4:
    ELSE
      SET XCUR TO (-XSTEP)
      IF XSTEP IS ZERO AND Y IS CENTERED THEN
        INCREMENT YCUR BY 1
        CALL WRITF TO WRITE DUMMY AS CENTERED RECORD
        ERREXIT IF WRITE ERROR TO :ERR4:
      ENDIF
      COMPUTE X AS (XCENT + XINC + FLOAT(XCUR))
      COMPUTE Y AS (YCENT + YINC + FLOAT(YCUR))
    ENDIF
  ENDIF
ELSE
  INCREMENT XCUR BY 1
  IF THERE IS 1 VARIABLE AND X IS THE CENTERED OR
  THERE ARE 2 VARIABLES AND X IS THE CENTERED AND Y IS THE CENTERED THEN
  INCREMENT XCUR BY 1
  CALL WRITF TO WRITE DUMMY AS CENTERED RECORD
  ERREXIT IF WRITE ERROR TO :ERR4:
ENDIF
COMPUTE X AS (XCENT + XINC + FLOAT(XCUR))
COMPUTE Y AS (YCENT + YINC + FLOAT(YCUR))
ENDIF
ENDIF
END IF SCAN VARIABLES
END}

END IF SCAN VARIABLES
(18) XSTEP # STEPS OF X
(19) XCUR CURRENT X STEP NUMBER
(20) YSCAN NAME OF Y VARIABLE
(21) X+YPL DISPLACEMENT FOR Y
(22) YCENT CENTROID OF Y
(23) YINCRC INCREMENT FOR T
(24) YSTEP # STEPS OF Y
(25) YCUR CURRENT Y STEP NUMBER
(26) IDCB 144 WORD DCB FOR DATBOX

RTE SUBROUTINES USED:
CLOSE, CREAT, EXEC, FLOAT, IAND,
KCVT, MOD, POSNT, PURGE, RMPAR,
WRITF

FDS SUBROUTINES USED:
XPATH, XPSET, XPPUT, XPYIT, XRCPA,
XREAT, XREMOV, XRFFM, XRNUM, XUDBG,
XPAW, XMSG
BEGIN SCAN
CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XPFET TO GET PROCID AND # SCAN VARIABLES
ERREXIT IF # SCAN VARIABLES < 8 OR > 2 TO :ERR3;
GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
ERREXIT IF SUMTAB IS A LITERAL TO :ERR3;
ERREXIT IF DISPLACEMENT IS NOT AN ELEMENT BOUNDARY TO :ERR3;
GET DATA BOX NAME FROM INTERFACE TABLE
DO FOR # SCAN VARIABLES
CALL XPAT TO GET NAME AND DISPLACEMENT
COMPUTE SUBSCRIPTS FROM DISPLACEMENT AND IDIN
CALL XPFET TO GET UNITS, CENTROID, IMER, # STEPS
ERREXIT IT # STEPS < 0 OR > 5 TO :ERR3:
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR $SETB AND $SCNTB
CALL XPRM TO REQUEST AWA MANAGEMENT
CALL EXEC TO READ IN $SETB
IF $SCNTB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF $SCNTB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN $SCNTB
ERREXIT IF # SCANS > MAXIMUM ALLOWED (4) TO :ERR3:
ERREXIT IF THIS DATABASE NAME IS ALREADY IN USE TO :ERR3:
ENDIF
ERREXIT IF THIS IS THE LAST ENTRY IN $SETB TO :ERR3:
GET THE SEQUENCE NUMBER OF THIS SCAN FROM $SETB
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH $SETB FOR THE SEQUENCE NUMBER
ERREXIT IF THIS SCAN IS THE LAST ENTRY IN $SETB TO :ERR3:
IF THIS IS A SEMI-OVERRIDE (2 PROCESSOR NAMES NOT EQUAL) THEN
SET RESET NUMBER TO THIS ENTRY SEQUENCE NUMBER
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
ELSE
SET RESET NUMBER TO NEXT ENTRY SEQUENCE NUMBER
ENDIF
END SCAN
SCAM
873 2 ERREXIT IF NO AWS SPACE TO :ERR1:
874 2 CALL ARXIT TO EXIT NORMALLY
875 1 EXIT SCAN
876 2 :ERR1:  
877 2 IF THERE ARE MORE ACTIVE SCANS (#SCANS > 0) THEN
878 3 CALL EXEC TO READ IN NEW &SCNTB
879 3 CALL EXEC TO WRITE OUT ORIGINAL &SCNTB
880 3 SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL &SCNTB
881 2 ENDFI
882 2 :ERR2:  
883 2 CALL CLOSE TO CLOSE DATBOX
884 2 CALL PURGE TO PURGE DATBOX
885 2 SET VALUE FOR XPUT
886 2 SAVE REQUEST TO DELETE DATBOX FROM AWS
887 2 CALL XPPAW TO REQUEST AWS MANAGEMENT
888 2 :ERR3:  
889 2 CALL XMSG TO DISPLAY ERROR
890 2 CALL XPXIT TO ABEND SCAN
891 1 END SCAN
1 BEGIN XICHR
2   SET # WORDS TO BE STORED IN OBJECT = # WORDS/ELEMENT FOR OBJECT
3
4   CASE (RESULT DATA TYPE) :
5     :CHRSTR, :CHRRE, :CHRRE, :FIXERR, :
6
7     :FIXERR, :FIXERR:
8
9   :CHRSTR:
10      DETERMINE # WORDS IN CHARACTER STRING
11      MOVE CHARACTER STRING TO RESULT LOCATION
12      IF # WORDS IN STRING < # WORDS/ELEMENT FOR OBJECT, THEN
13      BLANK FILL AFTER CHARACTER STRING
14
15   :CHRRE:
16      IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
17      POP DISPLACEMENT FROM RESULT STACK
18      ELSE
19      SET DISPLACEMENT = 0
20
21     ENDIF
22
23     POP RESULT OPERAND FROM STACK
24      IF RESULT OPERAND IS FREE, THEN
25      SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
26      ELSE CHARACTER = CHARACTER
27      SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
28
29     ENDIF
30      CALL XRER TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
31      IF # WORDS RETRIEVED IS < # WORDS TO BE STORED, THEN
32      BLANK FILL REMAINING DATA
33
34     ENDIF
35
36     END CASE
37
38     1 EXIT XICHR
39
40     2 :FIXERR:
41     2   SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO NUMERICAL DATA"
42     2
43     2   CALL XING TO OUTPUT MESSAGE TO USER
44     2   CALL XLISS TO LIST SYMBOLIC STRING
45     2   CALL XPXIT TO EXIT PROCESSOR
46     1 END XICHR
**XZDIN**

**XZDIN - DBSP INPUT PROCESSOR**

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

**XZDIN**

**INPUT**

**ALL INPUT COMES FROM THE 26 INTERFACE TABLE ARGUMENTS**

**OUTPUT**

**COMMON**

**NCVARL, NCRELL, CVALE, NDBVARL, NDBVRL, NC**

**NOTES**

**USES ROUTINES**

**EXEC**

**XPGET**

**XPIT**

**XRMV**

**XZLS**

**REFERENCES**

**ARE POOR**
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 INDX TO START
1085 2 DO UNTIL INDEX .GE. END
1086 3 IF ARRAY(INDEX) .EQ. TOKEN, THEN
1087 4 SET END TO INDEX
1088 3 ELSE
1089 4 CASE ARRAY(INDEX)+1 ()
1090 5 :ADD1:, 0 END OF STRING
1091 5 :ADD2:, 1 INTEGER
1092 5 :ADD3:, 2 REAL
1093 5 :ADD4:, 3 DOUBLE
1094 5 :ADD1:, 4 UNKNOWN
1095 5 :ADD4:, 5 NAME
1096 5 :ADD1:, 6 UNKNOWN
1097 5 :ADD1:, 7 UNKNOWN
1098 5 :CALST:, 8 CHARACTER STRING
1099 5 :ADD1:, 9 UNKNOWN
1100 5 :ADD1:, 10 +
1101 5 :ADD1:, 11 -
1102 5 :ADD1:, 12 *
1103 5 :ADD1:, 13 /
1104 5 :ADD1:, 14 <
1105 5 :ADD1:, 15 >
1106 5 :ADD1:, 16 =
1107 5 :ADD1:, 17 @
1108 5 :ADD1:, 18 =
1109 5 :ADD1:, 19 %
1110 5 :ADD1:, 20 ?
1111 5 :ADD1:, 21 !
1112 5 :ADD1:, 22 ,
1113 5 :ADD2:, 23 START OF SYMBOLIC STRING
1114 5 :ADD1:, 24 END OF STRING
1115 5 :ADD1:, 25 %
1116 5  :ADD1:  26 BACK SLASH
1117 5  :ADD1:  27 $
1118 5  :ADD1:  28 .
1119 5  :ADD1:  29 OPEN BRACKET
1120 5  :ADD1:  30 CLOSE BRACKET
1121 5  :ADD2:  31 REPEAT
1122 5  :ADD1:  32 ,
1123 5  :ADD1:  )  33 ;
1124 5  :ADD1:
1125 5  INDEX=INDEX+1
1126 5  :ADD2:
1127 5  INDEX=INDEX+2
1128 5  :ADD3:
1129 5  INDEX=INDEX+3
1130 5  :ADD4:
1131 5  INDEX=INDEX+4
1132 5  :CALST:
1133 5  INDEX+INDEX+2+(ARRAY(INDEX+1)+1)/2
1134 4  ENDCASE
1135 3  ENDF
1136 2  ENDDO
1137 1  END
XZDP1 - DATA BOX DISPLAY OVERLAY - PREPARES DATA FOR DISPLAY
- SCHEDULED BY DBDSP

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

INPUTS FROM THE DATA BOX

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

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

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

SUMMARY TABLE RECORDS
- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH
1306 1 CD-------
1307 1 CD0     XIDMK - CONSTRAINT MASKING ROUTINE
1308 1 CD0     XIDMK IS CALLED ONCE BY XIDPI TO BUILD THE CONSTRAINT
1309 1 CD0     VIOLATION MASKS FOR ALL VALUE POSITIONS OF THE DISPLAY GRID.
1310 1 CD-------
1311 1 CD1     INPUTS
1312 1 CD1     COMMON - ATABLE, NCVARL, NCRELL, CVALUE
1313 1 CD1     OUTPUTS
1314 1 CD1     COMMON - MSKERR, MASK1, MASK2
1315 1 CD1     SAVE
1316 1 CD-------
1317 1 CD2     NOTES
1318 1 CD2     USES Routines
1319 1 CD2     XRCFR
1320 1 CD2     XRMV
1321 1 CD2     XRESET
1322 1 CD-------
1323 1 CD-------
1324 1 CD-------
1325 1 CD-------
1326 1 CD-------
1327 1 CD-------
1328 1 CD-------
1329 1 CD-------
1330 1 CD-------
1331 1 CD-------
1332 1 CD-------
1333 1 CD-------
1334 1 CD-------
1335 1 CD-------
1336 1 CD-------
1337 1 CD-------
1338 1 CD-------
1402 1 CD *******
1403 1 CD 0  
1404 1 CD 0  XZDP2 - DATA BOX DISPLAY OVERLAY - PROMPTS USER FOR DISPLAY
1405 1 CD 0  DESIRED, THEN FORMATS AND DISPLAYS DATA ACCORDINGLY
1406 1 CD 0  
1407 1 CD 0  SCHEDULED BY DDSP
1408 1 CD 0  
1409 1 CD 1  *******
1410 1 CD 1  XZDP2 Issues PROMPTS TO THE USER REQUESTING PAGE NUMBER.
1411 1 CD 1  (STARTING ROW NUMBER, NUMBER OF ROWS, AND DESIRED
1412 1 CD 1  LU-) SELECT OPTION CAN BE SET WHEN ENTERING
1413 1 CD 1  DDSP IN THE INTERFACE TABLE TO OUTPUT ALL PAGES
1414 1 CD 1  TO NAMED LU.
1415 1 CD 1  XZDP2 UTILIZES SUBROUTINE XZDOT TO ACTUALLY FORMAT
1416 1 CD 1  AND DISPLAY THE SCAN RESULTS.
1417 1 CD 1  
1418 1 CD 1  *******
1419 1 CD 1  INPUTS TO XZDP2 FROM XZDP1
1420 1 CD 1  
1421 1 CD 2  MASK TABLES CONTAINING THE CONSTRAINT MASKS FOR EACH
1422 1 CD 2  CONSTRAINT WHICH WAS VIOLATED AND INDICATION OF
1423 1 CD 2  WHETHER OR NOT ANY CONSTRAINT WAS VIOLATED FOR EACH
1424 1 CD 2  ARRAY COORDINATE
1425 1 CD 2  
1426 1 CD 2  LIST OF VALID DISPLAY DEPENDANT VARIABLES FOR EACH
1427 1 CD 2  PAGE (UP TO 16 PAGES)
1428 1 CD 2  
1429 1 CD 2  DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES
1430 1 CD 2  
1431 1 CD 2  *******
1432 1 CD 2  OUTPUT FROM XZDP2
1433 1 CD 2  
1434 1 CD 3  DISPLAY FORMAT SHOWN IN DOCUMENTATION IS SENT TO
1435 1 CD 3  NAME LU DEVI CE
1436 1 CD 3  
1437 1 CD 3  *******
1438 1 CD 3  OUTPUT FROM XZDP2
1439 1 CD 3  
1440 1 CD 4  COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XROM
1441 1 CD 4  
1442 1 CD 4  DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DDSP
1443 1 CD 4  
1444 1 CD 4  DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
1445 1 CD 4  
1446 1 CD 4  IDV1 - Pointer to DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
1447 1 CD 4  IDV2 - Pointer to DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.
1448 1 CD 4  
1449 1 CD 4  ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XZMSK (MAX OF 8)
1450 1 CD 4  ISAVE1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O )
1451 1 CD 4  ISAVE2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR O)
1452 1 CD 4  
1453 1 CD 4  IXSCN1 - FIRST SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
1454 1 CD 4  IXSCN2 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR O)
1455 1 CD 4  
1456 1 CD 4  LENBF - LENGTH OF COMBUF IN 16-BIT WORDS - XROM
1457 1 CD 4  LENST - LENGTH IN WDS OF CHAR STRING USED FOR USER PROMPT - XROM
1458 1 CD 4  
1459 1 CD 4  NAMVAR - NAME LIST FOR VARIABLES SCANNED BY SCAN-ENDSCN
1460 1 CD 4  
1461 1 CD 4  NC - NUMBER OF CONSTRAINTS INPUT BY USER (INT/GER)
1462 1 CD 4  
1463 1 CD 4  NCOLL - LIST OF CONSTRAINT RELATIONS INPUT BY USER
1464 1 CD 4  NCOLL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32, MAX)
**XDOT - DISPLAY OUTPUT ROUTINE**

XDOT IS CALLED ONCE BY XZDP2 TO OUTPUT THE DISPLAY PAGE

**INPUTS**

- NCVARL
- NDVARL
- NCRELL
- CVALUE
- K1
- K2
- ZTABLE
- ATABLE
- NDVURL
- NPAGE
- DIBX
- XSCNM1
- XSCNM2
- XUMTS
- YUMTS
- IXSCN1
- IXSCN2
- IYSCN1
- IYSCN2

**OUTPUTS**

XDOT

**NONE**

**NOTES**

USES RELTIAES

- XRDPSV
- XEC
- XZISP
- XRPCK
- XRUPK
- XR16
- XRE16
- IABS

**REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR**
1591 1 CD******
1592 1 CD0 FORTRAN CALLING PROCEDURE
1593 1 CD0 CALL XZ15P (STRING, LEN)
1594 1 CD0 CALL XZ15P (STRING, LEN)
1595 1 CD0 CALL XZ15P (STRING, LEN)
1596 1 CD0 CALL XZ15P (STRING, LEN)
1597 1 CD1 XZ15P REMOVES DUPLICATE (I.E. CONSECUTIVE) BLANKS FROM
1598 1 CD1 A CHARACTER STRING AND FILLS THE VACATED TRAILING WORDS
1599 1 CD1 WITH BLANKS
1600 1 CD1 WITH BLANKS
1601 1 CD1 WITH BLANKS
1602 1 CD******
1603 1 CD2 1 CD2 1 CD2
1604 1 CD2 INPUT
1605 1 CD2 CALLING SEQUENCE
1606 1 CD2 CALLING SEQUENCE
1607 1 CD2 STRING - INPUT CHARACTER STRING
1608 1 CD2 LEM - NUMBER OF WORDS IN STRING
1609 1 CD2 LEM - NUMBER OF WORDS IN STRING
1610 1 CD2 LEM - NUMBER OF WORDS IN STRING
1611 1 CD******
1612 1 CD3 OUTPUT
1613 1 CD3 CALLING SEQUENCE
1614 1 CD3 CALLING SEQUENCE
1615 1 CD3 STRING - CHARACTER STRING WITH ALL FIELDS OF CONSECUTIVE
1616 1 CD3 BLANKS REDUCED TO 1 BLANK AND TRAILING BLANK FILL
1617 1 CD3 STRING REDUCED TO 1 BLANK AND TRAILING BLANK FILL
1618 1 CD3 LEM - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
1619 1 CD3 LEM - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
1620 1 CD3 LEM - NO. OF WORDS IN STRING PRIOR TO TRAILING BLANK FILL
1621 1 CD******
1622 1 CD4 NOTES
1623 1 CD4 USES ROUTINES
1624 1 CD4 USES ROUTINES
1625 1 CD4 USES ROUTINES
1626 1 CD4 XZ15P
1627 1 CD4 XZ15P
1628 1 CD4 XZ15P
1629 1 CD******
1630 1 BEGIN XZ15P
1631 2 CALL XZ15P TO REMOVE DUPLICATE BLANKS FROM STRING
1632 2 DC WHILE THERE ARE TRAILING WORDS IN STRING
1633 2 SET THIS TRAILING WORD TO BLANKS
1634 2 END DO
1635 1 END XZ15P
**REPRODUCIBILITY OF THE ORIGINAL PAGE IS PAGE**
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 - LU, SSTRNG, OPRNDS, OPIINFO

OUTPUTS TO ASGCOM
RESULT, RSLTPT, OPRNDS

EXTERNAL REFERENCES
FDX - XPXIT, XRMOV, XILSS, XIZMG

RTE - ABS, AZMT, ALOG, ALGOT, AMOD, ATAN, ATAN2, COS, DABS, DATAN,
DATN2, DBLE, DCOS, DINT, DEXP, DLOG, DLGOT, DMOD, DSIN, DSIN, DSQRT,
DI, DFLAT, DABS, DINT, DFIX, DSIGN, MOD, OVF, SIGN, SIN, SMGE, SGRT,
TAN, TANN

EXTERNAL REFERENCES

1711 1 BEGIN XIFNC
1712 2 PERFORM FUNCTION INDICATED BY ENTRY
1713 2 EREXIT TO :OVER: IF OVERFLOW OR UNDERFLOW IS INDICATED
1714 2 PUSH RESULT AND TYPE ONTO RESULT STACK
1715 1 EXIT XIFNC
1716 2 :OVER:
1717 2 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
1718 2 CALL XIMSG TO OUTPUT MESSAGE TO USER
1719 2 CALL XLISP TO LIST SYMOLIC STRING
1720 2 CALL XEXIT TO EXIT PROCESSOR
1721 1 END XIFNC
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1723</td>
<td>1 CD0 FORTRAN CALLING PROCEDURE:</td>
</tr>
<tr>
<td>1724</td>
<td>1 CD0</td>
</tr>
<tr>
<td>1725</td>
<td>1 CD0 CALL XIFRE</td>
</tr>
<tr>
<td>1726</td>
<td>1 CD0</td>
</tr>
<tr>
<td>1727</td>
<td>1 CD0</td>
</tr>
<tr>
<td>1728</td>
<td>1 CD0 XIFRE IS USED BY THE ASSGN ROUTINE XIPS2 TO PROCESS DATA ASSIGNMENTS FOR FREE-TYPE OBJECT DATA ELEMENTS</td>
</tr>
<tr>
<td>1729</td>
<td>1 CD1</td>
</tr>
<tr>
<td>1730</td>
<td>1 CD1</td>
</tr>
<tr>
<td>1731</td>
<td>1 CD1</td>
</tr>
<tr>
<td>1732</td>
<td>1 CD1</td>
</tr>
<tr>
<td>1733</td>
<td>1 CD1</td>
</tr>
<tr>
<td>1734</td>
<td>1 CD2</td>
</tr>
<tr>
<td>1735</td>
<td>1 CD2 INPUTS FROM ASGCOM</td>
</tr>
<tr>
<td>1736</td>
<td>1 CD2</td>
</tr>
<tr>
<td>1737</td>
<td>1 CD2 SYMTAB,SSTRNG,RESULT,RSLTPT,CLSTAN,MAPWDS</td>
</tr>
<tr>
<td>1738</td>
<td>1 CD2</td>
</tr>
<tr>
<td>1739</td>
<td>1 CD2</td>
</tr>
<tr>
<td>1740</td>
<td>1 CD3</td>
</tr>
<tr>
<td>1741</td>
<td>1 CD3 OUTPUTS TO ASGCOM</td>
</tr>
<tr>
<td>1742</td>
<td>1 CD3</td>
</tr>
<tr>
<td>1743</td>
<td>1 CD3 CPRNDS,REXT,NI/NUMDS,RSLTPT</td>
</tr>
<tr>
<td>1744</td>
<td>1 CD3</td>
</tr>
<tr>
<td>1745</td>
<td>1 CD3</td>
</tr>
<tr>
<td>1746</td>
<td>1 CD5 EXTERNAL REFERENCES</td>
</tr>
<tr>
<td>1747</td>
<td>1 CD5</td>
</tr>
<tr>
<td>1748</td>
<td>1 CD5 FDS - XPRES,FRMO,XPICS</td>
</tr>
<tr>
<td>1749</td>
<td>1 CD5</td>
</tr>
<tr>
<td>1750</td>
<td>1 CD5</td>
</tr>
<tr>
<td>1751</td>
<td>1 CD5 RTE - IANL</td>
</tr>
<tr>
<td>1752</td>
<td>1 CD5</td>
</tr>
<tr>
<td>1753</td>
<td>1 CD5</td>
</tr>
</tbody>
</table>

Note: The document appears to be a FORTRAN program listing, detailing the calling procedure for the subroutine XIFRE and its usage in the context of data assignments for free-type object data elements. The listing includes input and output parameters and external references.
1755 1 BEGIN XZFRE
1756 2 * CASE (RESULT DATA TYPE ) :ZFRESTR:, :ZFREF:, :ZFREF:, :ZPREFIX:
1757 3 *
1758 4 * :ZPREFIX:, :ZPREFIX:,
1759 3 *
1760 3 :ZFRESTR:
1761 3 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1762 3 MOVE CHARACTER STRING TO RESULT LOCATION
1763 3 :ZFREF:
1764 3 IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
1765 4 POP DISPLACEMENT FROM RESULT STACK
1766 4 ELSE FREE OR CHARACTER ELEMENT HAS NOT BEEN SUBSCRIPTED
1767 4 SET DISPLACEMENT = 0
1768 3 ENDIF
1769 3 POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
1770 3 GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
1771 3 IF RESULT OPERAND IS FREE, THEN
1772 4 CALL XPRSTO 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 XPREQ 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 :ZPREFIX:
1780 3 CALL XZPCS TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781 3 SET # WORDS TO BE STORAD = RESULT DATA TYPE
1782 2 END CASE
1783 1 END XZFRE
FORTRAN CALLING PROCEDURE:

CALL XIFXD

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

INPUTS FROM ASGCOM

OUTPUTS TO ASGCOM

RSLTPT,REQST,NUMWDS,OPRND5

FDS - XPREQ,XPXIT,XRMOV,XILSS,XIMSG,XIPCS

RTE - IAND
1815  1 BEGIN XIFX:
1816  2    SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1817  3    CASE (RESULT DATA TYPE):
1818  4      :NOCHAR:
1819  5      :FIXRE:
1820  6      :FIXF:
1821  7      :LIST:
1822  8      :FIXFIX:
1823  9    END CASE
1824  10  END XIFX:
1825  
1826  1 BEGIN XIFX:
1827  2    SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1828  3    CASE (RESULT DATA TYPE):
1829  4      :NOCHAR:
1830  5      :FIXRE:
1831  6      :FIXF:
1832  7      :LIST:
1833  8      :FIXFIX:
1834  9    END CASE
1835  10  END XIFX:
1836  
1837  1 BEGIN XIFX:
1838  2    SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
1839  3    CASE (RESULT DATA TYPE):
1840  4      :NOCHAR:
1841  5      :FIXRE:
1842  6      :FIXF:
1843  7      :LIST:
1844  8      :FIXFIX:
1845  9    END CASE
1846  10  END XIFX:
FORTAN CALLING PROCEDURE

CALL XILSS (LU, STRING, INDEX)

XILSS IS CALLED TO LIST A SYMBOLIC STRING AND AN INDICATOR TO A
PARTICULAR TOKEN IN THAT STRING

INPUT

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

OUTPUT

THE SYMBOLIC STRING IS OUTPUT TO THE LU FOLLOWED BY A LINE CONTAINING
AN INDICATOR (UP ARROW) TO THE DESIGNATED TOKEN.
### INTERNAL VARIABLES

- `CONTRL`: Control Table Describing Disposition and Processing.
- For each of the token values 1-32, each control table entry is 3 words:
  - Word 1 (SIZE) = No. Of Words In Print Buffer
  - Word 2 (FIELD) = Contents To Go Into Print Buffer
  - Word 3 (TOKSZ) = No. Of Words In Symbolic Strings

<table>
<thead>
<tr>
<th>Token</th>
<th>Word 1 (SIZE)</th>
<th>Word 2 (FIELD)</th>
<th>Word 3 (TOKSZ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>3</td>
<td>1 =&gt; CALL XR16</td>
<td>2</td>
</tr>
<tr>
<td>REAL</td>
<td>7</td>
<td>2 =&gt; CALL XR14</td>
<td>3</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>9</td>
<td>3 =&gt; CALL XR13</td>
<td>4</td>
</tr>
<tr>
<td>NAME</td>
<td>3</td>
<td>4 =&gt; USE 3 WORDS</td>
<td>4</td>
</tr>
<tr>
<td>FOLLOWING TOKEN</td>
<td>0</td>
<td>0 =&gt; ERROR (INVALID)</td>
<td>0</td>
</tr>
<tr>
<td>8 CHAR. STRING</td>
<td>1</td>
<td>1 =&gt; USE VALUE</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>BEGINNING 2 PAST</td>
<td>1</td>
<td>USE SIZE WORDS</td>
</tr>
<tr>
<td>2</td>
<td>USE SIZE+2 WORDS</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
1924 1 C*******
1925 1 CD5
1926 1 CD5 NOTES
1927 1 CD5 USES ROUTINES
1928 1 CD5
1929 1 CD5 EXEC
1930 1 CD5 X116
1931 1 CD5 X1R14
1932 1 CD5 XRD18
1933 1 CD5 XMD86
1934 1 CD5 XMSG
1935 1 CD5 XMNOV
1936 1 CD5 C*******
1937 2 BEGIN XLS
1938 2 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER
1939 2 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED
1940 2 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32
1941 2 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)
1942 2 EXIT TO ERROR 2 IF FIELD = 0
1943 2 IF SIZE < 0, THEN
1944 3 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING
1945 3 ENDIF
1946 3 IF TOKSIZ < 0, THEN
1947 4 SET TOKSIZ TO SIZE + 2
1948 3 ENDIF
1949 3 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN
1950 4 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE
1951 4 SET BUFFER POINTER TO 1ST POSITION FOR DATA
1952 5 IF INDICATED TOKEN PROCESSED, THEN
1953 5 OUTPUT LINE WITH INDICATOR
1954 4 ENDIF
1955 3 ENDIF
1956 3 IF FIELD > 0, THEN
1957 4 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION
1958 3 ELSE
1959 3 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD
1960 4 SOME: CALL X116 WITH VALUE IN NEXT WORD OF SYMB. STRING
1961 5 AND PUT RESULTS INTO PRINT BUFFER
1962 5 :TWO: CALL X1R14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING
1963 5 AND PUT RESULTS INTO PRINT BUFFER
1964 5 :THREE: CALL XRD18 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING
1965 5 AND PUT RESULTS INTO PRINT BUFFER
1966 5 :FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER
1967 5 :FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING
1968 5 :SIX: CALL X116 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS
1969 5 INTO PRINT BUFFER FOLLOWED BY AN "R"
1970 5 :EXIT: PUT A ' CHARACTER INTO THE PRINT BUFFER, INCREMENT THE BUFFER
1971 5 INDEX BY 1
1972 5 CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE
1973 5 IF INDICATED TOKEN HAS BEEN PROCESSED, AND
1974 5 INDICATOR LINE NOT YET OUTPUT, THEN
Fortran Calling Procedure

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

1 ****************
1 CONT - THREE CONTROL WORDS CONTAINING THE TERMINAL LU AND THE DEBUG CONTROL FLAG. IF CONT(3) BIT 12 IS ON, XUGDG WILL BE CALLED AFTER THE MESSAGE IS OUTPUT.

1 NUMBER - INTEGER MESSAGE NUMBER OF THE FORM 'ANN' WHERE
1 A - AREA INDICATOR AS FOLLOWS
1 1 - AS
1 2 - XB
1 3 - XE
1 4 - XI
1 5 - XS
1 6 - XT
1 7 - XX
1 8 - XL
1 9 - DF
1 10 - SC
1 MN - MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'

1 LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED
1 'SOURCE' (NOT USED IF 'MN' OF 'NUMBER' IS ZERO)

1 LENGTH - INTEGER NUMBER OF WORDS OF 'SOURCE' TO BE INSERTED INTO MESSAGE. ZERO INDICATES NO INSERTION

1 SOURCE - ARRAY OF CHARACTERS TO BE INSERTED INTO MESSAGE (NOT USED IF 'LENGTH' IS ZERO)

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

1 ****************
1 NOTES
1 USES FOS SYSTEM MESSAGE FILE JXMSG
1 USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XRMOV, XUGDG
1 BEGIN XZMSG
2 SEPARATE NUMBER INTO AREA AND MESSAGE NUMBER
3 SET NUMBER IN PREFIX
4 READ MESSAGE DIRECTORY RECORD
5 IF AREA VALID
6 THEN
7 SET AREA CODE IN PREFIX
8 IF MESSAGE NUMBER > 0
9 THEN
10 IF VALID MESSAGE NUMBER
11 THEN
12 COMPUTE MESSAGE RECORD NUMBER
13 READ RECORD
14 CALL XRMOV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
15 ELSE
16 EXIT TO ERROR:
17 ENDIF
18 ENDIF
19 CALL XRMOV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
20 IF MESSAGE NUMBER > 0
21 THEN
22 CALL XRMOV TO MOVE 2 REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
23 ENDIF
24 ENDIF
25 ELSE
26 SET AREA IN PREFIX
27 DEERROR: CALL XRMOV TO MOVE 'XZMSG ERROR' INTO BUFFER
28 CALL XRMOV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
29 ENDIF
30 OUTPUT BUFFER TO USER'S TERMINAL
31 IF DEBUG IS REQUESTED
32 THEN
33 CALL XUDOOG
34 ENDIF
35 END
36 END XZMSG
BEGIN XOPR

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

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

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

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

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

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

:SUBSCR:
IF RESULT STACK IS NOT EMPTY, THEN
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 OF CHARACTER, THEN CALL XPRN 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

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

:UNIMIN:
CHANGE SIGN OF OPERAND FOR RESULT
ENDCASE

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

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

CALL XPCS(TARGET,OPNUM)

: XPCS IS USED BY ASSIGN TO POP AN OPERAND FROM THE RESULT STACK, CONVERT 3 TO A TARGET TYPE, AND STORE IT FOR USE IN A MATH OR FUNCTION OPERATION

INPUTS

TARGET - DESIRED FDS FIXED DATA TYPE
OPNUM - OPERAND NUMBER FOR ENTRY CURRENTLY BEING SET UP

FROM ASGCOM - LU,STRTXG,RESULT,RSLTPT,DATYPES

OUTPUTS TO ASGCOM
RSLTPT,OPNUM

EXTERNAL REFERENCES
FBS - XPXIT,XXMOV,XXLSX,XXMSG
RTE - CBE,FLOT,IFIX,OUF,SNGL
FORTRAN CALLING PROCEDURE:
CALL XIPS1

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

INPUTS FROM ASGXM
LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTL

OUTPUTS TO ASGXM
EXPTR, RNGSTK, POLISH, SYNTAX

INTERNAL VARIABLES
GRPSK - 2X120 ARRAY USED TO TRACK FUNCTIONS, SUBSCRIPTS, AND
PARENTHEtical GROUPINGS
OPSK - 2X120 ARRAY; EACH ENTRY CONTAINS OPERATOR TOKEN AND ITS OUTPUT
PRIORITY
PRCNT - PRECEDENT TYPE FOR PRECEDING CHARACTER; USED FOR SYNTAX CHECK
TKNPR - POINTER TO TOKEN BEING PROCESSED IN SYMBOLIC STRING
TOKEN - TOKEN CURRENTLY BEING PROCESSED

EXTERNAL REFERENCES
RTE - IAMD, MINO
FD5 - XPXII, XILSS, XIMSG, XISYM
2265 1 BEGIN XIPS1
2266 2 DO FOR EACH TOKEN UNTIL ; OR ESS IS REACHED
2267 3 PERFORM SETUP FOR SPECIAL HANDLING FOR NUMBERS, NAMES, +, *, (, AND ","
2268 4 PERFORM STRING FOR SYNTAX CHECKING AND POLISH STRING BUILD
2269 5 END DO
2270 2 PERFORM RANGE TO PROCESS RANGE DEFINITION, IF NECESSARY
2271 1 END XIPS1
1 BEGIN SETUP
2 CASE TOKEN
3   :INTVL: :SYMBOL: :SYMBOL:
4     NAME CHARACTER + - * / < >
7   = ( ) 
10   :LPAREN: :ENDCASE: :ENDCASE: :ENDCASE: 
11   :INTVL: :ENDCASE: :ENDCASE: :ENDCASE: 
12   :ENDCASE:
13   :SYMBOL:
14   IF TOKEN IS A NAME AND NEXT TOKEN IS A LEFT BRKT, THEN
15   START SEARCH UNTIL FUNCTION TABLE IS CHECKED
16   EXIT IF TOKEN NAME MATCHES FUNCTION NAME
17   CHANGE TOKEN TO FUNCTION'S INDEX IN TABLE + 128
18   PUSH COUNT= BINARY/UNARY FLAG AND LIMIT=0 ONTO GROUPING STACK
19   END LOOP
20   ERROEXIT: IF NO MATCH IS FOUND
21   ELSE TOKEN IS AN OPERAND
22   IF TOKEN IS A NAME AND NEXT TOKEN IS (, THEN
23   SET SUBSCRIPT FLAG
24   ENDIF
25   CALL XSYM TO STORE SYMBOL AND SUBSCRIPT FLAG
26   PUSH SYMBOL INDEX ONTO EXPRESSION STACK
27   ENDIF
28   :CHAR:
29   PUSH NEGATIVE POINTER TO CHARACTER STRING ON EXPRESSION STACK
30   :PLUS:
31   IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
32   INCREASEMENT TO NEXT TOKEN AND EXIT TO :ENDIF
33   ENDIF
34   :MINUS:
35   IF PRECEDING TOKEN WAS LEFT BRACKET OR ( OR , OR =, THEN
36   CHANGE TOKEN TO UNARY MINUS
37   ENDIF
38   :ASTER:
39   IF NEXT TOKEN IS *, THEN
40   CHANGE TOKEN TO EXPONENTIATION
41   ENDIF
42   :LPAREN:
43   IF PRECEDING TOKEN WAS A VARIABLE OR DATA ELEMENT NAME, THEN
44   PUSH COUNT AND LIMIT ALLOWING DOUBLE SUBSCRIPTING ONTO GROUPING STACK
45   ELSE EXPRESSION IS BEING PROCESSED
46   PUSH COUNT AND LIMIT ALLOWING NO SUBSCRIPTING ONTO GROUPING STACK
47   ENDIF
48   :ENDCASE
2325 2 :ENDCAS:
2326 1 EXIT SETUP

2327 2 :INVLD:
2328 2 SET MESSAGE TO BE OUTPUT TO "INVALID CHARACTER"

2329 2 :BADFUN:
2330 2 SET MESSAGE TO BE OUTPUT TO "FUNCTION NOT SUPPORTED BY THIS PROCESSOR"
2331 2 CALL XINSG TO OUTPUT MESSAGE TO USER
2332 2 CALL XILSS TO LIST SYMBOLIC STRING
2333 2 CALL XPXIT TO EXIT PROCESSOR
2334 1 END SETUP
BEGIN STRING
SET INDEX INTO SYNTAX TABLE TO MIN(TOKEN, 40)
ERREXIT TO :SYNTAX1: IF THIS TOKEN IS NOT VALID ACCORDING TO SYNTAX TABLE
IF TOKEN IS NOT AN OPERAND, THEN
DO UNTIL TOKEN IS PUSHED ONTO OPERATOR STACK OR DISCARDED
IF INPUT PRIORITY OF THIS TOKEN > OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR
STACK, THEN
IF TOKEN IS , THEN
INCREMENT COUNT FOR TOP ENTRY IN GROUPING STACK
ERREXIT TO :SYNTAX1: IF COUNT > COMMA LIMIT FOR TOP ENTRY IN GROUPING STK
IF COMMA LIMIT SHOWS SUBSCRIPTING IN FUNCTION LIST (LIMIT > 0), THEN
PUSH TOKEN AND OUTPUT PRIORITY ONTO OPERATOR STACK
ELSE
DISCARD (FUNCTION LIST IS BEING PROCESSED)
ENDIF
ELSE OPERATOR IS NOT ,
PUSH TOKEN AND ITS OUTPUT PRIORITY ONTO OPERATOR STACK
ENDIF
ELSE INPUT PRIORITY IS < OR = OUTPUT PRIORITY
IF INPUT PRIORITY < OUTPUT PRIORITY OF TOP ENTRY IN OPERATOR STACK OR
INPUT PRIORITY = OUTPUT PRIORITY NOT = 2, THEN
ERREXIT TO :SYNTAX1: IF INPUT PRIORITY = 0 AND OPERATOR STACK IS EMPTY
IF TOP ENTRY OF OPERATOR STACK IS =, THEN
ERREXIT IF THIS IS NOT LAST ENTRY ON OPERATOR STACK TO :BADEQL:
ERREXIT IF GROUPING STACK IS NOT EMPTY TO :SYNTAX2:
DISCARD TOKEN (; OR ESS)
ENDIF
POP OPERATOR STACK
POP OPERATOR ONTO EXPRESSION STACK
ELSE BRACKETS OR PARENTHESIS HAVE BEEN MATCHED
IF TOP ENTRY OF OPERATOR STACK IS [ THEN
ERREXIT TO :SYNTAX2:
IF TOP ENTRY OF GROUPING STACK INDICATES SUBSCRIPTING (LIMIT > 0), THEN
PUSH SUBSCRIPTING OPERATOR ONTO EXPRESSION ARRAY
ENDIF
ELSE TOP OPERATOR ENTRY IS OPEN BRACKET (FUNCTION LIST)
ERREXIT IF CURRENT TOKEN IS NOT CLOSE BRACKET TO :SYNTAX2:
ERREXIT TO :FLSTR: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GROUPING STK COUNT = 0)
ENDIF
POP OPERATOR STACK
POP GROUPING STACK
DISCARD CURRENT TOKEN
ENDIF
ENDIF
END DO
INCREMENT TO NEXT TOKEN USING TOKEN LENGTH FROM SYNTAX TABLE
EXIT STRING
:SYNTAX1:
SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"
:SYNTAX2:
SET MESSAGE TO BE OUTPUT TO "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY"
:COMERR:
SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
2400 1 BEGIN RANGE
2401 2 INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
2402 3 IF LAST TOKEN INPUT HAS ; THEN
2403 4 DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
2404 5 ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
2405 6 CALL XISYM TO INSERT NAME IN SYMOL TABLE, IF NECESSARY, FLAG ENTRY AS
2406 7 AN INDEX, AND RETURN SYMBOL TABLE INDEX
2407 8 ERREXIT IF NEXT TOKENS ARE NOT "-INTEGER,INTEGER" TO :RNGSYM:
2408 9 IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
2409 10 SET INCREMENT TC = 1
2410 11 EMDIF
2411 12 PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
2412 13 EMDNT
2413 14 ERMEXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
2414 15 EMDIF
2415 16 EXIT RANGE
2416 17 :RNGSYM:
2417 18 SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ";NAME=
2418 19 * INTEGER,INTEGER"
2419 20 CALL XMSG TO OUTPUT MESSAGE TO USER
2420 21 CALL XILSS TO LIST SYMBOLIC STRING
2421 22 CALL XPRIT TO EXIT PROCESSOR
2422 23 END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

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

INPUTS FROM ASGCMD

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

OPINFO, NUM流DS, DATTYP, RSLTPT, EXPRX, CLSREF, CLSTRM, DECLAS,

RNGSTR, POLISH, SYNTAX, FNCTBL

OUTPUTS TO ASGCMD

SYMTAB, RESULT, RERST, RSLTPT, OPINFO

INTERNAL VARIABLES

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

EXTERNAL REFERENCES

FDS - XPERE, XPXIT, XRMOD, XZCMR, XZFRM, XZFXD, XZLSS, XZMSG,

XZPRK, XZPSC, XZRET

RTE - IABS, IADM, MAXB
BEGIN XIPS2
PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPTED FIXED-TYPE DATA ELEMENTS
INITIALIZE RANGE VALUES FOR ITERATION
DO UNTIL ALL RANGES ARE FINISHED
DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
POPE ENTRY FROM POLISH STRING
IF ENTRY IS AN OPERAND, THEN
PERFORM RPUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH ONTO RESULT STACK
ELSE ENTRY IS AN OPERATOR
IF OPERATOR IS NOT "=", THEN
PERFORM EXEVAL TO EVALUATE POLISH STRING
ELSE OPERATOR IS =
PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
ENDEF
END IF
END DO
END DO
END XIPS2
2486 1 BEGIN TOC
2487 2 DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
2488 3 IF SYMBOL NOT = O (SYMBOL IS DE OR INDEX), THEN
2489 4 IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN
2490 5 CALL XPREQ TO RETRIEVE TOC ENTRY
2491 4 ELSE SYMBOL IS A RANGE INDEX
2492 5 SET DATA TYPE TO INTEGER
2493 4 ENDF
2494 3 ENDF
2495 2 END DO
2496 2 CALL XPREQ WITH A CLOSE BUFFER REQUEST
2497 1 END TOC
2508 1 BEGIN RPUsh
2509 2 IF OPERAND IS A CHARACTER STRING POINTER (OPERAND < 0), THEN
2510 3 PUSH ABSOLUTE VALUE OF OPERAND AND -3 DATA TYPE INTO RESULT STACK
2511 4 ELSE OPERAND IS A SYMBOL TABLE INDEX
2512 5 CALL SIZE TO RETRIEVE DATA AND DATA TYPE FOR OPERAND
2513 6 PUSH RETRIEVED VALUE AND DATA TYPE INTO RESULT STACK
2514 7 ENDR
2515 8 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 ENDS
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 :MIXERR:
2526 10 CALL XSPF TO RETRIEVE DATA AND TYPE FOR FIRST 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 :MIXERR:
2530 14 ENDS
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 XIFMC TO PERFORM FUNCTION FOR RESULTS
2538 22 SET 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 :SAME:
2543 27 IF # OPERANDS > 1, THEN
2544 28 SET TARGET TYPE TO MAX OF TWO DATA TYPES
2545 29 ELSE # OPERANDS = 1
2546 30 SET TARGET TYPE TO OPERAND'S DATA TYPE
2547 31 ENDS
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 XIPPR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE ONTO STACK
2555 39 END IF
2556 40 EXIT EVAL
2557 41 :MIXERR:
2558 42 SET MESSAGE TO BE OUTPUT TO "FREE OR CHARACTER DATA ELEMENT FOUND IN AN
2559 43 EXPRESSION"
2560 44 CALL XSPG TO OUTPUT MESSAGE TO USER
2561 45 CALL XLSS TO LIST SYMBOLIC STRING
2562 46 CALL XPAR TO EXIT PROCESSOR
2563 47 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 ) :
0 :FREE; :FIXED; :FIXED; :CHAR; :CHAR; :
1 :FIXED; :FIXED; :CHAR; :CHAR; :
2 :FREE; :FREE; :CHAR; :CHAR; :
3 :FREE; :FREE; :CHAR; :CHAR; :
4 :FREE; :FREE; :CHAR; :CHAR; :
5 :FREE; :FREE; :CHAR; :CHAR; :
6 :FREE; :FREE; :CHAR; :CHAR; :
7 :FREE; :FREE; :CHAR; :CHAR; :
8 :FREE; :FREE; :CHAR; :CHAR; :

CALL XFRP TO RETRIEVE DATA AND SET UP FOR STORE
CALL FIXOBJ TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
CALL CHROUP TO RETRIEVE DATA AND SET UP FOR STORE
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 NUM WORDS CALCULATED INTO OBJECT AT OBJECT'S DISPLACEMENT
CALL XPREQ TO STORE DATA IN OBJECT
END REPLAC
1 BEGIN RNGSET
2 DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
3 * RANGES ARE PROCESSED
4 IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
5 INCREMENT RANGE VALUE
6 ELSE
7 SET RANGE INDEX VALUE TO BEGIN VALUE
8 ENDIF
9 END DO
10 END RNGSET
FORTRAN Calling Procedure:

CALL XIRET(ENTRY)

XIRET is used by the ASSGN routine XIPS2 to retrieve the data value for a given symbol index from the symbol table and push it onto the result stack along with its data type.

INPUTS

ENTRY - Symbol Table Index Currently Being Processed
FROM ASSGN - SYMTAB, RSLPT

OUTPUTS TO ASSGN
RESULT, RSLPT

EXTERNAL REFERENCES
FDS - XRMOV
RTE - IAMO

Other
2639 1 BEGIN XIRET
2640 2 IF SYMBOL ENTRY IS FOR NON-SUBSCRIPTED FIXED-TYPE DATA (DATA ELEMENT OR
2641 3 RANGE INDEX), THEN
2642 4 SET RETURN VALUE = VALUE IN SYMBOL TABLE FOR THIS ENTRY
2643 5 SET RETURN DATA TYPE = DATA TYPE IN SYMBOL TABLE FOR THIS ENTRY
2644 6 ELSE SYMBOL IS CHARACTER OR FREE DATA OR SUBSCRIPTED FIXED DATA
2645 7 SET RETURN VALUE = SYMBOL TABLE INDEX
2646 8 SET RETURN DATA TYPE = -1
2647 9 ENDIF
2648 1 END XIRET
2650 1 CD0 FORTRAN CALLING PROCEDURE
2651 1 CD0 CALL XSYM (TKMPTR, FLAG, SYMIND)
2652 1 CD0
2653 1 CD0
2654 1 CD0********
2655 1 CD1 XSYM PROVIDES SYMBOL TABLE BUILDING AND ACCESS FOR ASSGN. IT
2656 1 CD1 USES XSYM TO PERFORM GENERAL SYMBOL TABLE FUNCTIONS THEN
2657 1 CD1 ACCOMPLISHES RANGE AND SUBSCRIPT PROCESSING DIRECTLY
2658 1 CD1
2659 1 CD1
2660 1 CD0********
2661 1 CD2 INPUT
2662 1 CD2 TKMPTR - INDEX INTO INPUT SYMBOLIC STRING (SSTRING) OF TOKEN TO BE
2663 1 CD2 ENTERED OR LOCATED IN SYMBOL TABLE (SYMTAB)
2664 1 CD2 FLAG - SPECIAL PROCESSING FLAG
2665 1 CD2 0, NORMAL PROCESSING
2666 1 CD2 1, SYMBOL IS A RANGE
2667 1 CD2 2, SYMBOL IS SUBSCRIPTED
2668 1 CD2 ASCOM COMMON - LU, SSTRING, STWIDE, SYMTAB, TOKENS,
2669 1 CD2
2670 1 CD0********
2671 1 CD3 OUTPUT
2672 1 CD3 SYMIND - INDEX INTO SYMBOL TABLE (SYMTAB) WHERE SYMBOL IS LOCATED.
2673 1 CD3 VALUE OF SYMIND INDICATES ENTRY NUMBER, I.E., 1, 2, ...
2674 1 CD3 ASCOM COMMON - SYMTAB
2675 1 CD3
2676 1 CD0********
2677 1 CD4 INTERNAL
2678 1 CD4 SYMPTR - INDEX TO FLAG WORD OF ENTRY IN SYMTAB. VALUE IS EQUAL TO
2679 1 CD4 SYMIND * STWIDE
2680 1 CD4
2681 1 CD0********
2682 1 CD5 EXTERNAL REFERENCES
2683 1 CD5 XPRINTF, XILSS, XIMSG, XSYM
BEGIN XSYM

CALL XSYM TO ENTER TOKEN INTO TABLE OR RETURN INDEX TO EXISTING ENTRY

IF TOKEN IS A NAME, THEN

IF SPECIAL PROCESSING FLAG IS SET (1=RANGE INDEX, 2=SUBSCRIPTED), THEN

IF FLAG INDICATES RANGE INDEX, THEN

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

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

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

ENDIF

SET ENTRY FLAG WORD TO FLAG VALUE

ENDIF

ENDIF

EXIT XSYM

BADREG:

CALL XMSG TO OUTPUT ERROR DESCRIPTION

CALL XLSS TO DISPLAY SYMBOLIC STRING AND POINT TO ERROR

CALL XPTT TO TERMINATE PROCESSOR

EXIT XSYM
FORTRAN CALLING PROCEDURE

CALL XISYT (TKMPTR, SYMIND)

A GENERAL CAPABILITY FOR SYMBOL TABLE ACCESS FOR TOKENS INPUT VIA

A SYMBOLIC STRING

Provided key values are suitably located in common, XISYT provides

ENTERED OR LOCATED IN SYMBOL TABLE (SYMTAB)

INPUT

TKMPTR - INDEX INTO INPUT SYMBOLIC STRING (SSTRING) OF TOKEN TO BE

ENTERED OR LOCATED IN SYMBOL TABLE (SYMTAB)

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

OUTPUT

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

VALUE OF SYMIND INDICATES ENTRY NUMBER, i.e., 1, 2, ...

ASCOM COMMON - LASTSY, SYMTAB

INTERNAL

DISP - DISPLACEMENT INTO SYMBOL TABLE ENTRY FOR FIELD TO BE USED

AS KEY. NAMES USE A VALUE OF 1, CONSTANTS A VALUE OF 0.

EOLOOP - INDEX OF LAST ALLOCATED WORD IN SYMTAB. VALUE IS EQUAL

TO LASTSY*STWIDE

IBUF - INTERNAL BUFFER FOR SYMBOL VALUE USED TO ASSURE LAST

WORDS OF INTEGER AND REAL CONSTANTS ARE ZERO.

EXTERNAL REFERENCES

XRCPRI, XRMN

SPECIAL REMARKS

THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS

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

I = TOKEN CODE I (3 WORDS) I RESERVED FOR TOC ENTRY I (3 WORDS) I (N WORDS) I

OTHER
### SYMBOL DEFINITION TABLE

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD0</td>
<td>2114</td>
</tr>
<tr>
<td>ADD1</td>
<td>1108</td>
</tr>
<tr>
<td>ADD2</td>
<td>1105</td>
</tr>
<tr>
<td>ADD3</td>
<td>1103</td>
</tr>
<tr>
<td>ADD4</td>
<td>1124</td>
</tr>
<tr>
<td>ADD5</td>
<td>1112</td>
</tr>
<tr>
<td>ADD6</td>
<td>1123</td>
</tr>
<tr>
<td>ADD7</td>
<td>1122</td>
</tr>
<tr>
<td>ADD8</td>
<td>1102</td>
</tr>
<tr>
<td>ADD9</td>
<td>1120</td>
</tr>
<tr>
<td>ADD10</td>
<td>1119</td>
</tr>
<tr>
<td>ADD11</td>
<td>1118</td>
</tr>
<tr>
<td>ADD12</td>
<td>1117</td>
</tr>
<tr>
<td>ADD13</td>
<td>1116</td>
</tr>
<tr>
<td>ADD14</td>
<td>1115</td>
</tr>
<tr>
<td>ADD15</td>
<td>1114</td>
</tr>
<tr>
<td>ADD16</td>
<td>1101</td>
</tr>
<tr>
<td>ADD17</td>
<td>1100</td>
</tr>
<tr>
<td>ADD18</td>
<td>1111</td>
</tr>
<tr>
<td>ADD19</td>
<td>1110</td>
</tr>
<tr>
<td>ADD20</td>
<td>1109</td>
</tr>
<tr>
<td>ADD21</td>
<td>1108</td>
</tr>
<tr>
<td>ADD22</td>
<td>1090</td>
</tr>
<tr>
<td>ADD23</td>
<td>1094</td>
</tr>
<tr>
<td>ADD24</td>
<td>1099</td>
</tr>
<tr>
<td>ADD25</td>
<td>1104</td>
</tr>
<tr>
<td>ADD26</td>
<td>1107</td>
</tr>
<tr>
<td>ADD27</td>
<td>1097</td>
</tr>
<tr>
<td>ADD28</td>
<td>1096</td>
</tr>
<tr>
<td>ADD29</td>
<td>1126</td>
</tr>
<tr>
<td>ADD30</td>
<td>1091</td>
</tr>
<tr>
<td>ADD31</td>
<td>1113</td>
</tr>
<tr>
<td>ADD32</td>
<td>1121</td>
</tr>
<tr>
<td>ADD33</td>
<td>1092</td>
</tr>
<tr>
<td>ADD34</td>
<td>1128</td>
</tr>
<tr>
<td>ADD35</td>
<td>1095</td>
</tr>
<tr>
<td>ADD36</td>
<td>1093</td>
</tr>
<tr>
<td>ADD37</td>
<td>1130</td>
</tr>
<tr>
<td>ASSGN</td>
<td>272</td>
</tr>
<tr>
<td>ASTEN</td>
<td>2314</td>
</tr>
<tr>
<td>BADDEL</td>
<td>2593</td>
</tr>
<tr>
<td>BADFUN</td>
<td>2529</td>
</tr>
<tr>
<td>BADRNG</td>
<td>2701</td>
</tr>
<tr>
<td>CALSTY</td>
<td>1098</td>
</tr>
<tr>
<td>CALSTY</td>
<td>1132</td>
</tr>
<tr>
<td>CHAR</td>
<td>2304</td>
</tr>
<tr>
<td>CHAR</td>
<td>2596</td>
</tr>
<tr>
<td>CHRRE</td>
<td>937</td>
</tr>
<tr>
<td>CHRSTR</td>
<td>931</td>
</tr>
<tr>
<td>COMARR</td>
<td>2389</td>
</tr>
<tr>
<td>DATA1</td>
<td>2499</td>
</tr>
<tr>
<td>DSP</td>
<td>423</td>
</tr>
<tr>
<td>DREF</td>
<td>2208</td>
</tr>
<tr>
<td>DEFIN</td>
<td>483</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>2123</td>
</tr>
<tr>
<td>ENDCAS</td>
<td>2325</td>
</tr>
<tr>
<td>ENDO</td>
<td>2362</td>
</tr>
<tr>
<td>ENOSC</td>
<td>641</td>
</tr>
</tbody>
</table>
DO - CONDITIONAL ITERATION (LOOPING) PROCESSOR

SCHEDULED BY FBS

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

INPUT FROM MANAGER VIA SCHEDULING PARAMETERS

LU - LOGICAL UNIT OF USER'S TERMINAL
FLAGS - DEBUG FLAGS FROM USER SIGN-OUT
SENDSP - INDEX OF CURRENTLY EXECUTING SEQUENCE TABLE 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
WHILE - LOOP WHILE RELATION BETWEEN OPRND1 AND OPRND2 IS
TRUE, TEST IS MADE PRIOR TO STARTING EACH LOOP

OPRND1 - FIRST REAL QUANTITY TO COMPARE (VALUE NOT RETRIEVED,
CHARACTERISTICS PASSED TO ENDDO)
OPRND2 - SECOND REAL QUANTITY TO COMPARE (SEE OPRND1)

INPUT FROM ALL ON REQUEST TO MANAGER
INTBUF - INTERFACE TABLE (LESS LITERAL AREA)
LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE
LITERAL AREA
SEQSTK - SEE OUTPUT DEFINITION
SEQRTH - 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 - $FRAWH $RESET NUMBER OF ENDDO IF RETURN = 3

INPUT/OUTPUT FROM TO ALL
ADSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27,M) 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 SDR, VOLUME IV, FIGURE 1.2-17)
WORDS 8-14 - INTERFACE TABLE ENTRY FOR OPRND1 (SAME AS
IN ORIGINAL TABLE EXCEPT FOR POSSIBLE NEW
VALUES POINTING INTO LITERAL AREA)
WORDS 15-21 - INTERFACE TABLE ENTRY FOR OPRND2 (SEE
ABOVE)
WORDS 22-25 - LITERAL AREA FOR VALUE(S)/SUBSCRIPT(S) OF
OPRND1 & 2
WORD 26 - RELATION ID CODE AS FOLLOWS
0 - =
1 - >
2 - >= OR =>
3 - =
4 - <= OR <=
5 - <
WORD 27 - SEQUENCE NUMBER OF TOP OF LOOP

INTERNAL VARIABLES
CODE - ARRAY OF EIGHT ACCEPTABLE RELATION PHENOMICS AND
CORRESPONDING INTERNAL CODES
DOENT - INDEX INTO BOSSTK FOR NEW 27 WORD ENTRY
PRDUFF - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD
ENTRIES ARE OF THE FORM:

REINT - AWA MANAGEMENT REQUEST CODE
CLASS - CLASS AND TYPE OF DATA
NAME - SIX CHARACTER DATA NAME
SIZE - SIZE OF DATA
DSP - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN
CLASHO - CLASS I/O NUMBER THROUGH WHICH DATA IS
TRANSMITTED
TOP - INDEX INTO BERTH TOP OF LOOP
XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS (SET BY
XPGET)

REFERENCED Routines
EXEC, IAHD; MOD; XRPR; XRPR; XRMOV;
XRSET, XUDEL, XYPAM, XRMSG

DO AND ENDDO MUST BE USED IN PAIRS

BOSSTK IS ALLOCATED IN THE AWA FOR INTERPROCESSOR COMMUNICATIONS
ZESCH CLEANS UP ANY RESIDUAL BOSSTK AFTER EXECUTION CONTROLER
FOR TERMINATION
THE MAXIMUM NUMBER OF NESTED LOOPS IS 4
ALL NESTING (STRUCTURE) ERRORS ARE LEFT FOR EXECUTION TIME
124 1 BEGIN DO
125 2 CALL XGET TO INITIALIZE ACCESS TO XPRER AND TO RETRIEVE DOTYPE AND RELATN
126 3 CALL XPVM TO RETRIEVE EOSTK INTO BUFFER
127 4 IF RETRIEVAL FAILED, THEN
128 5 INITIALIZE BUFFER FOR BUILDING FIRST EOSTK ENTRY
129 6 ENDIF
130 7 IF EOSTK IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
131 8 IF XPRER INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXISTS, THEN
132 9 CALL XPER TO RETRIEVE LITERALS
133 10 ENDIF
134 11 IF RELATN IS A VALID RELATION OPERATOR, THEN
135 12 SET RELATION CODE IN NEW ENTRY IN BUFFER
136 13 SET INTERFACE TABLE HEADER WITH NAME OF EINTAB AND NUMBER OF PARAMETERS OF 2
137 14 DO FOR EACH OPND
138 15 MOVE OPND ENTRY INTO NEW INTERFACE TABLE BUFFER
139 16 IF OPND HAS LITERAL VALUE OR DOUBLE SUBSCRIPTS, THEN
140 17 MOVE LITERAL DATA
141 18 ADJUST LITERAL POINTERS
142 19 ENDIF
143 20 ENDDO
144 21 CALL XPER TO RETRIEVE SSEPDB (EXECUTING SEQUENCE TABLE)
145 22 IF SSEPDB DISPLACEMENT (SSEQSP) > 0, THEN
146 23 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (0 IF END OF TABLE)
147 24 ELSE INSERTED COMMAND
148 25 EXIT TO :ERROR2: IF SEQUENCE NUMBER IS ZERO (MANUAL)
149 26 LOCATE ORIGINAL SEQUENCE ENTRY
150 27 IF ORIGINAL ENTRY WAS ALSO A DO (OVERRIDE CONDITION), THEN
151 28 SET TOP OF LOOP TO NEXT SEQUENCE NUMBER IN TABLE (OR ZERO)
152 29 ELSE (INSERT)
153 30 SET TOP OF LOOP TO CURRENT NUMBER
154 31 ENDIF
155 32 ENDIF
156 33 CASE (:WHILE:, :UNTIL:, :OTHER:) DOTYPE
157 34 :WHILE:
158 35 INVERT RELATION CODE
159 36 INITIALIZE NEST COUNTER TO 1
160 37 START SEARCH FROM TOP OF LOOP ENTRY UNTIL ALL ENTRIES HAVE BEEN EXAMINED
161 38 IF COMMAND IS ENDDO, THEN
162 39 DECREMENT NEST COUNTER
163 40 ELSE
164 41 IF COMMAND IS ANOTHER DO, THEN
165 42 INCREMENT NEST COUNTER
166 43 ENDIF
167 44 ENDIF
168 45 EXIT IF NEST COUNTER IS ZERO
169 46 SET RESET NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
170 47 ENDO LOOP
171 48 EXIT TO :ERROR4: FOR NO MATCHING ENDDO
172 49 END SEARCH
173 50 :UNTIL:
174 51 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
175 52 :OTHER:
176 53 TERMINATE WITH ERROR7 FOR UNRECOGNIZED DOTYPE
177 54 END CASE
178 55 CALL XPER TO OUTPUT NEW EXPANDED EOSTK
179 56 ELSE INVALID RELATION
180 4  TERMINATE WITH ERR05 FOR INVALID RELATION
181 3   ENDIF
182 4   ELSE BDOSTK OVERFLOW
183 4   TERMINATE WITH ERR01 FOR BDOSTK FULL
184 4   ENDIF
185 1   EXIT DO
186 2   :ERR02: TERMINATE FOR EXECUTING IN MANUAL MODE
187 2   :ERR04: TERMINATE FOR NO MATCHING ENDDO FOUND DURING WHILE PROCESSING
188 1   END DO
### FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

**CD0**

190  CD0

191  CD0

192  CD0

193  CD0

194  CD0

**CD1**

195  CD1

196  CD1

THE ELSE UTILITY PROCESSOR LOCATES THE ENDIF CORRESPONDING TO

197  CD1

198  CD1

199  CD1

THE PROCESSOR FOLLOWING ENDIF IS EXECUTED NEXT IN THE SEQUENCE.

200  CD1

**CD2**

201  CD2

202  CD2

SCHEDULING PARAMETERS

203  CD2

204  CD2

LU - LOGICAL UNIT NO. OF THIS FDS USER

205  CD2

206  CD2

FLAGS - DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT

207  CD2

208  CD2

**CD3**

209  CD3

OUTPUT

210  CD3

211  CD3

RPARMS - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPXIT

212  CD3

(1) = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE

213  CD3

NO. GIVEN IN RPARMS(1)

214  CD3

215  CD3

(2) = ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE

216  CD3

217  CD3

**CD4**

218  CD4

REFERENCE ROUTINES

219  CD4

220  CD4

221  CD4

222  CD4

223  CD4

224  CD4

225  CD4

226  CD4

227  CD4

228  CD4

**CD5**
This image contains a page of text that seems to be a part of a computer program or a technical document. The text is not legible due to the low resolution and quality of the image.
298 1 BEGIN ENDDO
299 2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XVSTB
300 2 CALL XUPAN TO RETRIEVE $DOSTRK
301 2 IF RETRIEVAL WAS SUCCESSFUL, THEN
302 3 WRITE $INTAB TO CLASS I/O FROM LAST $DOSTK ENTRY
303 3 WRITE NON-LITERAL PORTION OF $INTAB TO CLASS I/O (LEAVE FOR XPGET)
304 3 CALL XUPAW TO RESTORE $INTAB FROM FIRST CLASS BUFFER INTO AWA
305 3 IF RESTORE SUCCESSFUL, THEN
306 4 CALL XPGET TO RETRIEVE $PRMDS
307 4 CALL XEVAL TO EVALUATE RELATION
308 4 IF RELATION IS TRUE, THEN
309 5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
310 5 IF THIS IS LAST $DOSTK ENTRY, THEN
311 6 DELETE $DOSTK FROM AWA
312 6 ELSE
313 7 CALL XPREG TO REPLACE $DOSTK LESS LAST ENTRY
314 7 ENDIF
315 4 ELSE
316 5 SET RESET NUMBER TO TOP OF LOOP
317 5 ENDIF
318 3 ELSE
319 4 TERMINATE WITH ERROR FOR AWA OVERFLOW
320 3 ENDIF
321 3 ELSE
322 3 TERMINATE WITH ERROR3 FOR ENDDO WITH OUT DO
323 3 ENDIF
324 1 END ENDDO
FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

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

INPUTS

SCHEDULING PARAMETERS

LU = LOGICAL UNIT NO. OF FDS USER

FLAGS = DEBUG FLAGS -- BIT 17 ON WILL CAUSE DEBUG PRINT

OUTPUTS

RMPAMS - RETURN PARAMETERS FOR FDS MANAGER VIA XPXIT

(1) = 0 -> CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE

ROUTES USED

RMPAR

XREXT

XPXIT

XUDBG

ENDIF
466 1 CDB******
467 1 CDB
468 1 CDB
469 1 CDB
470 1 CDB******
471 1 CDB
472 1 CDB
473 1 CDB
474 1 CDB
475 1 CDB
476 1 CDB
477 1 CDB
478 1 CDB
479 1 CDB
480 1 CDB
481 1 CDB
482 1 CDB
483 1 CDB
484 1 CDB
485 1 CDB
486 1 CDB
487 1 CDB
488 1 CDB
489 1 CDB
490 1 CDB******
491 1 CDB
492 1 CDB
493 1 CDB
494 1 CDB
495 1 CDB
496 1 CDB
497 1 CDB
498 1 CDB
499 1 CDB
500 1 CDB
501 1 CDB******
502 1 CDB
503 1 CDB
504 1 CDB
505 1 CDB
506 1 CDB
507 1 CDB
508 1 CDB
509 1 CDB
510 1 CDB
511 1 CDB
512 1 CDB
513 1 CDB******
1 BEGIN IF
2 CALL IMPAR TO GET INPUT (SHEMULING) PARAMETERS
3 CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
4 VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSVERSE IT TO A CODE
5 ERROR IF RELATIONAL OPERATOR INVALID :ERROR:
6 CALL XIEVL TO EVALUATE THE RELATIONAL EXPRESSION
7 IF THE EXPRESSION IS FALSE, THEN
8 CALL XISCH TO LOCATE THE ELSE OR ELSE 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

CVALUE = XIEVL (OPRND1, RELAT, 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

RELAT - RELATION CODE AS FOLLOWS
0 - NOT EQUAL
1 - GREATER THAN
2 - GREATER THAN OR EQUAL
3 - EQUAL
4 - LESS THAN OR EQUAL
5 - LESS THAN

OUTPUT
XIEVL - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

INTERNAL VARIABLES

TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS
FOLLOWS

OPRND1 - OPRND2

<table>
<thead>
<tr>
<th>CODE RELATION</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>0 =</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>1 =</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2 =</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>3 =</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4 =</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5 =</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

(VALUES ASSUMED FOR CODE = 0)

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

REFERENCES ROUTINES

XEXT

NO CHECKS FOR VALID RELATION CODES OR OVERFLOW/UNDERFLOW ARE MADE


1 BEGIN XIEVL
2 CASE (;+; :0; :--; ) DIFFERENCE OF OPND1 AND OPND2
3 ;+; SET FIELD OFFSET TO ZERO (BITS 0-2 OF TTABLE)
4 0; SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
5 ;--; SET FIELD OFFSET TO SIX (BITS 6-8 OF TTABLE)
6 END CASE
7 IF RELAT > 2 (BOTTOM OF TRUTH TABLE), THEN
8 9 COMPLEMENT TTABLE
9 10 DECREMENT RELAT BY 3
11 ENDIF
12 ADD RELAT TO FIELD OFFSET (INDEXES TO CORRECT BIT FOR RELATN AND DIFFERENCE)
13 SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
14 END XIEVL
**FORTRAN CALLING PROCEDURE**

CALL ZISCH (SRCFG, PARGS)

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

**CALLING SEQUENCE**

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

COMMON

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

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

**OUTPUTS**

**CALLING SEQUENCE**

PARGS - PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XPXI?
- (1) = 0 => CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
- (2) = 3 => EXECUTE SEQUENCE NO. GIVEN BY PARGS(2) NEXT
- (6) => ABNORMALLY TERMINATE PROCESSOR EXECUTION SEQUENCE

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

**Routines Used**

EXEC
RMPAR
XPAN
XUDG
XRCPB
XINS
CO
560 1 BEGIN XISCH
561 2 RETRIEVE &SEQTAB3 FROM THE AVA USING XVPAM
562 3 STARTSEARCH UNTIL ALL COMMANDS IN &SEQTAB
563 4 EXIT IF CURRENT COMMAND IS FOUND
564 5 SET NUMBER OF IF NESTS TO 1
565 6 STARTSEARCH FROM NEXT COMMAND IN &SEQTAB UNTIL ALL FOLLOWING COMMANDS
566 7 IF COMMAND IS FOR ENDFPROCESSOR, THEN
567 8 DECREMENT NUMBER OF IF NESTS BY 1
568 9 ELSE
569 10 IF COMMAND IS FOR IF PROCESSOR, THEN
570 11 INCREMENT NUMBER OF IF NESTS BY 1
571 12 ELSE
572 13 IF CALLED BY IF PROCESSOR, AND
573 14 COMMAND IS FOR ELSE PROCESSOR, THEN
574 15 ERREXIT IF THIS IS THE END OF &SEQTAB :ERROR1:
575 16 IF NUMBER OF IF NESTS IS 1, THEN
576 17 DECREMENT NUMBER OF IF NESTS TO 0
577 18 ENDF
578 19 ENDF
579 20 ENDF
580 3 EXIT IF NUMBER OF IF NESTS IS 0
581 4 SET SEQUENCE RESET NUMBER (RPARMS(2)) TO BE SEQUENCE NUMBER OF THE
582 5 NEXT COMMAND IN THE TABLE
583 6 ENDLOOP
584 7 ERREXIT :ERROR1:
585 8 ENDSCH
586 9 ENDFSCH
587 10 ENDSCH
588 11 ERREXIT :ERROR4:
589 12 ENDSCH
590 1 EXIT XISCH
591 2 :ERROR1: CALL X2MSG - "IF CANNOT BE EXECUTED WITHOUT MATCHING ENDF"
592 2 :ERROR4: CALL X2MSG - "SYSTEM ERROR - NO &SEQTAB"
593 1 END XISCH
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
DEMO.029.LC

029 2.0-10/05/78-08:39 AM
1:BSYM PRINTS, UED083
2:BASG,A PNN-191897#FDSIPDL.
3:USE F.,FDSIPDL.
4:BPCT C.
5:BPCT,FL F.
6:BXT F.POLIST
7:BADD F.COMMON
8:BXT F.POLIST
9:BADD F.DIRECT
10:BXT F.POLIST
11:BADD F.MESAGE
12:BXT F.POLIST
13:BADD F.XUPDL
14:BXT F.POLIST
15:BADD F.XA
16:BXT F.POLIST
17:BADD F.XC
18:BXT F.POLIST
19:BADD F.XD
20:BXT F.POLIST
21:BADD F.XE
22:BXT F.POLIST
23:BADD F.XI
24:BXT F.POLIST
25:BADD F.XL
26:BXT F.POLIST
27:BADD F.XM
28:BXT F.POLIST
29:BADD F.XP
30:BXT F.POLIST
31:BADD F.XR
32:BXT F.POLIST
33:BADD F.XS
34:BXT F.POLIST
35:BADD F.XT
36:BXT F.POLIST
37:BADD F.XU
38:BXT F.POLIST
39:BADD F.XW
40:BXT F.POLIST
41:BADD F.XW
42:BXT F.POLIST
43:BADD F.XZ
44:BXT F.POLIST
45:BADD F.XZI

END 029 - 45 IMAGES PROCESSED.

BSYM PRINTS, UED083

BASG,A PNN-191897#FDSIPDL.

USE F.,FDSIPDL.
### FNM-191897*FDSDPD(1) ELEMENT TABLE

<table>
<thead>
<tr>
<th>NAME</th>
<th>VERSION</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ #</th>
<th>SIZE-PRE</th>
<th>TEXT (CYCLE WORD)</th>
<th>PSRMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:0</td>
<td>1</td>
<td>55</td>
<td>0 0 1</td>
<td>17914</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:35:07</td>
<td>2</td>
<td>30</td>
<td>0 0 0</td>
<td>1815</td>
<td></td>
</tr>
<tr>
<td>SORT2</td>
<td></td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:59:32</td>
<td>3</td>
<td>25</td>
<td>0 0 1</td>
<td>1875</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td></td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>08:37:22</td>
<td>4</td>
<td>24</td>
<td>0 0 1</td>
<td>1932</td>
<td></td>
</tr>
<tr>
<td>XUPDL</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>27</td>
<td>0 0 3</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>22</td>
<td>0 0 2</td>
<td>2103</td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>34</td>
<td>0 0 1</td>
<td>2105</td>
<td></td>
</tr>
<tr>
<td>XC</td>
<td></td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>05:47:05</td>
<td>8</td>
<td>17</td>
<td>0 0 5</td>
<td>2345</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>10 MAR 77</td>
<td>06:19:22</td>
<td>9</td>
<td>39</td>
<td>0 0 3</td>
<td>2494</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td></td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>13:22:28</td>
<td>10</td>
<td>24</td>
<td>0 0 5</td>
<td>2612</td>
<td></td>
</tr>
<tr>
<td>XX</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>24</td>
<td>0 0 5</td>
<td>2613</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>19:37:22</td>
<td>12</td>
<td>24</td>
<td>0 0 5</td>
<td>2614</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>12:48:36</td>
<td>13</td>
<td>24</td>
<td>0 0 5</td>
<td>2615</td>
<td></td>
</tr>
<tr>
<td>XV</td>
<td></td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>23:50:36</td>
<td>14</td>
<td>24</td>
<td>0 0 5</td>
<td>2616</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>0 0 5</td>
<td>2657</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>24</td>
<td>0 0 5</td>
<td>5067</td>
<td></td>
</tr>
<tr>
<td>FSTURD</td>
<td></td>
<td>ELT SYMB</td>
<td>11 MAR 78</td>
<td>03:17:5</td>
<td>17</td>
<td>10</td>
<td>0 0 5</td>
<td>5068</td>
<td></td>
</tr>
<tr>
<td>XP</td>
<td></td>
<td>ELT SYMB</td>
<td>10 APR 78</td>
<td>10:54:29</td>
<td>18</td>
<td>101</td>
<td>0 0 5</td>
<td>5094</td>
<td></td>
</tr>
<tr>
<td>XT</td>
<td></td>
<td>ELT SYMB</td>
<td>10 APR 78</td>
<td>10:54:52</td>
<td>19</td>
<td>34</td>
<td>0 0 5</td>
<td>5193</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td></td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>11:43:48</td>
<td>20</td>
<td>38</td>
<td>0 0 5</td>
<td>5729</td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td></td>
<td>ELT SYMB</td>
<td>10 MAY 78</td>
<td>19:12:58</td>
<td>21</td>
<td>154</td>
<td>0 0 5</td>
<td>6113</td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td></td>
<td>ELT SYMB</td>
<td>10 MAY 78</td>
<td>12:52:31</td>
<td>22</td>
<td>40</td>
<td>0 0 5</td>
<td>7661</td>
<td></td>
</tr>
<tr>
<td>XI</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:39</td>
<td>23</td>
<td>34</td>
<td>0 0 5</td>
<td>8044</td>
<td></td>
</tr>
<tr>
<td>XR</td>
<td></td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:42</td>
<td>24</td>
<td>15</td>
<td>0 0 5</td>
<td>8537</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEp 78</td>
<td>09:46:47</td>
<td>25</td>
<td>16</td>
<td>0 0 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>10</td>
<td>0 0 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>00</td>
<td>0 0 5</td>
<td>10042</td>
<td></td>
</tr>
<tr>
<td>X11</td>
<td></td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:37</td>
<td>28</td>
<td>20</td>
<td>0 0 5</td>
<td>10642</td>
<td></td>
</tr>
</tbody>
</table>

**NEXT AVAILABLE LOCATION**

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

**EXIT F.P. POLIST**