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

(NASA-TM-80841) FLIGHT DESIGN SYSTEM-1 SYSTEM DESIGN DOCUMENT. VOLUME 9: EXECUTIVE LOGIC FLOW, PROGRAM DESIGN LANGUAGE (NASA) 449 p HC A19/MP A01

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

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

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

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

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

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

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

XECOMM

XE - EXECUTIVE FIXED COMMON (GLOBAL)

XECOMM

XB - EXECUTIVE DYNAMIC BLOCK (SUBSTA LEVEL DEPENDENT)

XECOMM

XS - EXECUTIVE SCRATCH SPACE (VOLATILE ACROSS ALL CALLS

XECOMM

TO FOS Routines EXCEPT XR___

XECOMM

******

XECOMM

XE CONTENTS

XECOMM

INTEGER

XECOMM

* CARTCG

XECOMM

* CLASMO

XECOMM

* COMPR

XECOMM

* FLAGS

XECOMM

* FCMNAME(3)

XECOMM

* RESPTR

XECOMM

* SEREND

XECOMM

* SUBSTA

XECOMM

* TOKEMS(32)

XECOMM

* XE

XECOMM

DIRECTIONS

XECOMM

* INTNAM(3)

XECOMM

EQUIVALENCE

XECOMM

* (XE(1), ALU)

XECOMM

* (XE(3), MCL)

XECOMM

* (XE(5), MASTA)

XECOMM

* (XE(7), SENRAM(3))

XECOMM

* (XE(11), SEREND)

XECOMM

* (XE(13), INTNAM(3))

XECOMM

* (XE(19), RESPTR)

XECOMM

* (XE(63), TOKEMS(32))

XECOMM

* (XE(139), EXTEND)

XECOMM

* (XE(141), OLIND)

XECOMM

* (XE(142), CARTAG)

XECOMM

* (XE(143), NOPROC)

XECOMM

* (XE(144), COMPR)

XECOMM

* (XE(145), COMBUF)

XECOMM

CARTAG - NUMBER OF THE DISK CARTRIDGE CONTAINING EXECUTIVE MASTER FILES

XECOMM

CLASMO - EXECUTIVE/MADE Request BLOCK Class I/O NUMBER

XECOMM

COMBUF - TERMINAL COMMUNICATIONS OUTPUT BUFFER

XECOMM

(1) - NUMBER OF TOKENS IN BUFFER

XECOMM

(2) - NUMBER OF USED WORDS IN BUFFER

XECOMM

(3_256) - TOKENS REPRESENTING USER'S RESPONSE

XECOMM

COMPR - POINTER TO Token CURRENTLY BEING PROCESSED FROM COMBUF

XECOMM

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

XECOMM

FLAGS - EXECUTIVE FLAG WORD (O-OFF, I-ON)

XECOMM

BITES 0-10 UNUSED

XECOMM

11 PROCESSOR ON-LINE DEBUG

XECOMM

12 MANAGER ON-LINE DEBUG

XECOMM

13 EXECUTIVE ON-LINE DEBUG

XECOMM

14 PRODUCE A DUMP ON ALL TERMINATIONS

XECOMM

15 - MANAGER REQUEST TRANSACTION Trace FLAG

XECOMM

INTNAM - NAME OF INTERFACE TABLE INPUT TO INTERFACE TABLE EDITOR

XECOMM
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

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

BITS 0-9 - NOT USED
BITS 10-13 - DIRECTIVE CONTROL MODE IF BITS 14-15 = 0
  0 - LIST
  1 - TID
  2 - SAVE
  3 - RECALL
  4 - DELETE
  5 - RENAME
  6 - COPY
  7 - CLEAR
  8 - OFF
  9 - STORE
  10 - RESTORE
  11 - UNLOAD
  12 - LOAD
  13 - BATCH

BIT 11 - EXECUTION CONTROL INITIALIZATION INDICATOR
  IF BITS 14-15 = 1
  0 - INITIALIZATION FROM DIRECTIVE
  1 - INITIALIZATION FOR REENTRY

BITS 12-13 - EXECUTION CONTROL 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

NPROC - NUMBER OF PROCESSES IN LIBRARY

OLIND - OLD INDEX TO CURRENTLY EXECUTING ENTRY IN SEQUENCE TABLE

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

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

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

REPT - POINTER TO END OF LAST COMPLETED 8 WORD ENTRY IN REBUFF (0 INDICATES REBUFF EMPTY) OR RETURN CODE FROM XREQ

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

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

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

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

SOFTWARE 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

* BREAK
* END
* ERROR
* PROMPT
* SPOKE
* TABSIZE
* XLIBD

M4V

* (X4(27), DEBUG)
* (X4(31), ENDTAB)
* (X4(35), HEXAD)
* (X4(36), PRMEN)
* (X4(37), PROMPT)
* (X4(38), INITN)
* (X4(40), INSERT)
* (X4(41), BEGNO)

DIMENSION

* (X4(120), XLIBD)

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

BREAK = TABLE IN XLIBD TO FIRST TABLE ENTRY TO BE LISTED OR DELETED

END = INDEX IN XLIBD TO LAST TABLE ENTRY TO BE LISTED OR DELETED

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

0 => REPLACE ENTRY AT TABMOD
1 => INSERT A NEW ENTRY IN FRONT OF TABMOD
2 => ADD AN ENTRY AT THE BOTTOM OF THE TABLE

INITN = INTERFACE TABLE NAME INPUT OR 0

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

LIBDSIZE = SIZE IN WORDS OF PROCESSOR DIRECTORY, XLIBD

NEWTAB = NAME OF SEQUENCE TABLE BEING CREATED

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

CURRENTLY IN XLIBD

OLDTAB = NAME OF EXISTING SEQUENCE TABLE INPUT TO THE EDITOR

PRMEN = LENGTH IN WORDS OF PROMENT CREATED BY XPROM

PRINTD = CURRENT PROMPTING MODE:

1 => UPDATE MODE ( # )
2 => CREATE OR MISSING MODE ( # 200: )
3 => ALL MODE ( # 200+PROC, TABLE: )

PROMPT = PROMPT BUILT BY XPROM

SPOKE = CURRENT SEQUENCE NUMBER BEING PROMPTED

TABSIZE = SIZE IN WORDS OF TABLE IN XLIBD

TABMOD = INDEX IN XLIBD TO CURRENT TABLE ENTRY

UCINIT = MAXIMUM SIZE IN WORDS OF XLIBD

XLIBD = WORKING BUFFER CONTAINING TABLE DURING EDIT SESSION

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

1 CD - INTEGER
1 CD * ARGO - CURRENT ARGUMENT'S N0. (I.E. 1 TO 64)
1 CD * BITM - BIT M0. IN BIT MASK CURRENTLY BEING PROCESSED
1 CD * BITM0 - BIT M0. IN BIT MASK OF NEXT DIFFERING BIT
1 CD * COMFLG - FLg SET TO 1 IF A CONTINUATION PROMPT FOR CURRENT ARGUMENT
1 CD * CRP - INDEX IN WBUF TO ARgUMENT PROMPT FOR CURRENT ARGUMENT
1 CD * DIRECT - ASCII ARRAY OF VALID INTERFACE TABLE EDIT DIRECTIVES
1 CD * DFLG - FLAG SET TO VALUE OF DATA BIT FOR THIS ARGUMENT
1 CD * DBUG - FLAG SET TO VALUE OF DEBUG FLAG (BITS 13-15 OF XE4))
1 CD * ITYPE - TYPE FLAG FOR THIS ARGUMENT
1 CD * IARG - INDEX IN WBUF TO CURRENT ARg
1 CD * ICCLASS - CLASS OF THIS ARGUMENT
1 CD * ICLASS - CLASS OF THIS ARGUMENT
1 CD * ISIZE - TOTAL SIZE (IN WORDS) FOR THIS ARGUMENT
1 CD * ISIZE - NO. OF WORDS/ELEMENT FOR EACH ALLOWED ITYPE VALUE
1 CD * ISUB - CURRENT EFFECTIVE SUBSCRIPT FOR THIS ARGUMENT
1 CD * IZM - NO. OF ELEMENTS IN Interface TABLE
1 CD * KBES - INDEX IN KBUF TO GBRM TABLE ENTRY
1 CD * KBUF - INTERNAL BUFFER FOR INDEXED USAGE
1 CD * LBFLG - FLAG SET TO I/O BITS FOR THIS ARGUMENT
1 CD * LNOK - GENERAL RETURN FLAG (0=OK, -1=ERROR, 5='EXIT')
1 CD * NEWTAG - NEWTAG FOR THIS ARGUMENT
1 CD * OPTMT - OPTMIZATION LEVEL
1 CD * ORTH - INDEX IN ORTH TO ELEMENT OF OBD
1 CD * PCL - PAGE CLASS FOR THIS ARGUMENT
1 CD * PCOMP - PAGE COMPARISON LEVEL
1 CD * PPTM - PAPER TRAY MOUNTED
1 CD * REND - INDEX IN WBUF TO CURRENT ARGUMENT
1 CD * XTABLE - INTERFACE TABLE TO EDIT
JSUB    - CURRENT SECOND SUBSCRIPT (OR 0) FOR THIS ARGUMENT
LENEFF - NO. OF WORDS/ELEMENT FOR THIS ARGUMENT
LISTLU - LU TO WHICH PRINT SHOULD GO (USED WHEN
LIST DIRECTIVE CALLS XILSD OR XICH)
LITDSP - DISPLAY TO LITERAL DATA FOR THIS ARGUMENT
LITDW - INDEX IN WKBuf TO END OF LITERAL DATA
LITLEN - LENGTH OF LITERAL DATA AREA OF WKBuf
LITPTR - INDEX IN WKBuf TO START OF LITERAL DATA
LITSZ - NUMBER OF ARRAY ELEMENTS FOR THIS ARGUMENT (LOGICAL SIZE)
LISTFLG - 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, PARAME TR OR INCOMPLETE INDICATORS
= 4, CALLED FROM LIST DIRECTIVE (PER CENT PROMPT) TO LIST
A DATA ELEMENT'S VALUE(S)
MODSAV - PREVIOUS VALUE OF PRMTMD WHILE PRMTMD = 4 (CONTINUE)
MARG - INDEX IN WKBuf TO START OF SHORT PROMPTS
MODTM - INDEX IN WKBuf TO BIT MASK WORD(S) FOR THIS ARGUMENT
WNTAB - ASCII NAME OF TABLE BEING GENERATED
NOBITM - NO. OF BIT MASES ASSOCIATED WITH THIS ARGUMENT
NUMARG - NO. OF ARGUMENTS IN THIS INTE. TABLE
NUMDIR - NUMBER OF DIRECTIVES ACTUALLY IN DIRECT
PRMLEN - LENGTH (IN WORDS) OF PROMPT BUILT
PRMTMD - CURRENT PROMPTING MODE
= 1, PROMPT FOR INCOMPLETE ARGUMENTS (MISSING)
= 3, PROMPT WITH A LIST OF CURRENT VALUES
= 4, CONTINUATION OF SAME ARGUMENT
= 5, PROMPT WITH "/" (CREATE MODE)
PROMPT - ASCII ARRAY FOR PROMPT BUILT
SFLAG - FLAG SET TO VALUE OF SUBSCRIPT BIT FOR THIS ARGUMENT
VERSOn - VERSION NO. OF THIS INTERFACE TABLE
WKBLNG - LENGTH OF WKBuf
WKBUF - 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.
4.0 PDL LISTING PROGRAM

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

INPUT
80 COLUMN PDL IMAGES SUBJECT TO THE FOLLOWING CONVENTIONS:
- IN COLUMN 1 INDICATES PAGE EJECT AND THE FIRST TOKEN (6 OR LESS
  CHARACTERS) IS REPRODUCED IN COLUMNS 127-132 OF OUTPUT LISTING
- UNTIL NEXT RECORD IS DETECTED. IF RECORD CONTAINS ONLY THE - THE
  TOKEN FROM THE PREVIOUS RECORD CONTINUES TO APPEAR IN THE
- IDENTIFICATION COLUMNS OF THE OUTPUT
- IN COLUMN 1 INDICATES A COMMENT TO BE COPED 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

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

PDLIST
33 1 BEGIN PDLIST
34 2 INITIALIZE SEQUENCE NUMBER, LEVEL AND DEFINITION TABLE COUNTER
35 3 DO UNTIL END-OF-FILE INPUT
36 4 READ RECORD
37 5 INCREMENT SEQUENCE NUMBER
38 6 IF COLUMN 1 = " "
39 7 THEN
40 8 SET PAGE EJECT IN IMAGE
41 9 IF REMAINDER OF IMAGE IS NOT BLANK
42 10 THEN
43 11 SET ID TO CONTENTS OF FIRST NON-BLANK FIELD
44 12 ENDIF
45 13 ELSE
46 14 CLEAR LEVEL INCREMENT
47 15 IF COLUMN 1 NOT = ",
48 16 THEN
49 17 IF FIRST CHARACTER = :     - INDICATES :LABEL:
50 18 THEN
51 19 SET LINE SKIP IN IMAGE
52 20 GENERATE DEFINITION TABLE ENTRY FOR LABEL
53 21 ELSE
54 22 CALL FSTWD TO GET FIRST WORD OF PDL
55 23 LOOKUP FIRST WORD IN KEY WORD TABLE
56 24 KEY WORD TABLE CONTAINS
57 25
58 26 1 BEGIN   - BEGIN SECTION INDICATOR
59 27 2 IF     - SECTION INDICATOR
60 28 3 DO     - SECTION INDICATOR
61 29 4 DFOR   - SECTION INDICATOR
62 30 5 DOUNTI - SECTION INDICATOR
63 31 6 DOWHIL - SECTION INDICATOR
64 32 7 CASE   - SECTION INDICATOR
65 33 8 START  - SECTION INDICATOR
66 34 9 STARTS - SECTION INDICATOR
67 35 10 ELSE  - SECTION SEPARATOR
68 36 11 THEC  - SECTION SEPARATOR
69 37 12 EXIT  - SECTION SEPARATOR
70 38 13 EXITF - SECTION SEPARATOR
71 39 14 OR    - SECTION SEPARATOR
72 40 15 ORELSE- SECTION SEPARATOR
73 41 16 ENDOLO - SECTION SEPARATOR
74 42 17 END   - END OR END LOOP?
75 43 18 ENDF  - SECTION TERMINATOR
76 44 19 ENDRD - SECTION TERMINATOR
77 45 20 ENDCAS- SECTION TERMINATOR
78 46 21 ENDOSEA- SECTION TERMINATOR
79 47
80 48 IF KEY WORD LOCATED
81 49 THEN
82 50 CASE LOCATION :BEGIN:, :SECOND:, :SECONID:, :SECONID:, :SECONID:, :SECONID:
83 51 (:SECONID:, :SECONID:, :SECONID:, :SECONID:, :SECONID:, :SECONID:
84 52 (:SECONID:, :SECONID:, :SECONID:, :SECONID:, :SECONID:, :SECONID:
86 54 :BEGIN: GENERATE DEFINITION TABLE ENTRY FOR SECTION NAME
87 55 SET LEVEL INCREMENT = 1
88 56 :SECONID: SET LEVEL INCREMENT = 1
89
:SEP: SET LEVEL INCREMENT = 1
DECREMENT LEVEL

:END?: CALL FSTWRD TO GET NEXT WORD OF PDL
DECREMENT LEVEL
IF WORD = LOOP
THEN
SET LEVEL INCREMENT = 1
ENDIF

:TERM: DECREMENT LEVEL
ENDCASE
ENDIF
ENDIF

COMPUTE INDENTATION FACTOR = MINIMUM OF 3(LEVEL-1) AND 36
ELSE
SET INDENTATION FACTOR = 1
ENDIF

CONSTRUCT OUTPUT IMAGE FROM SEQUENCE NUMBER, LEVEL, INDENTATION FACTOR,
INPUT RECORD AND ID FIELD
OUTPUT IMAGE
APPLY LEVEL INCREMENT
CLEAR PAGE EJECT

ENDIF
ENDDO
CALL SORT1 TO ORDER DEFINITION TABLE
OUTPUT DEFINITION TABLE
ENDDO POLIST
116  1 C01                        EXTRACT THE FIRST TOKEN FROM A PDL RECORD
117  1 C01
118  1 C01
119  1 C02                  INPUT
120  1 C02
121  1 C02
122  1 C02
123  1 C02
124  1 C02
125  1 C02
126  1 C02
127  1 C02
128  1 C02
129  1 *
130  1 *
131  1 *
132  1 *
133  1 BEGIN FSTWRD
134  2  BLANK OUTPUT WORD
135  2  LOCATE FIRST NON-BLANK CHARACTER
136  2  DO UNTIL SIX CHARACTERS STORED OR END-OF-RECORD
137  3  IF CHARACTER IS NON-BLANK AND NON-:
138  3    THEN
139  4  STORE CHARACTER
140  3    ELSE
141  3    EXIT DO
142  3    ENDIF
143  2  EDDO
144  2  RETURN LOCATION
145  1 END FSTWRD
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DEFINITION</th>
<th>TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>
5.0 FDS EXECUTIVE DETAILED LOGIC FLOW

A directory listing the major programs and subroutines in alphabetical order is presented initially. The detailed logic flow of each then follows in alphabetical order.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>* XPAR  USER RESPONSE PROCESSOR</td>
</tr>
<tr>
<td>62</td>
<td>* XPAR  OVERLAY INTERFACE ROUTINE</td>
</tr>
<tr>
<td>63</td>
<td>* XPAR  PARAMETER DATA PROCESSOR</td>
</tr>
<tr>
<td>64</td>
<td>* XPAR  PROMPT DIRECTIVE PROCESSOR</td>
</tr>
<tr>
<td>65</td>
<td>* XPAR  PROMPT CONSTRUCTOR</td>
</tr>
<tr>
<td>66</td>
<td>* XPAR  SUBSCRIPT PROCESSOR</td>
</tr>
<tr>
<td>67</td>
<td>* XPAR  LIBRARY MAINTENANCE PROGRAM</td>
</tr>
<tr>
<td>68</td>
<td>* XPAR  CREATES A HDD/PDD DATA BASE FILE</td>
</tr>
<tr>
<td>69</td>
<td>* XPAR  MAIN DATA BASE FILE SEGMENT</td>
</tr>
<tr>
<td>70</td>
<td>* XPAR  DELETES A PROCESSOR FROM LIBRARY DIRECTORY</td>
</tr>
<tr>
<td>71</td>
<td>* XPAR  CREATES A DEFAULT INTERFACE TABLE</td>
</tr>
<tr>
<td>72</td>
<td>* XPAR  DISPLAYS PARAMETER SPECIFICATIONS INSTRUCTION</td>
</tr>
<tr>
<td>73</td>
<td>* XPAR  MAIN LOGIC FOR DEFAULT VALUES</td>
</tr>
<tr>
<td>74</td>
<td>* XPAR  MAIN MAINTENANCE PROGRAM</td>
</tr>
<tr>
<td>75</td>
<td>* XPAR  MODIFIES A PROCESSOR INTERFACE TABLE</td>
</tr>
<tr>
<td>76</td>
<td>* XPAR  MAINTAINS FBS MESSAGE FILE</td>
</tr>
<tr>
<td>77</td>
<td>* XPAR  CREATES HDD/PDD LOG FILE</td>
</tr>
<tr>
<td>78</td>
<td>* XPAR  CREATES A PROCESSOR PROMPT TABLE</td>
</tr>
<tr>
<td>79</td>
<td>* XPAR  MODIFIES HDD/PDD LOG FILE</td>
</tr>
<tr>
<td>80</td>
<td>* XPAR  CREATES SYSTEM PROMPT FILES</td>
</tr>
<tr>
<td>81</td>
<td>* XPAR  ADD A PROCESSOR TO LIBRARY DIRECTORY AND ADDS DEFAULT INTERFACE TABLE</td>
</tr>
<tr>
<td>82</td>
<td>* XPAR  INTERFACE TABLE AND PROMPT FILE</td>
</tr>
<tr>
<td>83</td>
<td>* XPAR  ADDS SPEC FOR DEFAULT INTERFACE TABLE</td>
</tr>
<tr>
<td>84</td>
<td>* XPAR  MANAGER</td>
</tr>
<tr>
<td>85</td>
<td>* XPAR  DMA Free Space</td>
</tr>
<tr>
<td>86</td>
<td>* XPAR  DMA GET SPACE</td>
</tr>
<tr>
<td>87</td>
<td>* XPAR  DMA MANAGEMENT (XMPK) EPC</td>
</tr>
<tr>
<td>88</td>
<td>* XPAR  GENERAL PTF INTERFACE (XMPK) EPC</td>
</tr>
<tr>
<td>89</td>
<td>* XPAR  DMA AND CONTROL DATA MAPPING</td>
</tr>
<tr>
<td>90</td>
<td>* XPAR  DMA ALLOCATION (XMPK) EPC</td>
</tr>
<tr>
<td>91</td>
<td>* XPAR  DMA DELOCATION (XMPK) EPC</td>
</tr>
<tr>
<td>92</td>
<td>* XPAR  DMA INITIALIZATION (XMPK) EPC</td>
</tr>
<tr>
<td>93</td>
<td>* XPAR  DMA RETRIEVE (READ) (XMPK) EPC</td>
</tr>
<tr>
<td>94</td>
<td>* XPAR  DMA STORE (WRITE) (XMPK) EPC</td>
</tr>
<tr>
<td>95</td>
<td>* XPAR  DMA MANAGEMENT</td>
</tr>
<tr>
<td>96</td>
<td>* XPAR  FBS MANAGER</td>
</tr>
<tr>
<td>97</td>
<td>* XPAR  POST AND WAIT INTERFACE ROUTINE (XMPK) EPC</td>
</tr>
<tr>
<td>98</td>
<td>* XPAR  DMA COMPRESS</td>
</tr>
<tr>
<td>99</td>
<td>* XPAR  PHASE 1 COMPRESS (PURGE TO DMA) (XMPK) EPC</td>
</tr>
<tr>
<td>100</td>
<td>* XPAR  PHASE 2 COMPRESS (TOC COMPRESS) (XMPK) EPC</td>
</tr>
<tr>
<td>101</td>
<td>* XPAR  PHASE 3 COMPRESS (SPACE REORDER) (XMPK) EPC</td>
</tr>
<tr>
<td>102</td>
<td>* XPAR  TOC LOOK-UP ROUTINE</td>
</tr>
<tr>
<td>103</td>
<td>* XPAR  SEQUENCE LOCATION COUNTER RESET</td>
</tr>
<tr>
<td>104</td>
<td>* XPAR  SEQUENCE TABLE EXECUTION</td>
</tr>
<tr>
<td>105</td>
<td>* XPAR  PROCESSOR SERVICES</td>
</tr>
<tr>
<td>106</td>
<td>* XPAR  PROCESSOR PARAMETER ATTRIBUTE RETRIEVAL</td>
</tr>
<tr>
<td>107</td>
<td>* XPAR  PROCESSOR PARAMETER RETRIEVAL</td>
</tr>
<tr>
<td>108</td>
<td>* XPAR  PROTOẩm PARAMETER STORAGE (XMPK) EPC</td>
</tr>
<tr>
<td>109</td>
<td>* XPAR  PROTO资产重组 PARAMETER STORAGE (XMPK) EPC</td>
</tr>
<tr>
<td>110</td>
<td>* XPAR  PROMPT USER,READS RESPONSE,RETURNS ENCODED BUFFER</td>
</tr>
<tr>
<td>111</td>
<td>* XPAR  CALLS XPAR,READS BUFFER,RETURNS RESPONSE IN DATA AREA</td>
</tr>
<tr>
<td>112</td>
<td>* XPAR  PROCESSOR SERVICE FOR ANA ACCESS</td>
</tr>
<tr>
<td>113</td>
<td>* XPAR  PROCESSOR TERMINATION ROUTINE</td>
</tr>
<tr>
<td>114</td>
<td>* XPAR  EXECUTIVE SERVICES</td>
</tr>
</tbody>
</table>

**Notes:**
- The above list represents a portion of the document, focusing on specific entries related to XPAR commands and their functionalities.
- The entries are listed in a sequential manner, indicating the order of the commands or functions.
- The document appears to be a listing of system commands, with each entry detailing the function or purpose of a particular command or function within a system context.
120 1  XBIT     MULTIPLE WORD BIT STRING BIT CLEAR/SET
121 1  * XCPRT   COMPARE ARRAYS
122 1  * XRDB    DOUBLE PRECISION TO ASCII CONVERSION
123 1  * XEQ     ANA MANAGEMENT REQUEST ROUTINE
124 1  * XEXTRACT EXTRACTS A VARIABLE LENGTH FIELD FROM A WORD
125 1  * XFA14    CONVERT A WORD TO ASCII IN FP14 FORMAT
126 1  * XFA16    CONVERT A WORD TO ASCII IN FP16 FORMAT
127 1  * XLCLK    XLSB RH LOCK
128 1  * XLDC    RETURN 16-BIT ADDRESS OF ARGUMENT
129 1  * XMOV    MOVES WORDS FROM ARRAY1 TO ARRAY2
130 1  * XMS6    FDS EXECUTIVE MESSAGE ROUTINE
131 1  * XMSB    MULTIPLE WORD BIT STRING BIT SEARCH
132 1  * XPAGE   CONVERT A WORD TO ASCII IN FP6 FORMAT
133 1  * XPACK    PACKS CHARACTERS FROM R1 TO A2 FORMAT
134 1  * XSHFT    FILE NAME QUALIFICATION
135 1  * XSET     SETS A VARIABLE LENGTH FIELD INTO A WORD
136 1  * XSHIFT   SHIFT A WORD LEFT LOGICALLY
137 1  * XSFTR    SHIFT A WORD RIGHT LOGICALLY (XSFSL EP)
138 1  * XSLK    XLSB RH UNLOCK (XLCLK EP)
139 1  * XSVN    FILE NAME DEQUALIFY
140 1  * XUPK     REMOVES BLANKS AND UNPACKS FROM A2 TO R1 FORMAT
141 1  * XUPTR    REMOVE DUPLICATE BLANKS FROM A2 STRING
142 1  * XS
143 1  * XSEQ     SEQUENCE TABLE EDITOR
144 1  * XSCAN    SEQUENCE TABLE EDITOR (STE) DIRECTIVE SCANNER
145 1  * XSDEL    STE DELETE DIRECTIVE PROCESSOR
146 1  * XSSET    STE ENTRY PROCESSOR
147 1  * XSET     SEQUENCE TABLE EDITOR MAIN ROUTINE
148 1  * XLS    STE LIST DIRECTIVE PROCESSOR
149 1  * XLST     SEQUENCE TABLE LIST ROUTINE
150 1  * XNPUT   STE INPUT PROCESSOR
151 1  * XNMP     STE NUMBER DIRECTIVE PROCESSOR
152 1  * XPC    STE TABLE COMPACTER
153 1  * XPR    STE PROMPT DIRECTIVE PROCESSOR
154 1  * XPRM    STE PROMPT CONSTRUCTOR
155 1  * XT
156 1  * XTCOM    TERMINAL COMMUNICATIONS
157 1  * XTDE    PROMPTS USER, READS RESPONSE, CALLS XTLAN AND XTPRM
158 1  * XTDE    XTLAN CONVERTS ASCII USER'S RESPONSE TO TOKENS
159 1  * XTPRM   HANDLES EXTENDED PROMPTING REQUESTS
160 1  * XU
161 1  * XU     UTILITY (SOFTWARE AIDS)
162 1  * XUDBG   ON-LINE SNAP AND MEMORY MODIFICATION ROUTINE
163 1  * XUDMP   SYSTEM RESIDENT PARTITION DUMP (XVABN EP)
164 1  * XDMP    FILE MANAGER FILE DUMP PROGRAM
165 1  * XDPL    OCTAL AND ASCII DUMP LINE FORMAT
166 1  * XUTF    DUMP FORMATTER
167 1  * XV
168 1  * XV     SYSTEM SERVICES
169 1  * XVABN   FDS ABEND (SEE XVMP)
170 1  * XVAN    FDS COMMUNICATION SERVICES (POST AND WAIT)
171 1  * XX      EXECUTION CONTROL
172 1  * XXAUT   AUTOMATIC MODE
173 1  * XXCNT   EXECUTION CONTROL MAIN PROGRAM
174 1  * XXDEC   DECODES USER RESPONSE IN MANU AND SEMI
175 1  * XXDEF   READS IN DEFAULT INTERFACE TABLE IF NEEDED
176 1  * XXEDE   EXECUTES ASERTAB AND HANDLES ERROR CONDITIONS
177 1  * XXMAN   MANUAL MODE
179 1  XZSIN  SEMI - AUTOMATIC NODE
180 1  XZSTO  STORE SEQUENCE TABLE IN ASERTAB
181 1  XZTRP  TEMPORARY EXECUTION OF ONE ENTRY WITH BINTAB
182 1  *  UTILITY PROCESSORS
183 1  *  XZ  ASSIGN PROCESSOR
184 1  *  DBDSP  DATA BOX DISPLAY PROCESSOR
185 1  *  DEFIF  DEFINE PROCESSOR
186 1  *  DO  CONDITIONAL LOOP IN SEQUENCE TABLE
187 1  *  ELSE  EXECUTION POINT FOR FALSE IF CONDITION
188 1  *  ENDF  TERMINATES AN IF STRUCTURE
189 1  *  ENDOD  TERMINATES A DO LOOP STRUCTURE
190 1  *  ENDSX  END SCAN PROCESSOR
191 1  *  IF  CONDITIONAL EXECUTION OF SEQUENCE TABLE ENTRIES
192 1  *  SCAN  SCAN PROCESSOR
193 1  *  XICHR  CHARACTER OBJECT STORE FOR ASSGN
194 1  *  XZDFT  FIND ANY TOKEN IN A SYMBOLIC STRING
195 1  *  XZDIN  DATA BOX DISPLAY INPUT PROCESSOR
196 1  *  XZDNK  DATA BOX DISPLAY CONSTRAINT MASKER
197 1  *  XZDOT  DATA BOX DISPLAY OUTPUT ROUTINE
198 1  *  XZDOP  DATA BOX DISPLAY PASS 1 PROCESSOR
199 1  *  XZDOP2  DATA BOX DISPLAY PASS 2 PROCESSOR
200 1  *  XZEVL  PERFORMS EVALUATION BETWEEN TWO REAL NUMBERS
201 1  *  XZFCL  FIND PROCESSOR CLASS NUMBER
202 1  *  XZDFC  FUNCTIONAL OPERATIONS FOR ASSGN
203 1  *  XZFMX  FREE OBJECT STORE FOR ASSGN
204 1  *  XZFSX  FIXED OBJECT STORE FOR ASSGN
205 1  *  XZISX  REMOVE DUPL. BLANKS & BLANK FILL
206 1  *  XZLSS  SYMBOLIC STRING SYNTAX ERROR LISTER
207 1  *  XZMSG  FRS PROCESSOR MESSAGE ROUTINE
208 1  *  XZMPR  MATH OPERATIONS FOR ASSGN
209 1  *  XZPES  DATA CONVERSION AND STORAGE FOR ASSGN
210 1  *  XZPSX  PASS 1 SUBROUTINE FOR ASSGN PROCESSOR
211 1  *  XZPS2  PASS 2 SUBROUTINE FOR ASSGN PROCESSOR
212 1  *  XZRET  DATA RETRIEVAL FOR ASSGN
213 1  *  XZSCF  SEARCHES SEQUENCE TABLE FOR IF STRUCTURES
214 1  *  XZSYM  SYMBOL TABLE INTERFACE FOR ASSGN
215 1  *  XZSYT  SYMBOL TABLE MAINTENANCE
216 1  *
XATIN FUNCTION

FDG ATTENTION TASK

USER MAY REQUEST THE CURRENT STATUS OF FDG
FOR THE TERMINAL, TO TERMINATE CURRENT PROCESSOR,
OR IF FDG HAS TERMINATED, TO TEAR DOWN THE
FDG STRUCTURE FOR THIS TERMINAL.

NOTES

THE ID FOR XATIN IS CONNECTED TO THE USER'S
 TERMINAL AT FDG SIGN-ON BY USING THE ENT
FOR THE DEVICE. THE FUNCTION IS DISCONNECTED
AT FDG SIGN-OFF.
SAVE EOT ADDRESS(IN BREG ON ENTRY)
CALL ERLU(BREG) GET LU IN ASCII & BINARY
STARTSEARCH UNTIL LAST STATUS TABLE ENTRY
EXITIF STBLU EN LU
SET STB ENTRY ADDRESS
ENDLOOP
SET STB ENTRY TO ZERO
ENDSEARCH
IF STB ENTRY FOUND, THEN
GET MANAGER'S ID ADDRESS(STMG)
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 SLIRR DISABLE
IF STBE=EXECUTIVE ADDRESS .NE. 0, THEN
IF STAT(CURRENT) .NE. STBE, THEN
IF CURRENT AT IS NOT DORMANT AND BACK CHAIN POINTS TO OLD XHCR, THEN
FIND BOTTOM AT
DO UNTIL NEXT-AT .EQ. STMG(MANGER)
CALCULATE NEXT-AT FROM BOTTOM'S FATHER ID NUMBER
CLEAR BOTTOM'S WAIT BIT & FATHER ID NUMBER
CLEAR NEXT-AT'S PARM (P1)
CALL SLIRR ENABLE
CALL MESS 'OFF,BOTTOM'
CALL SLIRG DISABLE
SET BOTTOM TO NEXT-AT
ENDIF
ENDIF
CALL SLIRR MAKE EXEC DORMANT
CLEAR EXEC'S ID & STBE
ENDIF
DECRENENT NUMBER ACTIVE(STBAC)
GET EOT ADDRESS
RESTORE INTERRUPT HANDLER(FROM STBREG)
CLEAR STBE
CLEAR MANAGER'S ID, STMG, & STBLU 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(S), STATUS(S), OR RETURN(BLANK)'
READ (LU) REQUEST
IF REQUEST IS KILL OR S, THEN
PERFORM XAKILL
ELSE
IF REQUEST IS STATUS OR S, THEN
PERFORM XASTAT
ENDIF
ENDIF
ELSE
WRITE '***XAR- ERROR LU IS NOT SIGNED-ON TO FDS'
ENDIF
WRITE '"**XA04 FDS ATTENTION FUNCTION TERMINATING'
END
77  1 IF:GIN XSTAT  PRODUCE A FDS STATUS REPORT
78  2  CALL SLIBR - DISABLE
79  3  GET CURRENT-TIME FROM STIME
80  4  GET MANAGER'S ADDRESS FROM STMGB
81  5  MOVE NAME, STATUS, PARTITION, & PRIORITY
82  6  GET EXECUTIVE'S ADDRESS FROM XEBEX
83  7  MOVE NAME, STATUS, PARTITION, & PRIORITY
84  8  GET CURRENT AT FROM STBAT
85  9  MOVE NAME, STATUS, PARTITION, & PRIORITY
86 10  PERFORM XAUNIT(CURRENT) FIND BOTTOM AT
87 11  SET BOTTOM TO CURRENT
88 12  GO WHILE FATHER-ID NE ZERO
89 13  SET FATHER-ID FROM CURRENT
90 14  CALCULATE NEXT
91 15  IF MAX ENTRIES HAVE NOT BEEN PROCESSED, THEN USE NEXT TO
92 16  MOVE NAME, STATUS, PARTITION, & PRIORITY
93 17  ENDIF
94 18  IF NEXT IS THE MANAGER, THEN
95 19  4  SET CURRENT AS TOP
96 20  ENDIF
97 21  SET CURRENT TO NEXT
98 22  ENDDO
99 23  IF TOP EQ ZERO, THEN
100 24  USE TOP TO CURRENT
101 25  USE TOP TO MOVE NAME, PARTITION, & PRIORITY
102 26  SET STATUS TO 'IN USE' OCTAL 17
103 27  ENDF
104 28  CALL SLIBX ENABLE
105 29  WRITE FIRST SET OF HEADERS
106 29  SET TOP AS REPORT DATA
107 29  WRITE REPORT LINE
108 29  SET MANAGER AS REPORT DATA
109 29  WRITE REPORT LINE
110 29  SET EXECUTIVE AS REPORT DATA
111 29  WRITE REPORT LINE
112 29  SET CURRENT AS REPORT DATA
113 29  WRITE REPORT LINE
114 29  WRITE INTERMEDIATE HEADERS
115 29  DO UNTIL MAX ENTRIES OR NO MORE DATA
116 29  WRITE REPORT LINE
117 29  SET NEXT REPORT DATA
118 29  ENDDO
119 1  END XSTAT
120 1  SAMPLE REPORT  ***********************
121 1  #  FDS STATUS FOR LU 10 HN:MM:SS 360
122 1  #  NAME  PRIOR PART#  STATUS
123 1  #  TOP AT-  PROC 922  5 GENERAL WAIT
124 1  #  MANAGER-  XGNXN 40  3 GENERAL WAIT
125 1  #  EXECUTIVE-  XEXXN 80  3 GENERAL WAIT
126 1  #  CURRENT AT-  PROC 11311  4 GENERAL WAIT
127 1  #  BACK CHAIN (UP TO 8) FROM BOTTOM VIA FATHER-ID
128 1  #  PROC 32767  6 DISC ALLOCATE SUSPEND
129 1  #  PROC 11311  4 GENERAL WAIT
130 1  #  PROC 20600  3 GENERAL WAIT
131 1  #  PROC 843  6 GENERAL WAIT
BEGIN XAKILL
140 1 BEGIN XKILL
141 2 TERMINATE CURRENT FDS FUNCTION
142 3 IF MANAGER IS ACTIVE- SET FLAG FOR SEQUENCE TERMINATION
143 4 ON NEXT RETURN VIA A PAN
144 5 IF THE EXEC IS ACTIVE-DO NOTHING
145 6 IF A PROCESSOR IS ACTIVE- USE RTE MESS TO OFF THE PROCESSOR
146 7 SET MANAGER'S ID ADDRESS(STMGR)
147 8 IF STATUS OF MANAGER IS NOT WAIT, THEN
148 9 SET TERMINATE FLAG IN STM-ENTRY
149 10 WRITE "***XAO5 FDS MANAGER SIGNAL TO TERMINATE SEQUENCE"
150 11 ELSE
151 12 IF CURRENT(STBAT) EQ EXEC(STBEX), THEN
152 13 WRITE "***XAO6 FDS EXECUTIVE ACTIVE; NO ACTION TAKEN"
153 14 ELSE
154 15 PERFORM XAKTH(CURRENT) FIND BOTTOM AT
155 16 IF BOTTOM AT IS D.RTR OR SNP THEN
156 17 WRITE "***XAO8 MANAGER IS WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
157 18 EXIT XAKIL
158 19 ELSE
160 20 IF RETURNED BOTTOM IS MANAGER THEN
161 21 IF MANAGER IS NOT WAITING ON A PROGRAM THEN
162 22 WRITE "***XAO8 MANAGER WAITING FOR SYSTEM RESOURCES...NO ACTION TAKEN."
163 23 EXIT XAKILL
164 24 ELSE
165 25 SET RETURN PARAMETER TO PROCESSOR ABENDED
166 26 INCREMENT MANAGER SUSPEND ADDRESS PAST SCHEDULE OF PROCESSOR
167 27 CALL BLIST TO REACTIVATE MANAGER
168 28 ENDF
169 29 ENDF
170 30 WRITE "***XAO7 FDS PROCESSOR 'NAME' SCHEDULED TO ABORT."
171 31 IF RETURNED BOTTOM WAS NOT MANAGER THEN
172 32 SET NAME IN 'OFF' COMMAND
173 33 CALL MESS TO 'OFF' THE PROCESSOR
174 34 ENDF
175 35 ENDF
176 36 ENDF
177 37 END XAKILL
179 1 BEGIN XABTM  FIND BOTTOM AT
180 2 DO WHILE CURRENT IS IN GENERAL WAIT,
181 3 AND WAIT POINTER(P) HAS A SON ADDRESS,
182 4 AND SONS FATHER ID POINTS TO CURRENT
183 5 SET SON AS CURRENT
184 6 ENDDO
185 2 SET BOTTOM AS CURRENT
186 1 END XABTM
BEGIN XCNFG

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

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
29 2 * PARCH=LU,IO,DMA SIZE,PARK(ON OR OFF),OPTIONS
30 2 * SWITCH INPUT PARCH AROUND SO THAT
31 2 * NOW PARCH=LU,P2(ON OR OFF),IO,OPTS,DMA SIZE
32 2 * FOR COMPATIBILITY TO BUILD 1.
33 CALL RMPAR(PARCH)
34 2 IF LU IS .LT. 0, OR
35 3 .GT. LUNAM(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 4 1 PERFORM XCOFF SIGN OFF
42 4 ELSE
43 4 1 PERFORM XCONF SIGN ON
44 2 ENDIF
45 2 ENDIF
46 2 EXEC
47 2 CALL EXEC PROGRAM TERMINATION
48 1 END XCONF
BEGIN XCON

1  * SIGN ON A USER TO FDS
2  UNTIL VALID USER ID (P3)
3  IF ID NOT A - 2 - THEN
4  WRITE 'XCOB ENTER VALID ID (A - Z)'
5  READ RESPONSE
6  ENDDO
7  IF USER ID IS BEING USED, THEN
8  WRITE '***XCOB LU 'LU' IS CURRENTLY USING ID 'ID'- SIGN ON REJECTED'
9  EXIT :XCETA
10  IF FDS RESOURCE NUMBER NOT DEFINED, THEN
11  CALL RMGR (GLOBAL ALLOCATE, LOCAL SET)
12  ELSE
13      CALL RMGR (LOCAL SET)
14      IF NUMBER ALLOCATED (STBAC) .LE. MAXIMUM USERS(STBAM), THEN
15          ISSUE MESSAGE '***XCOB FDS CURRENTLY AT MAX USER'S.
16      ELSE
17          DO FOR STBAN (NUMBER OF FDS ENTRIES)
18              IF ENTRY'S LU (STBBLU) .EQ. REQUESTING LU (P1), THEN
19                  ISSUE MESSAGE '***XCOB 'LU' IS ALREADY SIGNED ON TO FDS'
20                  EXIT :XCETA
21          ELSE
22              IF THIS ENTRY IS AVAILABLE, THEN
23                  SET AS CURRENT-ENTRY-ADDRESS
24                  ENDDO
25          ENDDO
26      ENDDO
27      BECOME PRIVILEGED & DISABLED
28      CALL SLIBR
29      STARTSEARCH WHILE NUMBER-FOUND .LT. NUMBER-NEEEDED
30      SEARCH ID-SEGMENTS USING KEYID(1657)
31      IF XEXEC NOT FOUND AND THIS ID .EQ. XEXEC, THEN
32          SET ID ADDRESS OF XEXEC
33          INCREMENT NUMBER-FOUND
34      ELSE
35          IF XMGF NOT FOUND AND THIS ID .EQ. XMGF, THEN
36              SET ID ADDRESS OF XMGF
37              INCREMENT NUMBER-FOUND
38          ELSE
39              IF XATM NOT FOUND AND THIS ID .EQ. XATM, THEN
40                  SET ID ADDRESS OF XATM
41                  INCREMENT NUMBER-FOUND
42              ELSE
43                  IF FIRST-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
44                      SET ID ADDRESS OF FIRST-BLANK
45                      INCREMENT NUMBER-FOUND
46              ELSE
47                  IF SECOND-BLANK NOT FOUND AND THIS IS A BLANK ID, THEN
48                      SET ID ADDRESS OF SECOND-BLANK
49                      INCREMENT COUNT
50                  ENDDO
51              ENDDO
52          ENDDO
53          WRITE ENDDO
54          WRITE ENDDO
55          EXIT IF THERE ARE NO MORE IDS
CALL BLIXX ENABLE
 ISSUE MESSAGES "**XCO? CANNOT FIND 'NAME' ID-SIGNON TERMINATED"
ENDLOOP

BUILD ENTRY IN YSTD
SET LU EXIT STBLU
SET LU EXIT ASCII INTO STBLA
SET USER'S ID INTO STID
SET ADDRESS OF FIRST-BLANK INTO STBLK
SET ADDRESS OF SECOND-BLANK INTO STBLX
INCREMENT ACTIVE COUNT(STBLR)
BUILD XGMNN & XEXNN
MOVE PRIORITY THRU DISC ADDRESS FROM XGR TO FIRST-BLANK
TURN ON TN BIT
SET NAME TO XGMNN
MOVE PRIORITY THRU DISC ADDRESS FROM XEXEC TO SECOND-BLANK
TURN ON TN BIT
SET NAME TO XEXNN
LINK ATTENTION FUNCTION TO THE USER
DO FOR ANY TERMINAL EXCEPT SYSTEM'S CONSOLE
CALCULATE ERT OVERLAY
SAVE ERT VALUE IN STBR
SET ID ADDRESS OF XATTN INTO EGT
ENDIF
SET INPUT PARMS INTO ID OF XGMNN
SCHEDULE XGMNN VIA BLIST
CALL BLIST
CALL BLIXX ENABLE
IF FDS HAS A FATHER, THEN
CALL MESSAGE 'OFF,FATHER'
ENDIF
ISSUE MESSAGE "***XCO1 LU 'MM' SIGNED ON TO FDS"
ENDSEARCH

:XECTA
CALL RNMR (LOCAL CLEAR)
ENDIF
CALL EXEC TERMINATE
END XCON
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
40 1 #00  FORTRAN CALLING PROCEDURE
41 1 #00  CALL XELDS (XDLF)
42 1 #00
43 1 #00
44 1 #01
45 1 #01  XDLF DETERMINES WHICH OF THE FOLLOWING DIRECTIVES WAS
46 1 #01  REQUESTED AND CALLS THE APPROPRIATE HANDLER SUBROUTINE
47 1 #01  TOC
48 1 #01  SAVE
49 1 #01  RECALL
50 1 #01  DELETE
51 1 #01  RENAME
52 1 #01  COPY
53 1 #01  CLEAR
54 1 #01  OFF
55 1 #01
56 1 #02
57 1 #02  INPUT
58 1 #02  XE COMMON - MASSTA (BITS 10-13 CONTAIN 1 INDEX INTO A LIST OF
59 1 #02  DIRECTIVES)
60 1 #02
61 1 #02
62 1 #04  INTERNAL VARIABLES
63 1 #04  LIST - ORDERED LIST OF APPROPRIATE HANDLER ADDRESSES
64 1 #04
65 1 #04
66 1 #05
67 1 #05  NOTES
68 1 #05  USES "ENTR, XDLE, XDCOP, XDELE, XDFF, XDREC, XDREM, XDSAY,
69 1 #05  XDOTO, XERTH"
70 1 #05
71 1 #05  XDLF IS DESIGNED TO BE THE MAIN ROUTINE FOR THE OVERLAY SEGMENT
72 1 #05  CONTAINING THE REFERENCED DIRECTIVES
73 1 #05
74 1 *
75 1 *
76 1 *
77 1 BEGIN XDLF
78 2  EXTRACT DIRECTIVE INDEX FROM MASSTA AND DECREMENT BY 1
79 2  CASE (TOC, SAVE, RECA, DELE, RENAME, COPY, CLEAR, OFF) INDEX
80 3  .TOC: CALL XDOTO
81 3  .SAVE: CALL XDSSAV
82 3  .RECA: CALL XDREC
83 3  .DELE: CALL XDDELE
84 3  .RENAME: CALL XDREN
85 3  .COPY: CALL XDCOP
86 3  .CLEAR: CALL XDCLE
87 3  .OFF: CALL XDFF
88 2  ENDCASE
89 2  CALL XERTH TO RETURN FROM SEGMENT
90 1 END XDLF
*DIMENSION OF THE ORIGINAL PAGE IS POOR*
188 1 BEGIN XCEL
189 2 RETRIEVE TOC
190 3 BUILD REQUEST TO CLEAR AWA
191 4 DO UNTIL END OF PERMANENT SYSTEM TABLES CHAIN (CHAIN 1)
192 5 EXIT TO :ERR24; IF CHAIN POINTS BEYOND TOC BUFFER
193 6 EXIT TO :ERR48; IF RESTORATION REQUEST QUEUE IS FULL
194 7 BUILD REQUEST TO REALLOCATE TABLE
195 8 REQUEST MANAGER TO RETRIEVE TABLE (HOLD IN SAM)
196 9 BUILD REQUEST TO STORE TABLE INTO AWA FROM SAM
197 10 ENDDO
198 11 DO UNTIL END OF DATA BASE FILES CHAIN (CHAIN 8)
199 12 EXIT TO :ERR24; IF CHAIN POINTS BEYOND TOC BUFFER
200 13 EXIT TO :ERR48; IF RESTORATION REQUEST QUEUE IS FULL
201 14 BUILD REQUEST TO REALLOCATE TOC ENTRY
202 15 ENDDO
203 16 BUILD REQUEST TO TERMINATE LIST
204 17 DO UNTIL END OF DRDE CHAIN (CHAIN 3)
205 18 IF CHAIN POINTS BEYOND TOC BUFFER
206 19 THEN
207 20 OUTPUT XD13 'TOC TOO LARGE, DRDE PURGE INCOMPLETE'
208 21 EXIT PURGE LOOP
209 22 ENDF
210 23 PURGE FILE
211 24 ENDDO
212 25 DO FOR EACH BLOCK OF EIGHT REQUESTS
213 26 TRANSMIT BLOCK TO MANAGER
214 27 EXIT TO :ERR23; IF REQUESTS FAILED
215 28 ENDDO
216 29 EXIT XCEL
217 30 :ERR23:
218 31 DO FROM FAILING REQUEST TO END OF LIST
219 32 IF REQUEST TO STORE
220 33 THEN
221 34 READ SAM TO FREE BUFFER AND CLASS NUMBER
222 35 ENDF
223 36 ENDDO
224 37 DO UNTIL END OF DATA BASE FILE CHAIN (CHAIN 8)
225 38 IF FILE IS UTDB (TYPE 1)
226 39 THEN
227 40 CALL PURGE TO DELETE FILE
228 41 ENDF
229 42 ENDDO
230 43 PURGE ALL UTDB FILES
231 44 TERMINATE FDS WITH CLEAR FAILURE MESSAGE
232 45 :ERR48:
233 46 :ERR24:
234 47 DO FOR ALL STORE REQUESTS BUILT
235 48 READ SAM TO FREE BUFFER AND CLASS NUMBER
236 49 ENDDO
237 50 EXIT XCEL WITH CLEAR FAILURE MESSAGE
238 51 END XCEL
1 BEGIN XDCOP
2 EXIT TO :SYNTAX: IF FIRST TOKENS ARE NOT "NAME"
3 SAVE POINTER TO CURRENT NAME
4 IF NEXT TOKEN IS HYPHEN, THEN
5 DECODE CLASS NAME
6 EXIT TO :CLASSER: IF CLASS SPECIFIED IS NOT VALID (S,I,D,F,B)
7 ELSE
8 SET CLASS TO DATA ELEMENT
9 ENDIF
10 EXIT TO :SYNTAX: IF NEXT TOKENS ARE NOT "NAME"
11 SAVE POINTER TO NEW NAME
12 EXIT TO :SYNTAX: IF NEXT TOKEN IS NOT END-OF-MESSAGE
13 IF CLASS IS DATA BASE OR ORDE, THEN
14 EXIT IF NEW NAME IS MORE THAN 4 CHARACTERS :NAME ERR:
15 IF CLASS IS DATA BASE, THEN
16 EXIT TO :NAME ERR: IF NEW NAME STARTS WITH DOUBLE EXCLAMATION
17 ENDIF
18 BUILD AWA MANAGER REQUEST FOR TOC ENTRY FOR CURRENT NAME
19 BUILD AWA MANAGER REQUEST TO VERIFY NEW NAME
20 CALL AWA
21 EXIT TO :TOCERR: IF RETURN INDICATES ERROR ON FIRST REQUEST
22 EXIT TO :TOCERR: IF NO ERROR RETURNED ON SECOND REQUEST
23 CALL EXEC TO GET TOC ENTRY
24 IF CLASS IS DATA BASE, THEN
25 IF TYPE IS POB, THEN
26 CALL XDBDA TO ADD NEW POB TO LOG FILE (XDBD)
27 EXIT TO :FILE ERR: IF FMGR ERROR RETURNED
28 EXIT TO :TOCERR: IF NEW NAME IS DUPLICATE
29 EXIT TO :MAXERR: IF POB MAX IS EXCEEDED
30 SET FILE PREFIX TO RIGHT BRACKET
31 EXIT TO :INVLD: IF FILE IS NDB
32 SET FILE PREFIX TO *
33 ENDIF
34 SET FMGR FILE TYPE TO 1
35 SET SECURITY CODE TO 88
36 ELSE
37 SET FILE PREFIX TO /
38 SET SECURITY CODE = 0
39 SET FMGR FILE TYPE FROM TOC ENTRY
40 IF FILE IS TYPE 2, THEN
41 STORE RECORD LENGTH FROM TOC ENTRY
42 ENDIF
43 CALL AWA
44 CALL XRFM TO FORMAT NEW FILE NAME
45 CALL CREATE FOR NEW FILE
46 EXIT TO :FILE ERR: IF FMGR ERROR RETURNED
47 CALL XRFM TO FORMAT OLD FILE NAME
48 CALL OPEN FOR OLD FILE
49 EXIT TO :FILE ERR: IF FMGR ERROR RETURNED
50 EXIT TO :TYPE ERR: IF TYPE IS NOT SAME AS IN TOC
51 DO UNTIL END-OF-FILE IS READ ON OLD FILE
52 CALL READ TO READ RECORD FROM OLD FILE
53 EXIT IF ERROR DETECTED TO :FILE ERR:
54 CALL WRITE TO WRITE RECORD TO NEW FILE
55 EXIT IF ERROR DETECTED TO :FILE ERR:
56 END DO
57 CALL CLOSE FOR OLD FILE
58 CALL CLOSE FOR NEW FILE
BUILD AND ISSUE AHA MANAGER REQUEST TO ALLOCATE TOC ENTRY
EXIT TO :TOCERR: IF ERROR IS INDICATED
ELSE
BUILD MANAGER REQUEST FOR CURRENT TOC ENTRY
BUILD MANAGER REQUEST FOR DATA RETRIEVAL
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
CALL EXEC TO GET TOC ENTRY
BUILD MANAGER REQUEST TO ALLOCATE NEW TABLE
BUILD MANAGER REQUEST TO STORE TABLE
IF CLASS IS INTERFACE TABLE, THEN
CALL EXEC TO WRITE NEW TABLE NAME TO SAM
BUILD MANAGER REQUEST TO STORE NEW NAME IN TABLE
ENDIF
CALL XREQ
EXIT TO :TOCERR: IF ERROR IS INDICATED
ENDIF
EXIT
376 2 :SYNTAX: CALL XRMSG -"SYNTAX ERROR ..." AND EXIT
377 2 :CLASER: CALL XRMSG -"INVALID CLASS DESIGNATOR ..." AND EXIT
378 2 :NAMERR: CALL XRMSG -"NEW NAME IS INVALID ..." AND EXIT
379 2 :MAXERR: CALL XRMSG -"AUTHORIZED LIMIT ..." AND EXIT
380 2 :INVALD: CALL XRMSG -"NOB CANNOT BE ..." AND EXIT
381 2 :FILERR: CALL XRMSG -"FILE ACCESS ERROR ... ON ......." AND EXIT TO :END:
382 2 :TOCERR: CALL XRMSG TO OUTPUT APPROPRIATE MESSAGE AND EXIT TO :END:
383 2 :TYPEERR: CALL XRMSG -"INCONSISTENT FILE TYPE ..."
END:
384 2 IF PDB HAS BEEN LOGGED IN XPDG, THEN
385 3 CALL XDROD TO DELETE PDB FROM XPDG
386 2 ENDF
387 2 IF A NEW FILE HAS BEEN BUILT, THEN
388 3 PURGE NEW FILE
389 3 CLOSE OLD FILE
390 2 ENDF
391 1 ENDF XDOP
**FORTRAN CALLING PROCEDURES**

1 **CALL XDBA(NAME, IERR, ISIZE)**
2 **CALL XDBV(NAME, IERR)**

**INPUT**
- **NAME** - 2 WORD INTEGER ARRAY CONTAINING THE UNQUALIFIED ASCII
  - FOR XDBA ONLY
- **MBSIZE** - INTEGER WORD CONTAINING THE MBS/PDB SIZE IN BLOCKS
  - FOR XDBA ONLY

**OUTPUT**
- **IERR** - INTEGER WORD CONTAINING RETURN CODE
  - 0 = NO ERRORS
  - 1 = DUPLICATE NAME OR NAME NOT FOUND, AS APPROPRIATE
  - 2 = PDB/PDB LIMIT EXCEEDED
  - 3 = INTEGER WORD CONTAINING THE MBS/PDB SIZE IN BLOCKS
  - FOR XDBA ONLY

**COMMON USE**
- **XREL**
- **XS** - DCO (WORDS 1-144)

**NOTES**
- XRUL, XRULC, .ENTR, OPEN, CLOSE, READ, AND WRITF ARE USED
- WHEN WORKING WITH MBS'S, QUAI SHOULD BE SET TO 77 OCTAL
445 1 BEGIN XD08A
446 2 STORE RETURN ADDRESS
447 2 CALL :ENTRY TO SET UP CALLING ARGUMENTS
448 2 CALL XLCK FOR EXCLUSIVE USE OF XPDB
449 2 CALL OPEN FOR EXCLUSIVE USE OF XPDB
450 1 EXIT TO :FILERR: IF ERROR RETURNED
451 2 COMPUTE RECORD NUMBER FOR USER'S DIRECTORY (QUAL-77B)/2+1
452 2 CALL READF FOR RECORD COMPUTED
453 1 EXIT TO :FILERR: IF ERROR RETURNED
454 2 DETERMINE PART OF RECORD TO BE USED
455 2 IF REQUEST IS FOR ADD, THEN
456 3 EXIT TO :MAXERR: IF CURRENT # OF ENTRIES + 1 > MAX ENTRIES
457 3 START SEARCH UNTIL ALL CURRENT ENTRIES ARE TESTED
458 3 EXIT TO :NAMERR: IF ENTRY NAME MATCHES PARAMETER NAME
459 3 ENDOOP
460 3 ENDSEARCH
461 3 STORE NEW NAME AND SIZE IN ENTRY FOLLOWING LAST ENTRY
462 3 INCREMENT # OF CURRENT ENTRIES
463 2 ELSE
464 3 START SEARCH UNTIL ALL CURRENT ENTRIES, IF ANY, ARE TESTED
465 3 EXIT IF ENTRY NAME MATCHES PARAMETER NAME
466 3 ENDOOP
467 3 EXIT TO :NAMERR:
468 3 ENDSERCH
469 3 IF REQUEST IS FOR VERIFY, THEN
470 4 STORE WORD 3 OF ENTRY IN LSIZE
471 3 ELSE
472 4 REPLACE ENTRY WITH LAST ENTRY
473 4 STORE ZEROS IN LAST ENTRY
474 4 DECREMENT # OF CURRENT ENTRIES
475 3 ENDF
476 2 ENDIF
477 2 CALL WRITF TO WRITE RECORD TO XPDB
478 1 EXIT TO :RETURN:
479 2 :NAMERR: SET IERR = 1 AND EXIT TO :RETURN:
480 2 :MAXERR: SET IERR = 2 AND EXIT TO :RETURN:
481 2 :FILERR: SET IERR = FNGA ERROR CODE
482 2 :RETURN: CALL CLOSE FOR XPDB
483 2 CALL XLCK TO RETURN RESOURCE #
484 1 END XD08A
**FORTRAN CALLING PROCEDURE**

CALL XDELE

**XDELE PROCESSES THE DELETE DIRECTIVE. EACH ELEMENT**

**SPECIFIED ON THE DIRECTIVE IS DELETED FROM THE WAR.**

**IF THE ELEMENT IS A DATA BASE, THE ASSOCIATED FILE**

**MANAGER FILE IS PURGED AND FOR A PDB THE PDB DIRECTORY**

**IS UPDATED**

**INPUT**

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

**OUTPUT**

COMMON XE - REGBUF

**NOTES**

Routines Used

EXEC

IAND

PURGE

XDDDB

XREG

XREX

XRIG

XRMOV

XRMSG

XRPKX

XRPM

XRUPK

XRUDG

XUDES

XUDEL
1 BEGIN XDELE
2 DO WHILE END-OF-STATEMENT NOT REACHED PROCESSING EACH ELEMENT SPECIFIED
3 ERREXIT IF COMMA IS NOT NEXT LEXICAL ELEMENT :ERROR:
4 IF CLASS DESIGNATOR IS SPECIFIED, THEN
5 SET REQUESTED CLASS APPROPRIATELY (B, S, I, D, OR F)
6 ELSE
7 SET REQUESTED CLASS TO BE (D)
8 ENDIF
9 IF DATA BASE TO BE DELETED, THEN
10 BUILD AND ISSUE AWAMANAGER 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 XPROC TO DELETE THIS PDB FROM XPDG
15 IF ERROR IS RETURNED, THEN
16 CALL XRMSG - "FILE ACCESS ERROR 0... XPDG"
17 SET ERROR FLAG
18 ENDIF
19 ENDIF
20 BUILD AND ISSUE AWAMANAGER 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 DEDE (F), THEN
27 IF ERROR FLAG IS NOT SET, THEN
28 CALL XRMSG TO CONSTRUCT FILE NAME
29 IF RETURN CODE FROM PURGE, THEN
30 CALL XRMSG - "FILER ERROR XXXX XXXX"
31 ENDIF
32 ENDIF
33 ELSE
34 CALL XRMSG - "... IS A MDO. NOT DELETED."
35 ENDIF
36 END
37 1 EXIT TO :RETURN:
38 2 :ERROR: CALL XRMSG - "SYNTAX ERROR"
39 2 :RETURN:
40 1 END XDELE
CALL XMIS (TOCLST)

XMIS PROCESSES A LIST OF ELEMENTS TO BE STORED/RESTORED BY
DECODING 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) ABLGL, XB(201) NOTOC

OUTPUTS IN CALLING SEQUENCE:

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

OUTPUTS IN COMMON:

XB(151) ABLGL, XB(157) TOTSIZ, XB(158) TOTWD

INTERNAL XB COMMON USED:

XB(151) ABLGL - (INTEGER, 1 WORD) ABORT FLAG
XB(152) ERLGL - (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
BE STORED/RESTORED
XB(158) TOTWD - (INTEGER, 1 WORD) TOTAL # WORDS OF DATA TO
BE STORED/RESTORED
XB(198) RETC - (INTEGER, 1 WORD) XTCOM RETURN CODE
XB(199) BATCLS - (INTEGER, 1 WORD) CLASS OF DATA BEING SEARCHED
XB(200) HMPTR - (INTEGER, 1 WORD) POINTER TO NAME IN COMMON
XB(201) NOTOC - (INTEGER, 1 WORD) NUMBER ENTRIES IN TOCLST

COMMON USED:

EQUIVALENCE
+ (FB(5), MASSTA), (FB(8), EOS )
1 BEGIN XLIS
2 DO WHILE ERROR FLAG IS ON OR UNTIL RESPONSE IS CR
3 TURN ERFLAG OFF

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

:RTM2:
3 IF ERROR FLAG IS ON THEN
4 CALL XCOM TO REPROMPT USER TO CONTINUE
5 ERREXIT IF RESPONSE IS X TO :ERR3:
6 ENDIF
7 ENDDO
8 1 EXIT XLIS

:ERR1:
2 SET ERROR FLAG ON
2 CALL XMESG TO DISPLAY SYNTAX ERROR
2 GO TO :RTM2:

:ERR2:
2 IF ERROR FLAG IS OFF THEN
2 TURN ERROR FLAG ON
2 CALL XMESG TO DISPLAY NOT STORED/RESTORED MESSAGE
2 ENDF
2 CALL EXEC TO DISPLAY ELEMENT NAME
2 GO TO :RTM1:

:ERR3:
2 SET ABLFLG TO ABORT STORE/RESTORE OPERATION

1 END XLIS
FORTRAN CALLING PROCEDURE

CALL XILST

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

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

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

INTERNAL VARIABLES

COMMON XS - BUFFER = PRINT LINE TO BE OUTPUT
BUFFTR = INDEX INTO BUFFER FOR NEXT ASCII DATA

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

USES ROUTINES - EXEC, XEINT, XILSD, XILST,
XREX, XREX, XRMW, XRMSE,
XRNSX, XRNSX, XSLST, OPEN,
CLOSE, READY

NOTES
783 1 CD**********
784 1 CDD0
785 1 CDD0 FORTRAN CALLING PROCEDURE
786 1 CDD0
787 1 CDD0 CALL XDFF
788 1 CDD0
789 1 C**********
790 1 CD1 XDFF CONFIRMS THE USER'S REQUEST FOR TERMINATION,
791 1 CD1 DELETES ALL ORDE AND UTOD FILES LOGGED IN THE ANA,
792 1 CD1 PERFORMS ABNORMAL TERMINATION, IF INDICATED, OR
793 1 CD1 RETURNS NORMAL PARMS TO THE FDS MANAGER AND TERMINATES
794 1 CD1 NORMALLY VIA RTE.
795 1 CD1
796 1 CD1
797 1 C**********
798 1 CD2 INPUT
799 1 CD2
800 1 CD2 COMMON XE - LU, FLAGS, QUAL, RERBUF
801 1 CD2
802 1 CD2 COMMON XB - ORG = ORIGIN ADDRESS OF ANA, USED TO CALCULATE
803 1 CD2 INDICES INTO 'ANA' FROM ADDRESS POINTERS
804 1 CD2
805 1 CD2 OF TOC ENTRIES
806 1 CD2 ANA = IMAGE OF ANA HEADER, CHAIN HEADS, AND
807 1 CD2 TOC RETRIEVED VIA XREQ
808 1 CD2
809 1 C**********
810 1 CD4 INTERNAL VARIABLES
811 1 CD4
812 1 CD4 COMMON XS - POINTER = CHAIN POINTER TO NEXT (OR 1ST) TOC
813 1 CD4 ENTRY. MOST SIGNIFICANT (BIT 13) BIT
814 1 CD4 SET TO INDICATE END-OF-CHAIN
815 1 CD4
816 1 CD4 INDEX = VALUE COMPUTED FROM POINTER TO BE
817 1 CD4 FORTRAN INDEX INTO 'ANA' FOR NEXT TOC
818 1 CD4 ENTRY
819 1 C**********
820 1 CD5
821 1 CD5 NOTES
822 1 CD5
823 1 CD5 ROUTINE USED - EXEC, PURGE, XDSTA, XPAR, XREQ, XREXT, XRI6,
824 1 CD5 XRNISG, XRSFM, XRPCK, XRSRT, XRPFK, XTCOM,
825 1 CD5 XUDSG, XVABN
1 BEGIN XDOFF
2 PROMPT USER FOR TERMINATION CONFIRMATION
3 IF USER RESPONDS GO AHEAD WITH TERMINATION THEN
4 CALL TKN TO REQUEST TOC AND CHAIN HEADS
5 IF CHAIN HEAD FOR DRE FILE IS NOT NEGATIVE THEN
6 DO UNTIL DRE CHAIN HEAD IS NEGATIVE
7 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
8 OUTPUT 'X'TOC TOO LARGE, PURGE INCOMPLETE'
9 EXIT DO
10 ENDIF
11 CALL XFRNF TO CREATE FILE NAME '/XXXXX'
12 CALL PURGE TO SCRATCH FILE
13 SET DRE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
14 ENDDO
15 ENDIF
16 IF CHAIN HEAD FOR DATA BASES IS NOT NEGATIVE, THEN
17 DO UNTIL DATA BASE CHAIN HEAD IS NEGATIVE
18 IF CHAIN POINTS BEYOND END OF TOC BUFFER THEN
19 OUTPUT 'X'TOC TOO LARGE, PURGE INCOMPLETE'
20 EXIT DO
21 ENDIF
22 IF TYPE OF DATA BASE IS UTOB, THEN
23 CALL XFRNF TO CREATE FILE NAME '/XXXXX'
24 CALL PURGE TO SCRATCH FILE
25 ENDIF
26 SET DATA BASE CHAIN HEAD TO TOC ENTRY CHAIN POINTER
27 ENDDO
28 ENDDO
29 CALL XSTTA TO OUTPUT USAGE STATISTICS
30 IF USER REQUESTED DEBUG SNAP THEN
31 CALL XDBGC
32 ENDIF
33 IF USER REQUESTED ABEND DUMP THEN
34 CALL XVAHC - NO RETURN FROM THIS CALL
35 ENDIF
36 SET PARAMETER 1 TO INDICATE TERMINATE EXEC
37 CALL XPRT TO WAIT ON I/O COMPLETION, RETURN PARAMS AND TERMINATE EXEC
38 RETURN
39 ENDDO
40 XDOFF
### 871 1 CD

**FORTRAN CALLING SEQUENCE:**

1. CALL XDRDE (DATBUF, DBDCB)

---

### 878 1 CD

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

---

### 886 1 CD

**INPUTS FROM CALLING SEQUENCE:**

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

---

### 890 1 CD

**INPUTS FROM XA COMMON:**

1. XA(154) ALLFLG, XA(175) EMDBK, XA(182) TOTSIZ,
2. XA(201) NOTOC

---

### 898 1 CD

**OUTPUTS IN CALLING SEQUENCE:**

1. DATBUF, DBDCB

---

### 900 1 CD

**OUTPUTS IN XA COMMON**

1. XA(151) ABFLG, XA(152) ERLFLG, XA(165) TOCHDS,
2. XA(169) TOPBLK, XA(170) EMDBK, XA(171) TOCDX,
3. XA(201) NOTOC

---

### 908 1 CD

**INTERNAL XA COMMON USED:**

1. **XA(151) ABFLG** - ABORT FLAG
2. **XA(152) ERLFLG** - ERROR MESSAGE FLAG
3. **XA(154) ALFLG** - RESTORE ALL UTDB FLAG
4. **XA(155) DBFLG** - DB FILE NAME
5. **XA(156) DATBLK** - BLOCK # WHERE DATA ITEM BEGINS
6. **XA(157) DATHDS** - WORD INDEX INTO DATBUF WHERE DATA BEGINS
7. **XA(164) EFLG** - ERROR FLAG FOR FMGR CALLS
8. **XA(165) TOCHDS** - # WORDS OF TOC IN TOCBUF (NOTOC *8) + 8
9. **XA(166) NOBLKS** - # BLOCKS TO READ
10. **XA(167) LEN** - # WORDS OF DATA TO MOVE
11. **XA(168) DATEND** - BLOCK # WHERE DATA ITEM ENDS
12. **XA(169) TOPBLK** - BLOCK # OF FIRST BLOCK IN DATBUF
13. **XA(170) EMBLK** - BLOCK # OF LAST BLOCK READ
14. **XA(171) TOCDX** - INDEX INTO TOC OF CURRENT DATA ITEM
15. **XA(175) IL** - # WORDS OF DATA TO READ
16. **XA(176) TDEN** - DATA BASE TOC ENTRY
17. **XA(201) NOTOC** - # TOC ENTRIES IN TOCBUF
18. **XA(201) TOCBUF** - TOC BUFFER
1 BEGIN XRDE
2 INITIALIZE FILE INDICES TO INDICATE NO DATA IN DATBUF
3 INITIALIZE REQUEST BUFFER TO SAY NO REQUESTS
4 DO WHILE THERE ARE NON-DONE FILES TO PROCESS
5 IF ALLFLG IS ZERO OR IF STORE/RESTORE BIT IS ON THEN
6 IF DATA IS NOT CURRENTLY IN DATBUF THEN
7 CALL READ TO READ 1 BUFFER BEGINNING WITH DATABLK FOR THIS ELEMENT
8 ERREXIT IF READ ERROR TO :ERR1:
9 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
10 ELSE, DATA BEGINS IN DATBUF
11 IF DATA DOES NOT END IN DATBUF THEN
12 CALL XMOV TO MOVE PARTIAL DATA TO TOP OF DATBUF
13 COMPUTE SIZE AND LOCATION OF DATA TO BE READ
14 CALL READ TO READ ENOUGH TO FILL DATBUF
15 ERREXIT IF READ ERROR TO :ERR1:
16 SET FILE INDICES INDICATING WHICH DATA IS IN DATBUF
17 ENDIF
18 ENDIF
19 BUILD AW REQUEST TO ALLOCATE AND STORE DATA
20 CALL EXEC TO WRITE DATA TO SAN
21 ERREXIT IF ERROR FROM EXEC TO :ERR1:
22 IF AW REQUEST BUFFER IS FULL THEN
23 CALL XRDE TO ISSUE REQUEST
24 EXIT XRDE IF ERROR IN XRDE
25 ENDIF
26 ENDIF
27 1 ENDXRDE
28 :ERR1: CALL XR16 TO CONVERT ERROR CODE TO ASCII
29 CALL XRMSG TO DISPLAY ERROR MESSAGE (208)
30 GO TO :ERR3:
31 :ERR2: CALL XRMSG TO DISPLAY ERROR MESSAGE (212)
32 :ERR3: SET ABFLG TO SAY ABORT RESTORE
33 1 END XRDE
FORTRAN CALLING SEQUENCE:

CALL XDRDF (DATBUF, DBDCB)

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

INPUTS IN CALLING SEQUENCE:

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

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

INPUTS IN XD COMMON:

XB(152) ERFLG, XB(154) ALLFLG, XB(165) TOCBO, XB(166) ENUMLG, XB(170) ENDBL, XB(171) TOCHB,
XB(182) TOTBUZ

OUTPUTS IN CALLING SEQUENCE:

DBDCB

OUTPUTS IN XD COMMON:

XB(151) ABFLG

INTERNAL XD COMMON USED:

XB(151) ABFLG - ABORT FLAG
XB(152) ERFLG - ERROR MESSAGE FLAG
XB(154) ALLFLG - RESTORE ALL UTDB FLAG
XB(155) DBG - DEBUG FLAG
XB(159) FILNAM- DD FILE NAME
XB(162) DATBKLK- BLOCK # WHERE ORDE BEGINS
XB(163) DATCHBX - WORD INDEX INTO DATBUF WHERE ORDE BEGINS
XB(164) ERRR - ERROR FLAG FOR FNGR CALLS
XB(165) TOCHSK- # WORDS OF TOC IN TOCBUF (HOTOC #8) + 8
XB(166) NOBLS- # BLOCKS OF UTDB/ORDE TO READ/WRITE
XB(167) LEN- # WORDS OF DATA TO MOVE
XB(169) TOLBLK- BLOCK # OF FIRST BLOCK IN DATBUF
XB(170) ENDBL- BLOCK # OF LAST BLOCK READ
XB(171) TOCHBX- INDEX TO CURRENT TOC ENTRY
XB(172) DRBFIL- ORDE FILE NAME
XB(175) IL- # WORDS OF DATA TO READ/WRITE
XB(201) TOCBUF- TOC BUFFER

COMMON USED:
1061 1 BEGIN XDRDF
1062 2 DO WHILE THERE ARE TOC ENTRIES TO PROCESS
1063 3 IF ALLFLG IS ZERO OR THE STORE/RESTORE BIT IS ON THEN
1064 4 CALL XDRFN TO CREATE THE DREX FILE NAME
1065 4 CALL CREAT TO CREATE THE DREX FILE
1066 4 IF THERE WAS A CREATE ERROR THEN
1067 5 IF ERROR FLAG IS OFF THEN
1068 6 SET ERROR FLAG ON
1069 6 CALL XMSG TO DISPLAY MAIN MESSAGE
1070 5 ENDIF
1071 5 CALL XDRG TO CONVERT ERROR CODE TO ASCII
1072 5 CALL XRCR TO DISPLAY DREX NAME AND ERROR CODE
1073 4 ELSE
1074 5 CALL CLOSE TO CLOSE FILE
1075 5 ERREXIT IF CLOSE ERROR TO :ERR1;
1076 5 CALL OPEN TO OPEN DREX FILE AS TYPE 1
1077 5 ERREXIT IF OPEN ERROR TO :ERR1;
1078 5 DO UNTIL ALL BLOCKS OF DREX HAVE BEEN PROCESSED
1079 6 IF DATA FOR DREX IS IN DATABUF THEN
1080 7 CALL WRIF DATA TO DREX
1081 7 ERREXIT IF WRIF ERROR TO :ERR1;
1082 6 ELSE
1083 7 CALL READF TO READ NEXT BUFFER OF DATA
1084 7 ERREXIT IF READF ERROR TO :ERR2;
1085 7 SET INDICES INDICATING DATA IN BUFFER
1086 6 ENDIF
1087 5 ENDDO
1088 5 CALL CLOSE TO CLOSE DREX FILE
1089 5 ERREXIT IF CLOSE FAILED TO :ERR1;
1090 5 BUILD AND REQUEST TO ALLOCATE DREX IN AMA
1091 5 IF AMA REQUEST BUFFER IS FULL THEN
1092 6 CALL XDRER TO MAKE REQUEST
1093 6 EXIT XDRDF IF XERRR ERROR
1094 5 ENDIF
1095 4 ENDIF
1096 3 ENDDO
1097 2 ENDDO
1098 1 EXIT XDRDF
1099 2 :ERR1:
1100 2 CALL PURGE TO PURGE DREX FILE
1101 2 :ERR2:
1102 2 CALL XMSG TO DISPLAY ERROR MESSAGE WITH FILE NAME
1103 2 SET ABFLG TO SAY ABORT RESTORE
1104 1 END XDRDF
**FORTAN CALLING PROCEDURE**

CALL IXREC

**DESCRIPTION**

IXREC PROCESSES THE RECALL DIRECTIVE. A UTOD IS CREATED AND
THE CONTENTS OF THE SPECIFIED PDD ARE COPIED TO IT.

**INPUT**

COMMON XE - XATRG, COMBUF, COMPTR, FLAGS, LV, TOKENS

FILES - PDD FILE SPECIFIED

**OUTPUT**

COMMON XE - REGBUF, REGPTR

FILES - UTOD FILE SPECIFIED

**INTERNAL VARIABLES**

DCBPD0 - DCB FOR THE PDD FILE; ALLOCATED IN XB COMMON;

CONTAINS 1532 WORD BUFFER USED TO READ THE PDD

AND TO WRITE THE UTOD

DCBUTD - DCB FOR THE UTOD FILE; ALLOCATED IS XS COMMON
BEGIN XDEC

1141
1142     BEGIN XDEC
1143     SET STATUS FLAG INDICATING PRO & UTD0 FILES NOT OPEN
1144     ERROR IF "-" IS NOT NEXT TOKEN :ERROR9:
1145     INCREMENT TO NEXT TOKEN
1146     ERROR IF NEXT TOKEN IS NOT "NAME" :ERROR:
1147     ERROR IF THIS NAME IS > A CHARACTERS :ERROR8:
1148     IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR16:
1149     RETAIN THIS NAME AS PRO
1150     RETAIN CURRENT USER IN (QUALIFIER)
1151     INCREMENT TO NEXT TOKEN
1152     IF NEXT TOKEN IS "-", THEN
1153     INCREMENT TO NEXT TOKEN
1154     ERROR IF NEXT TOKEN IS NOT "NAME" :ERROR:
1155     ERROR IF NEXT TOKEN IS NOT "NAME" :ERROR:
1156     ERROR IF THIS NAME IS > A CHARACTERS :ERROR8:
1157     IF THIS NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR16:
1158     RETAIN THIS NAME AS UTD0
1159     INCREMENT TO NEXT TOKEN
1160     ERROR IF NEXT TOKEN IS NOT END :ERROR4:
1161     BUILD UTD0 REQUEST TO VERIFY EXISTENCE OF UTD0
1162     CALL XDR0 TO PROCESS XDR0 REQUEST
1163     ERROR IF UTD0 DOES ALREADY EXIST :ERROR22:
1164     CALL XDR0 TO VERIFY PRO AND RETRIEVE SIZE
1165     ERROR IF PRO DOES NOT EXIST :ERROR16:
1166     ERROR IF VMER ERROR RETURNED :ERROR4:
1167     CALL XRTH0 TO CULL PRO FILE NAME
1168     CALL OPEN TO OPEN PRO FILE
1169     ERROR IF OPEN FAILED :ERROR18:
1170     SET STATUS FLAG INDICATING PRO FILE OPEN
1171     BUILD UTD0 REQUEST TO OPEN PRO
1172     CALL XDR0 TO PROCESS XDR0 REQUEST
1173     ERROR IF XDR0 REQUEST FAILS :ERROR21:
1174     SET STATUS FLAG TO INDICATE UTD0 FILE ALLOCATED IN ADA
1175     RESTORE CURRENT USER'S ID (QUALIFIER)
1176     CALL XRTH0 TO BUILD UTD0 FILE NAME
1177     CALL CREATE TO CREATE TYPE 1 UTD0 FILE
1178     ERROR IF CREATE FAILED :ERROR:
1179     SET STATUS FLAG INDICATING UTD0 FILE NOW OPEN
1180     DO FOR EACH BUFFER OF DATA IN PRO FILE
1181     CALL READ TO READ 1 BUFFER FROM PRO FILE
1182     ERROR IF READ ERROR :ERROR18:
1183     CALL WRTH0 TO WRITE 1 BUFFER TO UTD0 FILE
1184     ERROR IF WRITE ERROR :ERROR9:
1185     ENDDO
1186     CALL CLOSE FOR PRO
1187     CALL CLOSE FOR UTD0
1188     EXIT XDEC

:ERROR4: ISSUE MESSAGE - "SYNTAX ERROR. EXTRANEOUS DATA"

:ERROR9: ISSUE MESSAGE - "ILLEGAL: UTD0 NAME (NOT FOUND OR TOO LONG)"
1200 2 :ERROR7: ISSUE MESSAGE - "UTDB FILE ACCESS ERROR ..."
1201 2 :ERROR9: ISSUE MESSAGE - "SYNTAX ERROR - ILLEGAL OR MISSING FIELD"
1202 2 :ERROR6: ISSUE MESSAGE - "INVALID PDB FILE NAME..."
1203 2 :ERROR8: ISSUE MESSAGE - "PDB FILE ACCESS ERROR ..."
1204 2 :ERROR9: ISSUE MESSAGE - "USER ID IS INVALID FOR PDB/UTDB LOGGING"
1205 2 :ERROR2: ISSUE MESSAGE - "AVM OVERFLOW - XXXX NOT LOGGED"
1206 2 :ERROR2: ISSUE MESSAGE - "XXXX ALREADY EXISTS"
1207 2 :ERROR4: ISSUE MESSAGE - "FILE ACCESS ERROR #... XDB"
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 AVM, THEN
1213 3 CALL XOR TO DELETE UTDB FROM AVM
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 "NO XDBREC"
**FORTRAN CALLING PROCEDURE**

**CALL XDREN**

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

**COMMON XE - CARTRG, COMBUF, COMPTR, QUAL, REPTR, TOKENS**

**INPUT**

**OUTPUT**

**COMMON XE - RERBUF**

**INTERNAL VARIABLES**

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

**INDEX**

**ERROR MESSAGE NUMBERS**

**NEW FILE NAME**

**NEW NAME IN COMBUF**

**OLD FILE NAME**

**OLD NAME IN COMBUF**

**TOC ENTRY FOR DATA BASE RENAME**

**DATA CLASS CODE (STORED IN LEFT BYTE)**

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

**EXTERNAL REFERENCES**

**Routines Used -**

**EXEC**

**JOIN**

**KVVT**

**MAIN**

**XDBM**

**XDBD**

**XREG**

**XRMOV**

**XRMSG**

**XRGN**

**XRFS**

**XRFSR**
1333 2 :NAMAGM:
1334 2 CALL FILE MANAGER TO CHANGE NAME BACK
1335 2 :UNDO:
1336 2 BUILD AND ISSUE AN AMA MANAGER REQUEST TO CHANGE NAME BACK
1337 2 CALL XRMFG ("FILE MANAGER ERROR # ; RENAME UNSUCCESSFUL") AND EXIT
1338 2 :TELUS:
1339 2 CALL XRMFG ("PDB NOT LOGGED IN XPDD; SYSTEM ERROR # ...") AND EXIT
1340 1 END XDREN
BEGIN XREG
CALL XREG TO PROCESS AWV REQUEST(S)
IF AN ERROR RETURNED BY AWV MANAGER, THEN
  IF ERROR FLAG (ERFLAG) IS ZERO, THEN
    TURN ON ERFLAG INDICATING THAT MSG 234 HAS BEEN ISSUED
    CALL XRMSG TO OUTPUT MSG 234 - 'FOLLOWING ELEMENTS NOT RESTORED'
  ENDIF
  CALL EXEC TO WRITE ELEMENT NAME, CLASS AND REASON
  IF CLASS OF ELEMENT IS DDE, THEN
    CALL PURGE TO DELETE THE FILE
  ELSE, ELEMENT RESIDES IN AWV
  CALL EXEC TO FREE CLASS NO. AND SAM BUFFER
  ENDIF
  IF AWV REQUESTS EXIST IN REGQBF BEYOND FAILING REQUEST, THEN
    MOVE THESE REQUESTS TO TOP OF REQUEST BUFFER
  ENDIF
ELSE
  SET REGPTR TO 1 INDICATING NO REQUESTS PRESENT
ENDIF
EXIT XREG
END XREG
*FORTRAN CALLING PROCEDURE*

**CALL XDBAF (DATBUF)**

**CALLING SEQUENCE**

**INPUT**
- COMMON XE - LU, FLAGS, REPTR, REPBUF, TOKENS,
  CARGS, COMPT, COMBUF

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

**OUTPUT**
- COMMON XE - REPBUF

**COMMON XB - ABFLG = ABORT FLAG, NON-ZERO VALUE FROM XDBAF,**
  XDBDF, XDBIS OR XDBEI INDICATES ABORT
  OF RESTORE

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

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

**ENDBLK = BLOCK # OF LAST BLOCK READ**

**ERFLG = SET NON-ZERO IF MESSAGE #54 ISSUED SO**

**THAT IT IS ISSUED ONLY ONCE**

**FILNM = UTDB/MD5 FILE NAME**

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

**WORDS**

**TOCEN = AWAY TOC ENTRY FOR THE UTDB**

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

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

**NOTES**

**ROUTINES CALLED**

**CLOSE**
**EXEC**
**OPEN**
**READF**
**XDBIS**
**XDBDF**
**XDBF**
**XDBE**
**XDBAF**
**XDBAF**
1476   1 BEGIN XDRES
1477   2 BEGIN XDRES
1478   3 SET ABLFLG TO ZERO
1479   4 ERREXIT IF NEXT TOKEN IS NOT A COMMA :ERR09:
1480   5 ERREXIT IF FOLLOWING TOKEN IS NOT A NAME :ERR09:
1481   6 BUILD AWS REQUEST FOR TOC ENTRY RETRIEVE
1482   7 CALL AWS TO PROCESS AWS REQUEST
1483   8 ERREXIT IF AWS REQUEST FAILED :ERR10:
1484   9 IF DATA BASE IS A PDB :ERR33:
1485  10 IF DATA BASE IS A UDB, THEN
1486  11 CALL XRMFS TO CONSTRUCT QUALIFIED FILE NAME
1487  12 ENDIF
1488  13 CALL OPEN TO OPEN SPECIFIED FILE
1489  14 ERREXIT IF OPEN FAILED :ERR08:
1490  15 CALL READ TO READ FIRST RECORD OF DATA BASE FILE INTO TOCBUF
1491  16 ERREXIT IF READ FAILED :ERR08:
1492  17 INITIALIZE E NDBLK TO NUMBER OF TOC BLOCKS
1493  18 IF TOC IS MORE THAN 1 BLOCK LONG, THEN
1494  19 CALL READ TO READ REMAINING TOC ENTRIES INTO TOCBUF
1495  20 ERREXIT IF READ FAILED :ERR08:
1496  21 ENDIF
1497  22 UPDATE TOTBLK TO NUMBER OF BLOCKS REMAINING IN FILE (DECREMENT BY E NDBLK)
1498  23 CLEAR ERROR MESSAGE FLAG (ERFLG)
1499  24 IF TOKEN IS EOS (i.e., NO LIST OF ELEMENTS), THEN
1500  25 SET ALLFLG TO ZERO INDICATING TO RESTORE ALL TOC ENTRIES
1501  26 ELSE
1502  27 SET ALLFLG NON-ZERO INDICATING TO RESTORE ONLY FLAGGED TOC ENTRIES
1503  28 CALL XDLIS TO PROCESS ELEMENTS SPECIFIED AND TO FLAG TOC ENTRIES
1504  29 EXIT XDRES IF ALLFLG SET BY XDLIS
1505  30 ENDIF
1506  31 CALL XDREO TO RESTORE AWS RESIDENT ELEMENTS
1507  32 EXIT XDRES IF ABLFLG SET BY XDREO
1508  33 CALL XDREO TO RESTORE ORDEO'S
1509  34 EXIT XDRES IF ABLFLG SET BY XDREO
1510  35 CALL CLOSE TO CLOSE DATA BASE FILE
1511  36 ERREXIT IF CLOSE FAILED :ERR08:
1512  37 DO WHILE AWS REQUESTS REMAIN IN ERBUF
1513  38 CALL XDREO TO PROCESS AWS REQUESTS
1514  39 EXIT XDRES IF ABLFLG SET BY XDREO
1515  40 ENDDO
1516  41 EXIT XDRES
1517  42 :ERR04: CALL XRMFS - 'SYNTAX ERROR. EXTRAMENOS DATA'
1518  43 :ERR08: CALL XRMFS - 'FILE MANAGER ERROR .... ------'
1519  44 :ERR09: CALL XRMFS - 'SYNTAX ERROR. MISSING OR ILLEGAL FIELD'
1520  45 :ERR10: CALL XRMFS - '....... NOT FOUND'
1521  46 :ERR33: CALL XRMFS - 'CAN NOT RESTORE A PDB'
1522  47 DO UNTIL ALL AWS REQUESTS IN ERBUF HAVE BEEN PROCESSED
1523  48 IF REQUEST IS TO STORE DATA, THEN
1524  49 CALL XREX TO FREE THE SPECIFIED CLASS NO. AND SAM BUFFER
1525  50 ENDIF
C**
C6000   FORTRAN CALLING PROCEDURE
C6001   CALL XDSAV
C6002   C**
C6003   CD1   XDSAV PROCESSES THE SAVE DIRECTIVE. A PDB IS CREATED AND THE
C6004   CD1   CONTENTS OF THE SPECIFIED UTDK ARE COPIED TO IT.
C6005   CD1   C**
C6006   CD2   INPUT
C6007   CD2   COMMON XE - CARTAG, COMBUF, COMPTA, FLAGS, LU, TOKENS
C6008   CD2   CD2   FILES - **XXXQ (UTDB FILE SPECIFIED)
C6009   CD2   C**
C6010   CD3   OUTPUT
C6011   CD3   COMMON XE - REGBUF, RERPTA
C6012   CD3   CD3   FILES - XXXXQ (PDB FILE SPECIFIED)
C6013   CD3   C**
C6014   CD4   INTERNAL VARIABLES
C6015   CD4   CD4   DCBPDB - DCB FOR THE PDB FILE; ALLOCATED IN XS COMMON
C6016   CD4   CD4   DCBUTD - DCB FOR THE UTDK FILE; ALLOCATED IN XS COMMON;
C6017   CD4   CD4   CONTAINS 1152 WORD BUFFER USED TO READ THE
C6018   CD4   CD4   UTDK AND TO WRITE THE PDB.
BEGIN XDSAV
1572 1 SET STATUS FLAG TO INDICATE NO FILES OPEN, NO PDB ALLOCATED
1573 2 IF "" IS NOT NEXT TOKEN :ERROR:
1574 3 INCREMENT TO NEXT TOKEN
1575 4 ERREXIT IF THIS NAME IS NOT 'NAME' :ERROR:
1576 5 ERREXIT IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR:
1577 6 ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
1578 7 RETAIN THIS NAME AS PDB
1579 8 INCREMENT TO NEXT TOKEN
1580 9 ERREXIT IF "" IS NOT NEXT TOKEN :ERROR:
1581 10 INCREMENT TO NEXT TOKEN
1582 11 ERREXIT IF TOKEN IS NOT 'NAME' :ERROR:
1583 12 ERREXIT IF THIS NAME IS MORE THAN 4 CHARACTERS :ERROR:
1584 13 ERREXIT IF NAME BEGINS WITH DOUBLE EXCLAMATION :ERROR:
1585 14 RETAIN THIS NAME AS PDB
1586 15 INCREMENT TO NEXT TOKEN
1587 16 ERREXIT IF TOKEN IS NOT EOS :ERROR:
1588 17 CALL AWA REQUEST TO RETRIEVE UDB'S TOC ENTRY
1589 18 CALL XHM TO PROCESS AWA REQUESTS
1590 19 ERREXIT IF TOC RETRIEVE FAILED :ERROR:
1591 20 ERREXIT IF TOC ENTRY DOES NOT INDICATE UDB :ERROR:
1592 21 CALL XDQA TO ADD PDB TO XDB
1593 22 ERREXIT IF PRER ERROR :ERROR:
1594 23 ERREXIT IF PDB LIMIT EXCEEDED :ERROR:
1595 24 ERREXIT IF OPEN FAILED :ERROR:
1596 25 SET STATUS FLAG INDICATING PDB LOGGED
1597 26 CALL XNRFN TO BUILD UDB FILE NAME
1598 27 CALL OPEN TO OPEN UDB FILE (SPECIFYING TYPE 1)
1599 28 ERREXIT IF OPEN FAILED :ERROR:
1600 29 SET STATUS FLAG INDICATING UDB FILE OPEN
1601 30 CALL AWA REQUEST TO ALLOCATE PDB -- USE SIZE OF UT:
1602 31 CALL XHEN TO PROCESS AWA REQUEST
1603 32 ERREXIT IF PDB IS DUPLICATE :ERROR:
1604 33 ERREXIT IF AWA OVERFLOW :ERROR:
1605 34 SET STATUS FLAG INDICATING PDB LOGGED IN AWA
1606 35 CALL XNRFN TO BUILD PDB FILE NAME
1607 36 ERREXIT IF PDB NAMER FAILURE :ERROR:
1608 37 CALL CREATE TO CREATE PDB FILE (SPECIFYING TYPE 1) USING SIZE
1609 38 OF UDB FILE IN LKM TOC ENTRY
1610 39 ERREXIT IF CREATE FAILED :ERROR:
1611 40 SET STATUS FLAG INDICATING PDB FILE OPEN
1612 41 DO FOR ALL DATA IN UDB FILE
1613 42 ERREXIT IF READ FAILED :ERROR:
1614 43 CALL WRITE TO WRITE 1 BUFFER TO PDB FILE
1615 44 ERREXIT IF WRITE FAILED :ERROR:
1616 45 ENDB
1617 46 CLOSE UDB FILE
1618 47 CLOSE PDB FILE
1619 48 EXIT XDSAV
1620 49 :ERROR: ISSUE MESSAGE "SYNTAX ERROR = ILLEGAL OR MISSING FIELD"
1621 50 :ERROR6: ISSUE MESSAGE "INVALID UDB FILE NAME ...."
1622 51 :ERROR16: ISSUE MESSAGE "INVALID PDB FILE NAME ...."
1623 52 :ERROR04: ISSUE MESSAGE "SYNTAX ERROR, EXTRANEOUS DATA"
1624 53 :ERROR17: ISSUE MESSAGE "PDB .... ALREADY EXISTS"
1625 2 :ERR10: ISSUE MESSAGE 'PDB FILE ACCESS ERROR ...'
1626 2 :ERR00: ISSUE MESSAGE 'FILE MANAGER ERROR ...
1627 2 :ERR20: ISSUE MESSAGE 'AUTHORIZED LIMIT OF ... PDB'S
1628 2 ALREADY REACHED'
1629 2 :ERR21: ISSUE MESSAGE 'AMA OVERFLOW. ... NOT LOGGED'
1630 2 :RETURN:
1631 2 IF STATUS FLAG INDICATES PDB FILE IS OPEN, THEN
1632 2 PURGE PDB FILE
1633 2 ENDF
1634 2 IF FLAG INDICATES PDB IS IN AMA, THEN
1635 2 CALL XRIB TO DELETE PDB FROM AMA
1636 2 ENDF
1637 2 IF FLAG INDICATES UTOB IS OPEN, THEN
1638 2 CLOSE UTOB
1639 2 ENDF
1640 2 IF FLAG INDICATES PDB IS IN XPRD, THEN
1641 2 CALL XRDP TO DELETE PDB FROM XPRD
1642 2 ENDF
1643 1 END XD5AV
164:  1 CD0   CALL XDSTA (LU)
164:  1 CD1   OUTPUT ANA AND DMA USAGE DATA AND STATISTICS ON ANA COMPACTION
164:  1 CD2   PHASES
164:  1 CD3   INPUT
164:  1 CD4   - LOGICAL UNIT OF OUTPUT DEVICE
164:  1 CD5   XB COMMON
164:  1 CD6   ASZ   - TOTAL SIZE OF ANA
164:  1 CD7   FREE   - AMOUNT OF FREE SPACE IN ANA
164:  1 CD8   MSZ   - TOTAL SIZE OF DMA (IN 64 WORD SECTORS)
164:  1 CD9   DFR   - AMOUNT OF FREE SECTORS IN DMA
164:  1 CD10  NP    - ARRAY OF THREE COMPACT PHASE COUNTS
164:  1 CD11  OUTPUT
164:  1 CD12  USAGE AND STATISTICS TO LU
164:  1 CD13  NOTES
164:  1 CD14  USES EXEC, XRIG
164:  1 CD15  **********
164:  1 CD16  * *
164:  1 CD17  * *
164:  1 CD18  * *
164:  1 CD19  **********
164:  2 BEGIN XDSTA
164:  2 FORMAT AND PRINT ANA USAGE DATA
164:  2 FORMAT AND PRINT DMA USAGE DATA
164:  2 FORMAT AND PRINT COMPACT PHASE STATISTICS
164:  1 END XDSTA
1683 1 C0***********
1684 1 C00                CALL XDS0 (DATBUF)
1685 1 C00***********
1686 1 C00                XDS0 IS THE STORE DIRECTIVE HANDLER. IT VERIFIES INPUTS ON
1687 1 C00 DIRECTIVE, BUILDS UTDB TOC, CREATES UTDB AND STORES UTDB TOC
1688 1 C00 ENTRY IN AWA.
1689 1 C00***********
1690 1 C02                INPUTS FROM CALLING SEQUENCE:
1691 1 C02                DATBUF (INTEGER 1480 WORDS) - BUFFER USED TO READ IN AWA TOC.
1692 1 C02***********
1693 1 C04                INTERNAL XB COMMON USED:
1694 1 C040                X0(151) ADF0 - (INTEGER, 1 WORD) ADF0 FLAG
1695 1 C040                X0(152) ERFM - (INTEGER, 1 WORD) ERROR MESSAGE FLAG
1696 1 C040                X0(153) MSGN - (INTEGER, 1 WORD) MESSAGE NUMBER TO BE DISPLAYED
1697 1 C040                X0(157) TOTSD - (INTEGER, 1 WORD) TOTAL SIZE OF UTDB FILE
1698 1 C040                X0(159) T0TNRD - (INTEGER, 1 WORD) TOTAL WORDS IN A BDBE FILE
1699 1 C040                X0(160) FILE - (INTEGER, 3 WORDS) UTDB FILE NAME (**XXXX)
1700 1 C040                X0(162) DATEC - (INTEGER, 1 WORD) RECORD # WHERE DATA GOES NEXT
1701 1 C040                X0(164) UDFERR- (INTEGER, 1 WORD) UTDB FILE ERROR FLAG
1702 1 C040                X0(166) UDBNAME- (INTEGER, 1 WORD) UTDB NAME (XXX)
1703 1 C040                X0(201) MOTOC - (INTEGER, 1 WORD) NUMBER OF TOC ENTRIES
1704 1 C040                X0(201) TOCBUF- (INTEGER, 1200 WORDS) UTDB TOC BUFFER
1705 1 C04***********
1706 1 C05                COMMON USED:
1707 1 C05                EQUVALENC3 (X0(1), LU ),
1708 1 C05                (X0(19), REBPTR),
1709 1 C05                (X0(20), REBSP), (X0(25), EDG ),
1710 1 C05                (X0(89), NAME ), (X0(113), COMMA ),
1711 1 C05                (X0(142), ICR ), (X0(144), COMPTR ),
1712 1 C050                (X0(143), COMBUFF )
1713 1 C05***********
1714 1 C05                RTE ROUTINES USED:
1715 1 C05                NULL, CREAT, EXEC, KCVT, PURGE, WRITF
1716 1 C05***********
1717 1 C05                FDS ROUTINES USED:
1718 1 C05                XDLIS, XDRWT, XRCPR, XREM, XREM, XRDYN,
1719 1 C05                XMGR, XRSET, XRSFL, XRSFR, XRFRM, XDBGS
1720 1 C05***********
1721 1 C00***********
1737  1 BEGIN XSTO,
1738   2  :ST ABFLG TO ZERO (ABORT FLAG)
1739   3  ERREXIT IF UTOB NAME IS NOT VALID TO :ERR2:
1740   4  BUILD REQUEST FOR AWA TOC
1741   5  CALL XRQ TO MAKE MANAGER REQUEST
1742   6  CALL EXEC TO GET AWA TOC
1743   7  ERREXIT IF SIZE OF ATOC > MAXIMUM SIZE TO :ERR2:
1744   8  DO FOR ALL DATABASE CLASS ENTRIES
1745   9  ERREXIT IF NAME SPECIFIED ALREADY EXISTS TO :ERR2:
1746  10  ENDDO
1747  11  SET ABFLG OFF (ERROR MESSAGE FLAG)
1748  12  SET TOTTSIZ = 0 (UTOB TOTAL SIZE)
1749  13  IF WHOLE AWA IS TO BE STORED THEN
1750  14  DO FOR IT, ST, DE, ORDE ATOC ENTRIES
1751  15  DO FOR EACH ENTRY IN THIS CHAIN
1752  16  IF PREFIX IS NOT DOUBLE EXCLAMATION AND
1753  17  PREFIX IS NOT AN AMPERSAND THEN
1754  18  SET STORE/RESTORE BIT ON IN ATOC ENTRY
1755  19  INCREMENT TOTTSIZ BY SIZE OF THIS ELEMENT
1756  20  ENDF
1757  21  ENDDO
1758  22  ELSE
1759  23    CALL XOLIS TO PROCESS LIST TO BE STORED
1760  24    ERREXIT IF ABFLG IS NOT ZERO TO :ERR5:
1761  25    ENDF
1762  26    SET NOTOC = 0 (NUMBER OF UTOB TOC ENTRIES)
1763  27    DO FOR IT, ST, DE, ORDE CHAINS
1764  28    DO FOR EACH ENTRY IN THIS CHAIN
1765  29    IF STORE/RESTORE BIT IS ON THEN
1766  30      TURN STORE/RESTORE BIT OFF
1767  31      BUILD UTOB TOC ENTRY
1768  32      INCREMENT NOTOC BY 1
1769  33    ENDF
1770  34    ENDDO
1771  35  END ELSE
1772  36  ERREXIT IF THERE ARE NO UTOB TOC ENTRIES (NOTOC=0) TO :ERR2:
1773  37  COMPUTE DATREC AS FIRST RECORD AVAILABLE FOR DATA
1774  38  CALL XP ' ' TO CREATE FILE NAME
1775  39  CALL TO CREATE UTOB FOR TOTTSIZ
1776  40  ERREX. ERROR IN CREATE TO :ERR3:
1777  41  CALL XTWR TO WRITE UTOB FILE
1778  42  ERREXIT IF ABFLG IS 4 (ORDE LARGER THAN SPECIFIED) TO :ERR1:
1779  43  ERREXIT IF ABFLG IS 2 (ORTD FILE ERROR) TO :ERR3:
1780  44  ERREXIT IF ABFLG IS 2 (ORTD FILE ERROR) TO :ERR3:
1781  45  CALL WRIT TO WRITE TOC RECORDS AT RECORD 1
1782  46  ERREXIT IF ERROR IN WRIT TO :ERR3:
1783  47  CALL close TO CLOSE UTOB FILE
1784  48  ERREXIT IF ERROR IN CLOSE TO :ERR3:
1785  49  BUILD REQUEST TO ALLOCATE UTOB IN AWA
1786  50  CALL XRQ TO MAKE REQUEST
1787  51  ERREXIT IF AWA OVERFLOW TO :ERR1:
1788  52  EXIT XSTO
1789  53  :ERR1:
1790  54  CALL XRMSG TO DISPLAY MSGNO
1791  55  GO TO :ERR4:
1792  56  :ERR2:
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1794</td>
<td>2 CALL XRM SG TO DISPLAY MSGNO</td>
</tr>
<tr>
<td>1795</td>
<td>2 GO TO :ERR5:</td>
</tr>
<tr>
<td>1796</td>
<td>2 :ERR3:</td>
</tr>
<tr>
<td>1797</td>
<td>2 CALL XRM SG TO DISPLAY MSGNO WITH UTDB FILE ERROR CODE</td>
</tr>
<tr>
<td>1798</td>
<td>2 :ERR4:</td>
</tr>
<tr>
<td>1799</td>
<td>2 CALL CLOSE TO CLOSE UTDB</td>
</tr>
<tr>
<td>1800</td>
<td>2 CALL PURGE TO PURGE UTDB</td>
</tr>
<tr>
<td>1801</td>
<td>2 :ERR5:</td>
</tr>
<tr>
<td>1802</td>
<td>2 CALL XRM SG TO DISPLAY STORE ABORTED/ UTDB NOT CREATED MESSAGE</td>
</tr>
<tr>
<td>1803</td>
<td>1 END XDS TO</td>
</tr>
</tbody>
</table>
FORTRAN CALLING PROCEDURE

CALL XDOTC

*********

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

*********

INPUT

XE COMMON - COMBUF, COMPTR, FLAGS, LU

MANAGER - AWS HEADER AND TOC (SEE MODULE XMAWA)

FILES - DATA BASE FILES AS APPROPRIATE

*********

OUTPUT

XE COMMON - COMPTR, RERBUF, RERPR

PRINTED OUTPUT - AWS OR UTBD TOC DISPLAY

*********

LOCAL VARIABLES

ASZ - TOTAL ALLOCATABLE SIZE OF AWS (SEE MODULE XMAWA)

CMA - FIRST PORTION OF AWS (HEADER AND TOC)

CLASS - ARRAY OF ALLOWABLE CLASS DESIGNATORS FOR TOC DIRECTIVE

CLS - CLASS NAME NUMBER BEING PROCESSED PLUS ONE

CODE - CLASS NUMBERS CORRESPONDING TO ELEMENTS OF 'CLASS'

EOC - END OF TOC CHAIN DESIGNATOR (-32768)

PRE - CURRENT AMOUNT OF FREE SPACE IN AWS (SEE MODULE XMAWA)

NO - ARRAY OF TOC CHAIN HEADS (SEE MODULE XMAWA)

IDIM - TOC ENTRY DIM FIELD (WORD 8)

LINE - BUFFER FOR CONSTRUCTING CURRENT OUTPUT IMAGE

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

OPTION - ARRAY OF SELECTED CHAINS TO LIST IN TOC DISPLAY

ORG - ORIGIN ADDRESS OF AWS (SEE MODULE XMAWA)

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

SIZE - TOC ENTRY SIZE FIELD (WORD 7)

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

TYPE - TOC ENTRY SIZE FIELD (BYTE 2 OF WORD 1)

TYPEID - ARRAY OF TYPE ID CODES FOR DE AND ORDE DISPLAYS

UNIT - LOGICAL UNIT SELECTED FOR OUTPUT OF DISPLAY

********

NOTES

USES CLOSE, EXEC, IAND, IXOR, KCNY, OPEN, READF, XDOTA, XREQ, XRXG, XRMOV, XRSG, XRSP, XRSP, XRUSG

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

IT IS ASSUMED THAT XDOTC AND XDOTA USE THE SAME SIZE BUFFER FOR TOC MANIPULATION; THUS, A DATA BASE FILE MAY NOT HAVE A TOC TOO LARGE FOR THE XDOTC INTERNAL BUFFER.
BECOME XTDC

FOR OUTPUT OF ALL USER CLASSES FROM AMA TO TERMINAL

IF NEXT TOKEN IS NOT EOS
  THEN
    IF TOKEN IS A HYPHEN
      THEN
        INCREMENT TO NEXT TOKEN
        IF TOKEN IS NOT A CHARACTER T
          THEN
            EXIT TO :ERROR3: IF TOKEN IS NOT A CHARACTER P
          SET OUTPUT UNIT FOR LINE PRINTER
          ENDDIF
          INCREMENT TO NEXT TOKEN
          ENDDIF
    IF TOKEN IS NOT EOS
    THEN
      EXIT TO :ERROR4: IF TOKEN IS NOT A COMMA
      INCREMENT TO NEXT TOKEN
      IF TOKEN IS NOT A COMMA
      THEN
        EXIT TO :ERROR5: IF TOKEN DOES NOT INDICATE A VALID CLASS (O, F, I, S OR B)
        SET OPTION FOR INDICATED CLASS
        INCREMENT TO NEXT TOKEN
        ENDDIF
      IF TOKEN IS NOT EOS
      THEN
        EXIT TO :ERROR6: IF NOT LOGGED IN TOC
        GET TYPE AND SET NAME IN HEADER
        CONSTRUCT FILE NAME
        READ FIRST DATA BASE FILE TOC RECORD
        INITIALIZE TOC HEAD TO APPEAR SIMILAR TO AMA TOC
        READ SUBSEQUENT TOC RECORDS
        EXIT TO :ERROR7: IF FILE ACCESS FAILS
        FOR EACH NON-EMPTY TOC CHAIN
        INDEX TO TOC ENTRY POINTED TO BY CHAIN HEAD
        IF NOT FIRST ENTRY IN TABLE, I.E., A PREVIOUS NON-NULL CHAIN EXISTED
        THEN
          MARK PREVIOUS ENTRY AS AN END OF CHAIN
          ENDDIF
        ENDDO
        DO FOR EACH ENTRY IN TOC
        IF NOT MARKED AS AN END OF CHAIN
        THEN
          STORE POINTER TO NEXT SEQUENTIAL TOC ENTRY IN CHAIN POINTER FIELD
          ENDDIF
        ENDDO
        EXCLUDE CHAIN B (DATA BASE FILES) FROM DISPLAY
      ENDDIFF
    ENDDIF
  ENDIF
ENDIF
IF DEBUG AND/OR TRACE FLAGS ARE SET
    THEN
    EXTEND OPTION TO INCLUDE SYSTEM CLASSES (0, 1, 5 AND 7)
    ENDIF
  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

IF PROCESSING AWTOC
THEN
CALL XSTA TO DISPLAY AWTOC USAGE STATISTICS
ENDIF
EXIT XDOTOC

:ERROR5: EXIT WITH INVALID OUTPUT DEVICE ID
:ERROR4: EXIT WITH SYNTAX ERROR
:ERROR5: EXIT WITH INVALID CLASS DESIGNATOR
:ERROR6: EXIT WITH INVALID UTOC FILE NAME
:ERROR7: EXIT WITH UTOC FILE ACCESS ERROR

1 END XDOTOC
INTERNAL XB COMMON USED:

- XB(151) ABFSL - (INTEGER, 1 WORD) ABORT FLAG
- XB(153) MSGNO - (INTEGER, 1 WORD) MESSAGE NUMBER TO BE DISPLAYED
- XB(155) NREQ - (INTEGER, 1 WORD) NUMBER OF REQUESTS IN REQBUF
- XB(158) TOTWRD - (INTEGER, 1 WORD) TOTAL # WORDS IN A FILE
- XB(162) DATREC - (INTEGER, 1 WORD) RECORD # WHERE DATA GOES NEXT
- XB(163) WRNNO - (INTEGER, 1 WORD) INDEX INTO DATBUF WHERE NEXT DATA ELEMENT IS TO START
- XB(166) UIDERR - (INTEGER, 1 WORD) UTDB FILE ERROR FLAG
- XB(166) DRRDE - (INTEGER, 1 WORD) DRDE FILE ERROR FLAG
- XB(169) DRDE3 - (INTEGER, 1 WORD) DRDE TYPE 3 FILE FLAG
- XB(170) TOCINC - (INTEGER, 1 WORD) TOC INDEX FOR CURRENT REQUEST
- XB(171) TOCPTR - (INTEGER, 1 WORD) CURRENT TOC INDEX
- XB(172) DDFIL - (INTEGER, 3 WORDS) DPHE FILE NAME
- XB(173) LFM - (INTEGER, 1 WORD) LENGTH OF 1 COPY (DRDE TO UTDB)
- XB(174) NREC - (INTEGER, 1 WORD) NUMBER RECORDS IN 1 COPY
- XB(201) NTOC - (INTEGER, 1 WORD) NUMBER OF UTDB TOC ENTRIES
- XB(201) TOCBUF - (INTEGER, 1200 WORDS) UTDB TOC BUFFER

COMMON USED:

- EQUIVALENCE (X(1), L, U)
- * (X(1), FLG), (X(19), REQBUF)
- 2620
- 2021
- RTE ROUTINES USED:
- CLOSE, EXEC, KCVT, OPEN, READF, WRITF
- FDS ROUTINES USED:
- XREQ, XREXT, XMOV, XRMSG,
- XRSM, XRSL, XRSPR
BEGIN XDRAT
SET WDRNO = 1 (WORD INDEX INTO DATREC WHERE ELEMENT BEGINS)
SET NDRE = 0 (NUMBER OF AVA REQUESTS IN RENBUF)
DO FOR ALL UTDB TOC ENTRIES UNTIL CLAS IS BRDE
BUILD REQUEST FOR DATA FROM AVA
INCREMENT NDRE BY 1
IF REQUEST BUFFER IS FULL (NDRE=8) THEN
PERFORM READ TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
ENDIF
ENOBO
IF THERE ARE REMAINING REQUESTS (NDRE>0) THEN
SET NEXT REQUEST TO SAY END OF REQUEST LIST
PERFORM READ TO REQUEST DATA FROM MANAGER AND HANDLE OUTPUT TO FILE
IF THERE IS A PARTIAL DATA RECORD LEFT (WDRNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
ENDIF
DO FOR EACH DRE DUTB TOC ENTRY
STORE DATREC IN DRE TOC ENTRY
CALL DRENF TO CREATE FNAME
IF DRE FILE IS TYPE 3 THEN
CALL OPEN TO OPEN FILE AS CORRECT TYPE
ERRIF IF OPEN ERROR TO :ERR1:
DO UNTIL EOF IS READ
CALL READ TO READ 1 RECORD
ERRIF IF READ ERROR TO :ERR1:
STORE RECORD LENGTH AT FRONT AND REAR OF DATA
INCREMENT WDRNO BY LENGTH + 2
IF THERE IS ENOUGH DATA TO WRITE (WDRNO>128) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
ENOBO
IF THERE IS REMAINING DATA (WDRNO>1) THEN
PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
ENDIF
ENDIF
SET DATREC TO NEXT AVAILABLE RECORD FOR DATA
ELSE
CALL OPEN TO OPEN FILE AS TYPE 1
ERRIF IF OPEN ERROR TO :ERR1:
COMPUTE TOTAL SIZE OF FILE IN WORDS
DO UNTIL ALL DATA IS COPIED TO UTDB (SIZE=0)
IF SIZE IS LESS THAN LENGTH TO BE WRITTEN THEN
SET LENGTH = SIZE
ENDIF
CALL READ TO READ LENGTH DATA
ERRIF IF READ ERROR TO :ERR1:
CALL WRITE TO WRITE LENGTH DATA
ERRIF IF WRITE ERROR TO :ERR5:
INCREMENT DATREC BY NUMBER OF RECORDS WRITTEN
DECREMENT SIZE BY LENGTH IN WORDS WRITTEN
ENOBO
ENDIF
CALL CLOSE TO CLOSE DRE FILE
ERRIF IF CLOSE ERROR TO :ERR1:
ENDDO
EXIT XDRAT
BEGIN REMT
2090 2 BEGIN REMT
2091 3 CALL XREX TO MAKE REQUEST
2092 4 ERREX IF THERE IS AN AWA OVERFLOW TO ERR4;
2093 5 DO FOR NUMBER OF REQUESTS (MORE) IN BUFFER
2094 6 CALL EXEC TO GET DATA FROM SAN
2095 7 SET DATREC AND WROD INTO THIS UTDB TOC ENTRY
2096 8 IF THERE IS ENOUGH DATA TO WRITE (WRONO>128) THEN
2097 9 PERFORM WRITE TO OUTPUT DATA TO UTDB FILE
2098 10 ENDIF
2100 3 ENDDO
2101 3 REINITIALIZE REQUEST BUFFER (MORE=0)
2102 2 END REMT
2103 2 *
2105 2 *
2106 2 BEGIN WRITE
2107 3 COMPUTE LENGTH TO WRITE IN WORDS AND RECORDS
2108 3 IF THERE IS NOT AT LEAST 1 RECORD TO WRITE THEN
2109 4 SET LENGTH TO MINIMUM OF 1 RECORD
2110 3 ENDIF
2111 3 IF THIS IS A DRE TYPE 3 FILE THEN
2112 4 ERREX IF THERE ARE MORE BLOCKS THAN SPECIFIED TO ERR2;
2113 3 ENDIF
2114 3 CALL WRITF TO WRITE LENGTH DATA TO UTDB FILE
2115 3 ERREX IF WRITE ERROR TO ERR3;
2116 3 DECREMENT WRONO BY LENGTH
2117 3 INCREMENT DATREC BY LENGTH/128
2118 3 IF THERE IS REMAINING DATA (WRONO>1) THEN
2119 4 CALL XRMOV TO MOVE REMAINING DATA UP IN BUFFER
2120 3 ENDIF
2121 2 END WRITE
2122 2 :ERR1:
2123 2 CALL XRMSG WITH DRE NAME AND RC
2124 2 SET ABFLG TO SAY A DRE ERROR
2125 2 GO TO :ERR3;
2126 2 :ERR2:
2127 2 CALL XRMSG TO SAY DRE FILE LARGER THAN SPECIFIED IN TOC
2128 2 SET ABFLG TO SAY DRE FILE ERROR
2129 2 :ERR3:
2130 2 CALL CLOSE TO CLOSE DRE FILE
2131 2 SET ABFLG TO SAY UTDB FILE ERROR
2132 1 EXIT XDWT
2133 2 :ERRA:
2134 2 SET ABFLG TO SAY AWA OVERFLOW
2135 2 DO FOR REMAINING REQUESTS
2136 2 CALL EXEC TO FREE CLASS NUMBER
2137 2 ENDDO
2138 1 END XDWT
FROM TRAN CALLING PROCEDURE

1   COMMON XR, XE, XBE - XEBT (THREE WORD ARRAY CONTAINING 'XECAL')

2   COMMON XE - SUBSTA

3   COMMON XE - INITIALIZED ACCORDING TO THE VALUE OF SUBSTA

4   (SEE APPROPRIATE INITIALIZATION SUBROUTINE)

5   NOTE:

6   USES XE, XBE, XEBT, XEBT, XEBT, XEBT

7   BEGIN XECAL

8   CASE (:GLOBAL, :DIRECT, :EXECUT, :SEREDT, :INTEDT) SUBSTA (-1 TO 3)

9   :GLOBAL: CALL KEINS TO INITIALIZE GLOBAL COMMON

10  :DIRECT: CALL XEBT TO INITIALIZE DYNAMIC COMMON FOR DIRECTIVES

11  :EXECUT: CALL XEBT TO INITIALIZE DYNAMIC COMMON FOR EXECUTION CONTROL

12  :SEREDT: CALL XEBT TO INITIALIZE DYNAMIC COMMON FOR SEQUENCE EDITING

13  :INTEDT: CALL XEBT TO INITIALIZE DYNAMIC COMMON FOR INTERFACE EDITING

14  END CASE

15  END XECAL
FORTRAN CALLING PROCEDURE

CALL XEIND

INITIALIZE XB COMMON FOR DIRECTIVE LEVEL

INPUT

COMMON XE - CARTAG, FLAGS

XDRM - FDS DIRECTIVE PROMPT FILE

OUTPUT

COMMON XB - DIRECT, NUMDIR

NOTES

USES CLOSE, EXEC, OPEN, PRM, READ, XREXT, XRG, XMSG, XVABN

BEGIN XEIND

READ FDS DIRECTIVE PROMPT FILE

STORE DIRECTIVES IN DYNAMIC COMMON

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

CALL XEINI

*******

INITIALIZE XE AND XB COMMON FOR INTERFACE TABLE EDITING

*******

INPUT

COMMON XE -

MASTA = MASTER STATE

TOKENS = LEXICAL TOKEN VALUES

NOPROC = NUMBER OF ENTRIES IN LIBRARY DIRECTORY

COMTR = INDEX OF NEXT TOKEN IN COMBUF

COMBUF = COMMUNICATIONS BUFFER

PRCNAM = PROCESSOR NAME FROM EXECUTION CONTROLLER

LIBRARY -

INTNam = OLD INTERFACE TABLE TO BE EDITED

<XXXXX = DEFAULT INTERFACE TABLE

>XXXXX = PROMPT TABLE

*******

OUTPUT

COMMON XE -

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

REBUF = REQUEST BUFFER FOR AWI INPUTS

PRCNAM = PROCESSOR NAME ON INTE DIRECTIVE

COMMON XB -

WKBLEN = LENGTH OF WKBUF (CONSTANT)

WKBUF = WORKING BUFFER CONTAINING INE. TABLE WITH

SHORT PROMPTS

DIRECT = SUPPORTED INTE DIRECTIVES

NEMID = NO. OF ENTRIES IN DIRECT

MENTAB = NEW INTE. TABLE NAME

NUMARG = NO. OF ARGUMENTS

WARG = NO. OF WORDS IN SPC. AND HEADER OF WKBUF

NITLEN = NO. OF WORDS IN LITERAL AREA

LITPTR = 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 ROUTINES

XRMOV = EXEC

XRMGR = CLOSE

XREXY = OPEN

XREG = READF

XRUPK = XRNSG

XRCPN

BEGIN XEINI

IF CALLED AS A RESULT INTE DIRECTIVE, THEN
POSITION TO 1ST TOKEN AFTER 'INTE'

EXIT IF TOKEN IS NOT ',' :ERR1:

INCREMENT TO NEXT TOKEN

EXIT IF TOKEN IS NOT A NAME :ERR2:

SET PRCNAM TO THIS NAME

INCREMENT TO NEXT TOKEN

SET INTNAM TO 0

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

INCREMENT TO NEXT TOKEN

EXIT IF TOKEN IS NOT EOS, THEN

IF TOKEN IS A NAME, THEN

SET INTNAM TO THE NAME

INCREMENT TO NEXT TOKEN

ENDIF

EXIT IF TOKEN IS NOT EOS, THEN

EXIT IF TOKEN IS NOT A COMMA :ERR2:

INCREMENT TO NEXT TOKEN

IF TOKEN IS A NAME, THEN

SET NEWTAB TO THE NAME

INCREMENT TO NEXT TOKEN

ENDIF

EXIT IF TOKEN IS NOT EOS :ERR2:

ENDIF

ELSE

INTNAM AND PRCNAM ARE INITIALIZED BY THE EXECUTION CONTROLLER

NEWTAB IS SET TO 'BINTAB'

ISSUE MESSAGE THAT INTERFACE TABLE EDITOR HAS BEEN INVOKED

ENDIF

MAKE MANAGER REQUEST FOR LIBRARY DIRECTORY FILE

EXIT IF REQUEST IS UNSUCCESSFUL :ERR11:

START SEARCH UNTIL ALL OF DIRECTORY IS SEARCHED, OR

EXIT IF PROCESSOR PRCNAM IS FOUND

ELSE

INCREMENT TO NEXT ENTRY

END LOOP

EXIT :ERR8:

END SEARCH

SET IVERS TO VERSION NO. OF DIRECTORY ENTRY FOUND

IF INTNAM = 0, THEN

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

OPEN THE DEFAULT INTERFACE TABLE FILE

EXIT IF OPEN FAILED :ERR12:

READ THE DEFAULT INTERFACE TABLE

EXIT IF THE READ FAILED :ERR16:

READ LITERAL RECORD FROM DEFAULT INTERFACE TABLE FILE INTO BOTTOM

OF WKBUF

EXIT IF READ FAILED :ERR14:

ELSE

MAKE MANAGER REQUEST FOR INTNAM INTERFACE TABLE

EXIT IF ERROR OR COULD NOT FIND :ERR4:

ENDIF

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

CONSTRUCT NAME OF PROMPT TABLE AS "PRCNAM"

OPEN THE PROMPT TABLE

EXIT IF OPEN FAILED :ERR15:
READ 2ND RECORD OF THIS PROMPT TABLE
ErroREXIT 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 NOT MATCH CURRENT VERSION
:ERR11: XEINI OP 1 INITIALIZATION ERROR
:ERR12: XEINI OP 2 INITIALIZATION ERROR
:ERR13: XEINI OP 3 INITIALIZATION ERROR
:ERR14: XEINI OP 4 INITIALIZATION ERROR
:ERR15: XEINI OP 5 INITIALIZATION ERROR
:ERR16: XEINI OP 6 INITIALIZATION ERROR
:ERR17: XEINI OP 7 INITIALIZATION ERROR
SET SUBSTATE TO DIRECTIVE LEVEL TO INDICATE THE ERROR
END XEINI
**FORTRAN CALLING PROCEDURE**

**CALL XEINS**

**INITIALIZE XE AND XB COMMON FOR SEQUENCE TABLE EDITING**

**INPUT**

**COMMON XE - CARTRG, COMBUF, COMPTR, LU, NOPROC, TOKENS**

**OUTPUT**

**COMMON XB - DEBUG, DIRECT, NEWTAB, NUMDIR, NUMENT,**

**OLDTAB, PRMTAB, WKBNG, WKBUF, XLIBD**

**NOTES**

**USES FILES - XSPRM - SEQUENCE TABLE EDITOR PROMPT FILE**

**USES Routines**

**EXEC**

**IAND**

**OPEN**

**READ**

**XCPTR**

**XEQ**

**XEXT**

**XMOV**

**XMSG**

**XDBG**

**C**********
1 BEGIN XEMS
2  BUILD XAM REQUEST TO RETRIEVE "XLIBD"
3  ERROR IF TOKEN IS NOT COMM :ERROR2:
4  INCREMENT TO NEXT TOKEN
5  IF TOKEN IS NAME, THEN
6  BUILD XAM REQUEST TO RETRIEVE OLDTAB
7  RETAIN THIS NAME AS OLDTAB
8  ELSE
9  ENDIF
10  SET OLDTAB TO ZERO
11  ENDIF
12  ERROR IF TOKEN IS NOT COMM :ERROR2:
13  INCREMENT TO NEXT TOKEN
14  ERROR IF TOKEN IS NAME :ERROR2:
15  RETAIN THIS NAME AS NEWTAB
16  IF NEWTAB NOT EQUAL TO OLDTAB, THEN
17  BUILD XAM REQUEST TO VERIFY EXISTENCE OF NEWTAB
18  ENDIF
19  CALL XRQ TO PROCESS XAM REQUEST(S)
20  IF NEWTAB NOT EQUAL TO OLDTAB, THEN
21  ERROR IF NO ERROR FROM XRQ :ERROR12:
22  ENDIF
23  IF OLDTAB NOT ZERO, THEN
24    ERROR IF 2ND REQUEST (RETRIEVE OLDTAB) FAILED :ERROR4:
25    SET PROMPT MODE AS UPDATE
26    SET NO. ENTRIES AS OLDTAB SIZE / 7
27    READ OLDTAB INTO WORKING BUFFER
28    ERROR IF FIRST REQUEST (RETRIEVE "XLIBD") FAILED :ERROR1:
29    READ "XLIBD INTO COMMON"
30  ELSE
31  ENDIF
32  SET PROMPT MODE AS CREATE
33  SET NO. ENTRIES TO ZERO
34  ENDIF
35  OPEN, READ AND CLOSE FILE XSPRM
36  SET COUNT AND SEREDIT DIRECTIVES INTO XSP COMMON
37  EXIT XEMS
38  ERROR1: CALL XRMSG - 'INITIALIZATION ERROR ....'
39  ERROR2: CALL XRMSG - 'SYNTAX ERROR'
40  ERROR4: DEFAULT MESSAGE TO '....NOT FOUND'
41  IF ERROR WAS NO XAM SPACE THEN
42  SET MSG TO '....NO XAM SPACE'
43  ENDIF
44  CALL XRMSG TO DISPLAY MESSAGE
45  ERROR12: CALL XRMSG - '.... ALREADY EXISTS'
46  END XEMS
FORTRAN CALLING PROCEDURE

CALL XEINT

INTERFACE TABLE LITERAL AREA INITIALIZATION

INPUT

COMMON XB - LITPTR, NUMARG, WKBNG, WKBUF

NOTE: WKBUF IS INPUT WITH THE INTERFACE

TABLE'S CHARACTERISTICS IN THE TOP

AND THE "PACKED" LITERAL AREA IN THE

BOTTOM.

OUTPUT

COMMON XB - LITPTR, LITOWN, NARC, W'5UF

NOTE: WKBUF IS OUTPUT WITH THE LITERAL AREAS

IN THEIR "UNPACKED" FORM.

USES ROUTINES

XIEXT

XRMV

XRMKB

XRMSG

XRESET

**********

**********
1 BEGIN XEINT
2 DO UNTIL ALL LITERAL ENTRIES HAVE BEEN PROCESSED
3 START SEARCH UNTIL ALL ARGUMENTS SEARCHED
4 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
5 E-BIT IS ON
6 BUILD 3 WORD LITERAL ENTRY (3, J-SUB, J-SUB)
7 EXIT IF 'DISP' FIELD FOR ARGUMENT = DISP. OF LITERAL ENTRY, AND
8 D-BIT IS ON
9 IF ARGUMENT IS COMPLETE (C-BIT ON), THEN
10 CREATE BIT MASK WORD(S) IN NEW LITERAL ENTRY AREA
11 ELSE
12 MOVE BIT MASK WORDS UP TO NEW LITERAL ENTRY AREA
13 ENDIF
14 DO UNTIL ALL BITS OF BIT MASK PROCESSED
15 DETERMINE NUMBER OF CONTIGUOUS BITS ON (OR OFF)
16 AND MOVE CORRESPONDING NUMBER OF DATA WORDS
17 (OR O'S) INTO NEW LITERAL ENTRY AREA
18 ENDDO
19 ORELSE
20 INCREMENT TO NEXT ARGUMENT
21 ENDO
22 ERREXIT IF NOT CALLED BY LIBRARY MAINTENANCE :ERR5:
23 SKIP TO NEXT LITERAL ENTRY
24 ENDSRNCH
25 2 ENDCO
26 1 EXIT TO :RETURN:
27 2 :ERR5: CALL XRMSG - 'SYSTEM INITIALIZATION ERR 5'
28 2 :RETURN:
29 1 END XEINT
FORTRAN CALLING PROCEDURE

CALL XEIXX

initialize XE and XB COMMON FOR EXECUTION CONTROLLER

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

COMMON XB - XEBUF, XBREX, XBRFG, XBRFG2, XBRFG3, XBRFG4

OUTPUT

COMMON XB - XEBUF, XBREX, XBRFG, XBRFG2, XBRFG3, XBRFG4

NOTES

USES EXEC, PRM, XREG, XREX, XRIG, XRMG, XDEG, XVABN

**********
BEGIN XEINX

1: IF INITIALIZATION FROM DIRECTIVE
2: THEN
3: IF DIRECTIVE IS NAME
4: THEN
5: EXIT TO :ERROR1: IF NEXT TOKEN IS NOT EOS
6: ELSE
7: IF DIRECTIVE IS AUTO
8: THEN
9: IF TOKEN IS A VYPHEN
10: THEN
11: INCREMENT TO NEXT TOKEN
12: EXIT TO :ERROR3: IF TOKEN IS NOT THE NAME 'T'
13: CHANGE EXECUTION CONTROL MODE TO AUTO-T
14: INCREMENT TO NEXT TOKEN
15: ENEIF
16: ENEIF
17: EXIT TO :ERROR4: IF NEXT TOKEN IS NOT A COMMA
18: INCREMENT TO NEXT TOKEN
19: EXIT TO :ERROR2: IF NEXT TOKEN 1:: NOT A NAME
20: STORE NAME IN SEQNAM
21: CALL XREG TO RETRIEVE SEQUENCE TABLE
22: EXIT TO :ERROR4: IF NON-ZERO RETURN CODE
23: INCREMENT TO NEXT TOKEN
24: SET SENSTA TO FIRST SEQUENCE NUMBER
25: SET SEGNAM TO LAST SEQUENCE NUMBER
26: IF TOKEN NOT EOS
27: THEN
28: EXIT TO :ERROR2: IF TOKEN NOT A COMMA
29: INCREMENT TO NEXT TOKEN
30: IF TOKEN IS AN INTEGER
31: THEN
32: STORE STARTING RANGE NUMBER
33: SEARCH SEQUENCE NUMBERS FOR STARTING VALUE
34: EXIT TO :ERROR1: IF NOT FOUND
35: INCREMENT TO NEXT TOKEN
36: ENEIF
37: IF TOKEN NOT E::
38: THEN
39: EXIT TO :ERROR2: IF TOKEN NOT A COMMA
40: INCREMENT TO NEXT TOKEN
41: EXIT TO :ERROR2: IF TOKEN NOT AN INTEGER
42: ERROR TO :ERROR2: IF 'ENDING SEQ # < BEGINNING SEQ #
43: SEARCH SEQUENCE NUMBERS FOR ENDING VALUE
44: EXIT TO :ERROR1: IF NOT FOUND
45: INCREMENT TO NEXT TOKEN
46: EXIT TO :ERROR2: IF TOKEN NO. E::
47: ENEIF
48: ENEIF
49: SET SENSTP TO SENSTA
50: ENEIF
51: ENEIF
52: INITIALIZE DYNAMIC COMMON WITH NUMBER OF PROCESSORS AND DIRECTORY NAME TABLE
53: EXIT TO :ERROR1: IF INITIALIZATION FAILS
54: 1 EXIT KEINF
55: 2 :ERROR1: INITIALIZATION FAILURE TERMINATION
56: 2 :ERROR2: SET SUBSTA 111 DIRECTIVE LEVEL & EXIT WITH SYNTAX ERROR
57:
551 2 :ERROR3: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH INVALID TRACE OPTION
552 2 :ERROR4: SET SUBSTA TO DIRECTIVE LEVEL
553 2 IF ERROR WAS NO AWA SPACE THEN
554 2 EXIT WITH NO AWA SPACE ERROR
555 2 ELSE
556 2 EXIT WITH SEQUENCE TABLE NOT FOUND ERROR
557 2 ENDIF
558 2 :ERROR5: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH RANGE ERROR
559 2 :ERROR13: SET SUBSTA TO DIRECTIVE LEVEL & EXIT WITH NUMBER NOT FOUND
560 1 EXIT ERROR
562 1 CO**********
563 1 CO
564 1 CO
565 1 CO
566 1 CO
567 1 CO
568 1 CO
569 1 CO
570 1 CO
571 1 CO
572 1 CO
573 1 CO
574 1 CO
575 1 CO
576 1 CO
577 1 CO
578 1 CO
579 1 CO
580 1 CO
581 1 CO
582 1 CO
583 1 CO
584 1 CO
585 1 CO
586 1 CO
587 1 CO
588 1 CO
589 1 CO
590 1 CO
591 1 CO
592 1 CO
593 1 CO
594 1 CO
595 1 CO
596 1 *
597 1 *
598 1 *
599 1 BEGIN XELDS
600 2 :XELDS:
601 2 CALL &ENT TO RETRIEVE PARAMETERS AND RETURN ADDRESS
602 2 MOVE THE ADDRESS OF THE SEGMENT NAME INTO EXEC PARAMETER LIST
603 2 CALL EXEC TO LOAD AND PASS CONTROL TO SEGMENT
604 2 :XERTH:
605 2 RETURN TO MAIN PROGRAM
606 1 END XELDS

FORTRAN CALLING PROCEDURE:
CALL XELDS (SEGNM) TO CALL SEGMENT FROM MAIN
CALL XERTH TO RETURN TO MAIN PROGRAM

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

INPUTS IN CALLING SEQUENCE:
SEGNM - (INTEGER, 3 WORDS) ARRAY CONTAINING NAME OF THE
SEGMENT TO BE LOADED

SUBROUTINES AND FUNCTIONS CALLED:
EXEC

NOTES
1) XELDS CONTAINS 2 ENTRY POINTS: XELDS AND XERTH
2) SEGMENT CALLED BY MAIN MUST BEGIN WITH A PROGRAM
STATEMENT - TYPE 3
3) IN ORDER TO RETURN TO MAIN, A SEGMENT MUST "CALL XERTH"
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

RETURN TO MAIN PROGRAM
END XELDS
1 BEGIN EXEC
2 RETRIEVE SCHEDULING PARAMETERS AND SET LU, CLASS, QUAL & FLAGS
3 CALL XEINI TO INITIALIZE GLOBAL COMMON
4 DO FOREVER -- TERMINATES INSIDE HANDLER FOR XOFF
5 CALL STMCA FOR INPUT OF DIRECTIVE
6 IF ERROR OR NOT A VALID DIRECTIVE NAME
7 THEN ISSUE MESSAGE E06
8 ELSE
9 IF NAME IS INTE
10 THEN SET STATES TO INTE LEVEL
11 CALL XEINI TO INITIALIZE DYNAMIC COMMON
12 EXIT TO :RESET; IF ERROR
13 CALL XINTE TO EDIT TABLE
14 ELSE IF NAME IS SRE
15 THEN SET STATES TO SRE LEVEL
16 CALL XEIMS TO INITIALIZE DYNAMIC COMMON
17 EXIT TO :RESET; IF ERROR
18 CALL XSENE TO EDIT TABLE
19 ELSE IF NAME IS FOR SOME EXECUTION CONTROL OPTION
20 THEN SET STATES TO APPROPRIATE EXECUTION CONTROL MODE
21 DO UNTIL MASSTA IS AT DIRECTIVE LEVEL
22 CALL XEINIT TO INITIALIZE DYNAMIC COMMON
23 EXIT TO :RESET; IF ERROR
24 CALL XNAME TO PERFORM EXECUTIONS
25 IF SUBSTA IS SET TO SRE LEVEL
26 THEN CALL XEIMS TO REINITIALIZE DYNAMIC COMMON
27 EXIT TO :RESET; IF ERROR
28 CALL XSENE TO SUPPORT EXECUTION CONTROL
29 ENDIF
30 IF SUBSTA IS SET TO INTE LEVEL
31 THEN CALL XEIMT TO REINITIALIZE DYNAMIC COMMON
32 EXIT TO :RESET; IF ERROR
33 CALL XINTE TO SUPPORT EXECUTION CONTROL
34 ENDIF
35 IF EXECUTION MODE WAS SEMI OR AUTO
36 THEN CALL XESCNO TO PURGE ANY RESIDUAL SCAN CONTROL DATA AND FILES
37 ENDIF
38 ELSE CALL APPROPRIATE DIRECTIVE HANDLER VIA XDCL?
39 ENDIF
40 ENDIF
41 :RESET;
42 IF SUBSTA IS NOT DIRECTIVE LEVEL
43 THEN CALL XEIND TO REINITIALIZE DYNAMIC COMMON
44 ENDIF
45 ENDIF
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
FORTRAN CALLING PROCEDURE

CALL XINIX

MAIN PROGRAM FOR INTERFACE TABLE EDITOR

INPUT

COMMON XE -

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

COMMON XB -

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

PRMTNO = CURRENT PROMPT MODE. SET TO 5 (MODIFY MODE) IF 'V' 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)

NOTE -

USES ROUTINES
EXEC
XREO
XILIT
XIPRM
XINPT
001 CD$ KXBMOV
002 CD$ KXBMSE
003 CD$ XTCON
004 E---------
005 1 "THE INTERFACE TABLE EDITOR IS ENTERED AS A RESULT OF THE 'INT?' DIRECTIVE
006 1 OR FROM THE EXECUTION CONTROLLER TO COMPLETE AN INTERFACE TABLE. THE DIRECTIVE
007 1 PROVIDES THE NAME OF THE TABLE TO BE EDITED AND THE NAME FOR THE NEW
008 1 TABLE. THE EDITOR INTERACTS WITH THE USER IN ORDER TO ACQUIRE DATA VALUES
009 1 OR VARIABLE NAMES FOR EACH OF THE PARAMETERS IN THE INTERFACE TABLE.
010 1 NOTE: ALL INITIALIZATION, INCLUDING WKBUF (OLD INTERFACE TABLE),
011 1 HAS BEEN PERFORMED BY XEINI.
012 1 * 1
013 1 BEGIN XEINI
014 2 IF BP (NO. OF PARAMETERS) > 0, THEN
015 3 SET ARGNO (NO. OF CURRENT ARGUMENT BEING PROCESSED) TO 0
016 3 PRMTNO = 1 => CREATE I MODE
017 3 PRMTNO = 3 => CREATE A MODE
018 3 PRMTNO = 4 => CREATE CONTINUE MODE
019 3 PRMTNO = 5 => MODIFY MODE
020 3 DO UNTIL 'EXIT' OR 'X' IS ENTERED
021 4 CALL XIPRM TO CONSTRUCT A PROMPT BASED ON PRMTNO, SIZE, TYPE, AND STATUS
022 4 OF NEXT ARGUMENT
023 4 CALL XTCON TO PROMPT USER AND RETURN PARSED INPUT
024 4 IF 'I' WAS NOT ENTERED, THEN
025 5 IF '{' WAS ENTERED, THEN
026 6 SET PRMTNO TO 5
027 6 ELSE IF NOTHING WAS ENTERED (I.E. TOKEN IS CD$), THEN
028 7 INCREMENT TO NEXT ARGUMENT
029 7 ELSE
030 7 CALL XIMPT TO PROCESS THE USER'S INPUT
031 6 ENDIF
032 5 ENDIF
033 4 ENDDO
034 3 IF 'I' WAS ENTERED, THEN
035 4 SET RETURN CODE INDICATING 'I' (I.E. MASSTO = 0)
036 4 ELSE
037 4 COMPRESS THE LITERAL LIST AREA
038 3 ENDIF
039 2 ENDIF
040 2 STORE INTERFACE TABLE AS NEWNAME
041 2 IF STORE INTO AWA FAILED, THEN
042 3 SET MASSTO TO INDICATE DIRECTIVE LEVEL (=0)
043 3 ELSE
044 3 SET GOOD RETURN CODE
045 2 ENDIF
046 1 END XEINI
1 * CONSTRUCT PROMPT TO BE ISSUED
2 BEGIN
3 DO UNTIL A PROMPT IS CONSTRUCTED
4 IF PRMTNL 5, THEN
5 CONSTRUCT A "!" PROMPT
6 ELSE
7 IF PRMTNL 4 (CONTINUE MODE), OR
8 PRMTNL 6 (CONTINUE HERE MODE), THEN
9 IF ARGNO IS A SCALAR, THEN
10 CONSTRUCT PROMPT AS \"ARG\", OR \"ARG\"
11 ELSE
12 IF PRMTNL NOT = 6, THEN
13 COMPUTE ISUB AS NEXT EMPTY ELEMENT BEYOND LAST ENTERED (LASTE)
14 ELSE
15 SET PRMTNL 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 PRMTNL TO MOOSAV (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\",
24 ELSE
25 CONSTRUCT PROMPT AS \"ARG\"(ISUB)
26 ENDIF
27 IF ARGNO = BP, THEN
28 SET PRMTNL TO 5
29 ELSE
30 INCREMENT ARGNO TO NEXT PARAMETER
31 SET IFLAG TO 1, 2, OR 3 INDICATING 1, 0, OR 00
32 SET LAST ENTERED INDICATOR (LASTE) TO 0
33 IF PRMTNL = 3, THEN
34 IF SOME DATA VALUE(S) OR PARAM NAME EXISTS FOR ARGNO, THEN
35 CALL XLS2O TO LIST DATA FOR THIS ARGUMENT
36 ENDIF
37 CONSTRUCT PROMPT AS \"ARG\" OR 0 OR #
38 ELSE
39 IF ARGNO IS MARKED INCOMPLETE, THEN
40 IF A PARTIAL LITERAL LIST EXISTS, OR
41 THIS ARGUMENT IS A SCALAR, THEN
42 COMPUTE ISUB AS FIRST EMPTY ELEMENT
43 IF DOUBLY SUBSCRIPTED PARAMETER, THEN
44 COMPUTE I & J FROM ISUB AND I-DIMENSION (IDIM)
45 CONSTRUCT PROMPT \"ARG\",
46 ELSE
47 CONSTRUCT PROMPT \"ARG\"(ISUB)
48 ENDIF
49 ELSE
50 CONSTRUCT PROMPT AS \"ARG\", OR 0 OR #
51 ENDIF
52 ENDIF
53 ENDIF
54 ENDIF
55 ENDIF
56 ENDIF
57 ENDIF
58 END
59 5.9
FOR calling procedure

CALL XIPMT

**+++**

PROMPT DIRECTIVE PROCESSOR

**+++**

INPUT

**+++**

COMMON XE - COMBUF, COMPTA, TOKENS

**+++**

OUTPUT

**+++**

COMMON XE - ARGNO, PRNTMD

**+++**

*XIPMT PROCESS IS THE PROMPT DIRECTIVE

BEGIN XIPMT

2 ERRRTT:

4 IF TOKEN IS NOT COMMA :ERROR2:

10 POSITION TO NEXT TOKEN

20 ERRRTT IF TOKEN IS NOT NAME :ERROR2:

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

IF NAME IS 'H', THEN

7 SET PRNTMD TO 1

3 ELSE

7 IF NAME IS 'A', THEN

7 SET PRNTMD TO 3

3 ELSE

7 ERRRTT :ERROR2:

7 ENDIF

ENDIF

2 SET ARGNO TO 0

1 EXIT TO :RETURN:

2 :ERROR2: CALL XMSG FOR 'INVALID SYNTAX'

2 :RETURN:

1 END XIPMT
1 * XILST PROCESSES THE LIST DIRECTIVE

1 BEGIN XILST
2 IF Token IS '->', THEN
3   POSITION TO NEXT TOKEN
4   ERREXIT IF Token IS NOT NAME ; ERR02 : 
5   ERREXIT IF NAME IS NOT 'C', 'V', OR 'A' ; ERR02 : 
6   SET MODEFG TO INDICATE SPECIFIED MODE (C=1, V=2, A=3)
7   POSITION TO NEXT TOKEN
8 ELSE
9   SET MODEFG TO 2
10  ENDF
11 IF Token IS EOS, THEN
12   WRITE A HEADER LINE INDICATING TABLE NAME, PROCESSOR VERSION
13   AND STATUS
14 DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
15   IF MODEFG = 1 OR MODEFG = 3, THEN
16     CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
17     ENDF
18   ELSE
19     IF MODEFG = 2 OR MODEFG = 3, THEN
20       CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
21     ENDF
22   ENDOO
23 ELSE
24   DO UNTIL EOS IS REACHED
25     ERREXIT IF Token IS NOT COMA ; ERR02 : 
26     ERREXIT IF NEXT TOKEN IS NOT NAME ; ERR02 : 
27     SET ARGO TO 1
28     START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
29     EXIT IF NAME = ARG0'S NAME IN PROPT TABLE
30     IF MODEFG = 1 OR MODEFG = 3, THEN
31       CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
32       ENDF
33     ELSE
34       IF MODEFG = 2 OR MODEFG = 3, THEN
35         CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
36       ENDF
37     ENDOO
38     ENDOO
39     ENDLOO
40     PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
41     ENDLOO
42     INCREMENT TO NEXT TOKEN
43   ENDOO
44   IF Token IS EOS, THEN
45     ERREXIT IF Token IS NOT COMA ; ERR02 : 
46     ERREXIT IF NEXT TOKEN IS NOT NAME ; ERR02 : 
47     SET ARGO TO 1
48     START SEARCH DO UNTIL ALL ARGUMENTS HAVE BEEN PROCESSED
49     EXIT IF NAME = ARG0'S NAME IN PROPT TABLE
50     IF MODEFG = 1 OR MODEFG = 3, THEN
51       CALL XICHR TO WRITE CHARACTERISTICS OF THIS ARGUMENT
52       ENDF
53     ELSE
54       IF MODEFG = 2 OR MODEFG = 3, THEN
55         CALL XILSD TO WRITE DATA VALUES OF THIS ARGUMENT
56       ENDF
57     ENDOO
58     ENDLOO
59     PRINT MESSAGE THAT NAME IS NOT A VALID PARAMETER
60     ENDLOO
61     INCREMENT TO NEXT TOKEN
62   ENDOO
63 ELSE
64   EXIT TO :RETURN:
65   :ERR02 : CALL XRMSS TO WRITE 'INVALID SYNTAX'
66   :RETURN:
67   END XILST
FORTRAN CALLING PROCEDURE

CALL XISUB

EFFECTIVE SUBSCRIPT CALCULATION ROUTINE

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - IDIM, ISIZE, LEN EFF

OUTPUT

COMMON XB - IRET, ISUB

USES ROUTINES

XRMSG

XISUB

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

BEGIN XISUB

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:

IF IDIM FOR THIS ARGUMENT > 0, THEN

ERREXIT IF SPECIFIED INTEGER VALUE > IDIM :ERR16:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT COMMA :ERR15:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT INTEGER VALUE :ERR14:

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

ELSE

SET ISUB TO INTEGER VALUE

ENDIF

ERREXIT IF ISUB > 4**XE :ERR16:

INCREMENT TO NEXT TOKEN

ERREXIT IF TOKEN IS NOT RIGHT PARENTHESIS :ERR14:

EXIT TO RETURN:

:ERR14: CALL XRMSG - 'INVALID SUBSCRIPT SYNTAX'

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

:ERR16: CALL XRMSG - 'INVALID SUBSCRIPT VALUE'

RETURN:

END XISUB
FORTRAN CALLING PROCEDURE

CALL XIDAT

LITERAL DATA PROCESSOR

INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XB - CFLAG, COMFLG, DFLAG, IARG, IARGA,
ISIZE, ISIZE, LTYPE, LNEFF, LITDSP,
LITOWN, LITSIZE, MODAY, MX8TM, NOBITM,
PRMND, SFLAG, WKBNG, WKBUF

OUTPUT

COMMON XB - IRETC, ISUB, LITDSP, LITOWN, MX8TH,
PRMND, WKBUF

INTERNAL VARIABLES

COMMON XS -

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

GROUPED

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

USES ROUTINES

XISUB
XRBIT
XRMOV
XRMSG
1 CD$  XRNDB
2 CD$  XRES
3 CD$  XINT
4 C*******
5 * XIDAT PROCESSES THE INPUT LITERAL LIST
6 1 BEGIN XIDAT
7 2 DO UNTIL EOS IS REACHED, OR
8 3 UNTIL AN ERROR OCCURS
9 4 IF THIS IS A DATA ELEMENT, THEN
10 5 (I.E. INTEGER, REAL, DOUBLE OR CHAR.)
11 6 IF ARG. TYPE (ITYPE) IS INTEGER, REAL, OR DOUBLE, THEN
12 7 EREXIT IF DATA TYPE IS NOT SAME AS ITYPE :ERR10:
13 8 SET LENGTH TO BE MOVED (LENNOV) TO EFFECTIVE LENGTH FOR DATA
14 9 OF ARG.'S TYPE (LENEFF) (WILL BE 1, 2, OR 3 WORDS)
15 10 ELSE
16 11 IF ARG. TYPE IS FREE, THEN
17 12 SET LENGTH TO BE MOVED (LENNOV) TO EFFECTIVE LENGTH
18 13 FOR DATA INPUT
19 14 ELSE THIS MUST BE CHARACTER DATA BEING INPUT
20 15 EREXIT IF ARGUMENT'S TYPE IS NOT CHARACTER DATA :ERR10:
21 16 INCREMENT TOKEN POINTER TO COUNT OF CHARACTERS
22 17 COMPUTE NO. OF WORDS IN INPUT CHARACTER STRING
23 18 EREXIT IF NO. WORDS (LENNOV) > EFFECTIVE LENGTH OF
24 19 THIS ARGUMENT'S DATA (LENEFF) :ERR10:
25 20 ENDIF
26 21 ENDIF
27 22 INCREMENT TOKEN POINTER TO THE DATA INPUT
28 23 VERIFY THAT SUFFICIENT SPACE EXISTS IN LITERAL AREA OF THIS
24 24 ARGUMENT FOR DATA INPUT (ISUB <= ISIZE - LENNOV + 1)
25 25 EREXIT IF INSUFFICIENT SPACE :ERR11:
26 26 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
27 27 ALLOCATE AND INITIALIZE A LITERAL AREA FOR THIS ARGUMENT
28 28 ENDIF
29 29 MOVE DATA FROM INPUT COMMUNICATIONS BUFFER TO LITERAL AREA
30 30 SET NUMBER OF ELEMENTS COMPLETED (NUMCMP) TO 1 OR, FOR A FREE
31 31 ARGUMENT, TO LENNOV
32 32 IF LENNOV < LENEFF (ONLY POSSIBLE FOR CHARACTER DATA), THEN
33 33 MOVE LENEFF - LENNOV BLANKS INTO LITERAL AREA AS A FILL
34 34 ENDIF
35 35 INCREMENT TO NEXT TOKEN
36 36 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY LENNOV
37 37 SET LAST ENTERED INDICATOR (LASTE) TO ISUB - 1
38 38 ELSE Если ТОКЕН ЗНАЧИТ НЕ-ДАННЫЙ ЭЛЕМЕНТ
39 39 IF THIS IS A "C", THEN
40 40 CALL ISUB TO CALCULATE EFFECTIVE SUBSCRIPT (ISUB)
41 41 BASED ON INPUT SUBSCRIPT, ARGUMENT TYPE (ITYPE) AND
42 42 SECONDARY DIMENSION (IDIM)
43 43 EXIT XIDAT IF ERROR (IHERE < 0)
44 44 ELSE
45 45 IF THIS IS AN "E", THEN
46 46 IF DATA DOES NOT EXIST FOR THIS ARGUMENT, THEN
47 47 CLEAR THE PARAMETER FIELD IN ARGUMENT'S CHARACTERISTICS
48 48 ELSE
49 49 MARK ONE ELEMENT OF THIS ARGUMENT AT ISUB AS INCOMPLETE
50 50 INCREMENT EFFECTIVE SUBSCRIPT (ISUB) BY EFFECTIVE
51 51 LENGTH OF ONE ELEMENT (LENEFF)
52 52 ENDIF
53 53 TURN OFF COMPLETE FLAGS FOR THIS ARGUMENT AND INTERFACE TABLE
54 54 INCREMENT TO NEXT TOKEN
55 55
615  1 EXIT TO :RETURN:
616  2  :ERR02: CALL XERMSG "INVALID SYNTAX"
617  2  :ERR07: CALL XERMSG "ONLY DATA VALID TO RIGHT OF =
618  2  :ERR10: CALL XERMSG "DATA TYPE INCOMPATIBLE WITH TYPE OF ARGUMENT"
619  2  :RETURN:
620  1 END XIDAT
1 * XINPT PROCESSES THE USER'S INPUT TEXT
2 BEGIN XINPT
3   IF 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 'n', THEN
8       POSITION TO NEXT TOKEN
9     IF TOKEN IS '0', THEN
10    SET IFLAG TO 10
11    POSITION TO NEXT TOKEN
12 ELSE
13    SET IFLAG TO 1
14 ENDIF
15 ELSE
16    IF TOKEN IS '0', THEN
17      SET IFLAG TO 0
18    ENDIF
19 ENDIF
20 IF IFLAG NOT SET, THEN
21 CASE NAME (:EXIT; :PROMPT; :LIST;)
22 ERREXIT IF ANOTHER TOKEN Follows :ERROR2:
23 :EXIT: SET IRET TO 0 THAT PROMPTING LOOP TERMINATES
24 :PROMPT: CALL XIPMT TO PROCESS PROMPT DIRECTIVE
25 :LIST: CALL XLIST TO PROCESS LIST DIRECTIVE
26 END CASE
27 ENDIF
28 START SEARCH UNTIL IF ENTRIES
29 EXIT IF NAME FOUND IN PROMPT TABLE
30 SET ARNO TO ENTRY NO.
31 SET ISUB TO 1
32 ORELSE
33 INCREMENT TO NEXT PROMPT TABLE ENTRY
34 ENDDO
35 ERREXIT :ERROR10:
36 ENDS SEARCH
37 ERREXIT IF IFLAG IS NOT SAME AS I/O TYPE OF ARGUMENT :ERROR5:
38 ENDIF
39 IF NEXT TOKEN IS A NAME, THEN
40 CALL XIPAR TO PROCESS A PARAMETER FIELD
41 ELSE
42 ERREXIT IF IFLAG IS NOT I ("n") :ERROR8:
43 CALL XIDAT TO PROCESS DATA LIST
44 ENDIF
45 EXIT XINPT
46 EXIT TO :RETURN:
47 :ERROR2: CALL XMSG "INVALID SYNTAX"
48 :ERROR8: CALL XMSG "MUST USE PARAMETER NAME TO RIGHT OF & OR =
49 :RETURN:
50 END XINPT
FORTAN CALLING PROCEDURE
CALL XIPAR

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

********
INPUT

COMMON XE - COMBUF, COMPTR, TOKENS

COMMON XE - DFLAG, IARG, IAREA, ICLASS,
LITDSP, SFLAG, MKBLNG, WBUF

********
OUTPUT

COMMON XE - IRETC, LITDWN, WBUF

********
NOTES

USES ROUTINES
XRMCS
XRMSS
XRSFI
XRES
* XIPAR PROCESSES A USER SPECIFIED PARAMETER FIELD

BEGIN XIPAR

IF A NAME IS SPECIFIED, THEN
INCREASE TO NEXT TOKEN
IF TOKEN IS '"", THEN
PROCESS I AND J SUBSCRIPTS
ERROR IF INVALID SUBSCRIPTING :ERR14:
IF DOUBLY SUBSCRIPTED, THEN
SET S-FLAG IN ARGNO'S SPECS. FIELD
SET LITOSP IN ARGNO'S SPECS TO NEXT LITERAL AREA SPACE (LITDOWN)
PUT ISUB AND JSUB INTO LITERAL AREA AT THIS SPOT
ELSE
SET LITOSP IN ARGNO'S SPECS TO ISUB
ENDIF
ELSE
SET LITOSP IN ARGNO'S SPECS TO 0
ENDIF
ERROR IF ORDE HAS MORE THAN 4 CHARACTERS :ERR18:
ERROR IF EXTRAN"OUS FIELD INPUT :ERRO2:
TURN OF D-FLAG (SAME LITERAL DATA) IN ARGNO'S SPECS.
SET PARAMETER NAME INTO ARGUMENT'S CHARACTERISTICS
SET COMPLETE (AND S) FLAG IN ARGUMENT'S CHARACTERISTICS
IF ALL ARGUMENTS ARE COMPLETE, THEN
SET INTERFACE TABLE COMPLETE FLAG
ENDIF
ELSE:" MUST BE A & INPUT
"ERROR IF NOT AN AMPERSAND (&) INPUT :ERRO2:
CLEAR PARAMETER NAME IN ARGUMENT'S CHARACTERISTICS
SET ARGUMENT AND INTERFACE TABLE INCOMPLETE
ENDIF
EXIT TO :RETURN:

ERROR2: CALL XMSG - "INVALID SYNTAX"
ERROR14: CALL XMSG - "INVALID SUBSCRIPT SYNTAX"
ERROR18: CALL XMSG - "INVALID ORDE NAME"

RETURN:
END XIPAR
FORTRAN CALLING PROCEDURE
CALL XILSD

INPUT
COMMON IE - .I
COMMON XE - .I, IFLAG, IARG, IDIN,
AVIS, IFLAG, IIUB, ITYPE,
LENYY, LITDYP, LITSEF, HARK,
HAXBTH, HOBTH, SFLAG, WBUF

INTERNAL VARIABLES
COMMON XS - BUFFER = LINE TO BE OUTPUT
BUFFTPR = INDEX INTO BUFFER FOR NEXT ASCII DATA
KETRNM = RETURN INDICATOR FOR INTERNAL ROUTINES

NOTES
USES Routines
EXEC
XRETA
XREI
XRM
XINX
XINX
XINX
XINX
XINX
XINX
XINX

© 1965 Xerox Corp. ORIG. PAGE IS POOR
833 9 * XILDX WILL LIST THE DATA ASSOCIATED WITH ONE ARGUMENT
834 1 * IS RETURNED AS A PROMPT.
835 2 BEGIN XILDX
836 3 SET ARGUMENT NAME INTO BUFFER
837 4 USE I-O-FLAG TO DETERMINE WHICH OF 'S', 'N', OR 'A'
838 5 WILL GO INTO THE PRINT BUFFER
839 6 IF B-FLAG IS OFF INDICATING NO LITERAL DATA, THEN
840 7 IF A PARAMETER NAME IS SPECIFIED, THEN
841 8 PUT PARAMETER NAME INTO BUFFER
842 9 IF S-FLAG IS ON INDICATING TWO SUBSCRIPTS, THEN
843 10 COMPUTE AND CONVERT TO CHARACTER FORMAT EACH SUBSCRIPT
844 11 PUT SUBSCRIPT INTO BUFFER
845 12 ELSE
846 13 IF LITDSP OF ARGUMENT IS > 0, THEN
847 14 COMPUTE AND CONVERT THIS SUBSCRIPT
848 15 PUT SUBSCRIPT INTO BUFFER
849 16 ENDF
850 17 ENDF
851 18 WRITE OUT THE PRINT BUFFER BUILT
852 19 ENDF
853 20 ELSE
854 21 LOCATE LITERAL LIST AND MASK
855 22 IF SYMBOLIC STRING, THEN
856 23 CALL XILDX TO PRINT SYMBOLIC STRING
857 24 ELSE
858 25 DO UNTIL ALL ELEMENTS PROCESSED
859 26 DO UNTIL A BUFFER OF DATA HAS BEEN GENERATED, OR
860 27 UNTIL ALL ELEMENTS PROCESSED
861 28 COMPUTE AND CONVERT THE SUBSCRIPT
862 29 IF MASK FOR ELEMENT INDICATES NO DATA, THEN
863 30 PUT "-*" INTO BUFFER
864 31 ELSE
865 32 CONVERT THE DATA USING XR06, XR14, OR XR16
866 33 PUT DATA AND "-*" INTO BUFFER
867 34 ENDF
868 35 ENDDO
869 36 IF ALL ELEMENTS OF THIS ARGUMENT HAVE BEEN PROCESSED, THEN
870 37 REMOVE THE TRAILING COMMA IN THE PRINT BUFFER
871 38 ENDF
872 39 WRITE OUT THE PRINT BUFFER BUILT
873 40 ENDDO
874 41 ENDF
875 2 ENDIF
876 1 END XILDX
FORTRAN CALLING PROCEDURE

CALL XILSS

C******

XILSS IS CALLED BY XILSS TO LIST SYMBOLIC STRING DATA

C******

INPUT

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

OUTPUT

COMMON XS - BUFFER, BUSPTR, DATPTR

C******
<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;&gt; \text{CALL XR16}</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>REAL</td>
<td>7</td>
<td>-2 &gt;&gt; \text{CALL XR14}</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>DOUBLE</td>
<td>9</td>
<td>-3 &gt;&gt; \text{CALL XR18}</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>NAME</td>
<td>3</td>
<td>4 &gt;&gt; \text{USE 3 WORDS}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>--------</td>
<td>0</td>
<td>\text{ERROR (INVALID)}</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>--------</td>
<td>0</td>
<td>\text{ERROR (INVALID)}</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>CHAR. STR.</td>
<td>-5 &gt;&gt; \text{USE SIZE + 3 WORDS}</td>
<td>-1 &gt;&gt; \text{USE SIZE + 2 WORDS}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>--------</td>
<td>0</td>
<td>\text{ERROR (INVALID)}</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>--------</td>
<td>0</td>
<td>\text{ERROR (INVALID)}</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>/</td>
<td>1</td>
<td>/</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>&gt;</td>
<td>1</td>
<td>&gt;</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>&lt;</td>
<td>1</td>
<td>&lt;</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>#</td>
<td>1</td>
<td>#</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>&amp;</td>
<td>1</td>
<td>&amp;</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>^</td>
<td>1</td>
<td>^</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>(</td>
<td>1</td>
<td>(</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>)</td>
<td>1</td>
<td>)</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>------</td>
<td>0</td>
<td>\text{ERROR (INVALID)}</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>EMO SYM. STR.</td>
<td>-7 &gt;&gt; \text{END SYM. STR.}</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>\text{BACKLASH}</td>
<td>1</td>
<td>\text{BACKLASH}</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>\text{BACKLASH}</td>
<td>1</td>
<td>\text{BACKLASH}</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>=</td>
<td>1</td>
<td>=</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>\text{LEFT BRACKET}</td>
<td>1</td>
<td>\text{LEFT BRACKET}</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>\text{RIGHT BRACKET}</td>
<td>1</td>
<td>\text{RIGHT BRACKET}</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>\text{REPEAT}</td>
<td>4</td>
<td>\text{APPEND &quot;R&quot;}</td>
<td>2</td>
</tr>
<tr>
<td>31</td>
<td>\text{APPEND &quot;R&quot;}</td>
<td>1</td>
<td>\text{APPEND &quot;R&quot;}</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>\text{APPEND &quot;R&quot;}</td>
<td>1</td>
<td>\text{APPEND &quot;R&quot;}</td>
<td>1</td>
</tr>
</tbody>
</table>
BEGIN XLSS

MOVE A CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER

DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED

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

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

EXIT TO ERROR 2 IF FIELD = 0

IF SIZE < 0, THEN

SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING

ENDIF

IF TOKSZ < 0, THEN

SET TOKSZ TO SIZE + 2

ENDIF

IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN

CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE

CLEAR PRINT BUFFER TO BLANKS

ENDIF

IF FIELD > 0, THEN

MOVE FIELD INTO CURRENT PRINT BUFFER POSITION

ELSE

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

ONE: CALL XR16 WITH VALUE IN NEXT WORD OF SYMB. STRING

AND PUT RESULTS INTO PRINT BUFFER

TWO: CALL XR14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING

AND PUT RESULTS INTO PRINT BUFFER

THREE: CALL XR18 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 INTO PRINT BUFFER FOLLOWED BY AN "R"

SIX: CALL XR16 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

EXIT BY 1

CALL EXEC TO WRITE THE PRINT BUFFER TO INDICATED DEVICE

EXIT XLSS

ENDCASE

INCREMENT PRINT BUFFER INDEX BY SIZE
1015  CD0 FORTRAN CALLING PROCEDEUE
1016  CD0
1017  CD0 CALL XICHR
1018  CD0
1019  CD0
1020  CD0
1021  CD0
1022  CD0 PRINT THE CHARACTERISTICS OF AN ARGUMENT
1023  CD0
1024  CD0
1025  CD0
1026  CD0 INPUT
1027  CD0
1028  CD0
1029  CD0 COMMON XE - LU
1030  CD0 COMMON XB - ARGNO, IOSTR, ISIZE, ITYPE,
1031  CD0 LENEFF, NARG
1032  CD0
1033  CD0
1034  CD0
1035  CD0 NOTES
1036  CD0
1037  CD0 USES ROUTINES
1038  CD0
1039  CD0
1040  CD0 EXEC
1041  CD0
1042  CD0
1043  CD0
1044  CD0
1045  CD0
1046  CD0 WRITE ARGUMENT CHARACTERISTICS
1047  CD0 BEGIN XICHR
1048  CD0 BUILD PRINT BUFFER WITH ARGUMENT NAME, SUBSCRIPTS, I/O TYPE AND
1049  CD0 DATA TYPE
1050  CD0 WRITE OUT THE PRINT BUFFER
1051  CD0 END XICHR
FORTRAN CALLING PROCEDURE

CALL XIEXT

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

INPUT

COMMON XB - ARGNO, ISIZES, WKBUF

OUTPUT

COMMON XB - CFLAG, DFLAG, IARG, IARGA,
ICLASS, IDIM, IDFLAG, ISIZE,
ISUB, ITYPE, LENEFF, LIDSP,
LITSZ, NDIBTM, NOBTM, SFLAG

*********

*********

*********

*********

*********

*********

*********

*********

*********

*********

NOBTS

USES ROUTINES

IAMD XIEXT

*********

** EXTRACT THE VARIOUS VALUES AND FLAGS ASSOCIATED WITH THIS
** ARGUMENT

BEGIN XIEXT

USING THE ARGUMENT NO. (ARGNO), LOCATE THIS ARGUMENT'S CHARACTERISTICS
IN THE WORKING BUFFER

EXTRACT EACH OF THE FIELDS INTO A WORD OF COMMON FOR GENERAL USEAGE

END XIEXT
1077 1 CBO FORTRAN CALLING PROCEDURE
1078 1 CBO
1079 1 CBO
1080 1 CBO
1081 1 CBO
1082 1 CBO
1083 1 CBO
1084 1 CBO
1085 1 CBO
1086 1 CBO
1087 1 CBO
1088 1 CBO
1089 1 CBO
1090 1 CBO
1091 1 CBO
1092 1 CBO
1093 1 CBO
1094 1 CBO
1095 1 CBO
1096 1 CBO
1097 1 CBO
1098 1 CBO
1099 1 CBO
1100 1 CBO
1101 1 CBO
1102 1 CBO
1103 1 CBO
1104 1 CBO
1105 1 CBO
1106 1 CBO
1107 1 CBO
1108 1 CBO
1109 1 CBO
1110 1 CBO
1111 1 CBO
1112 1 CBO
1113 1 CBO
1114 1 CBO
1115 1 CBO
1116 1 CBO
1117 1 CBO
1118 1 CBO
1119 1 CBO
1120 1 CBO
1121 1 CBO
1122 1 CBO
1123 1 CBO
1124 1 CBO
1125 1 CBO
1126 1 CBO
1127 1 CBO
1128 1 CBO
1129 1 CBO
1130 1 CBO
1131 1 CBO
1132 1 CBO
1133 1 CBO
1134 1 CBO
1135 1 CBO
1136 1 CBO
1137 1 CBO
1138 1 CBO
1139 1 CBO
1140 1 CBO
1141 1 CBO
1142 1 CBO
1143 1 CBO
1144 1 CBO
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 (LENBUFF) OF LITERAL
1156 9 UP IN THE WORKING BUFFER
1157 7 ENDEF
1158 6 ENDDO
1159 5 ENDF
1160 5 CALCULATE NEW DISPLACEMENT AND SET IN LITDSP
1161 4 ENDF
1162 3 ENDDO
1163 2 ENDDO
1164 1 ENd XILIT
### SYMBOL DEFINITION TABLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR2</td>
<td>292</td>
</tr>
<tr>
<td>ERROR3</td>
<td>616</td>
</tr>
<tr>
<td>ERROR4</td>
<td>782</td>
</tr>
<tr>
<td>ERROR5</td>
<td>379</td>
</tr>
<tr>
<td>ERROR6</td>
<td>709</td>
</tr>
<tr>
<td>ERROR7</td>
<td>617</td>
</tr>
<tr>
<td>ERROR8</td>
<td>710</td>
</tr>
<tr>
<td>ERROR9</td>
<td>618</td>
</tr>
<tr>
<td>ERRORA</td>
<td>783</td>
</tr>
<tr>
<td>ERRRA</td>
<td>432</td>
</tr>
<tr>
<td>ERRRS</td>
<td>433</td>
</tr>
<tr>
<td>ERRRT</td>
<td>434</td>
</tr>
<tr>
<td>ERRRB</td>
<td>784</td>
</tr>
<tr>
<td>EXIT1</td>
<td>1004</td>
</tr>
<tr>
<td>EXIT2</td>
<td>685</td>
</tr>
<tr>
<td>FIVE1</td>
<td>1001</td>
</tr>
<tr>
<td>FOUR1</td>
<td>1000</td>
</tr>
<tr>
<td>LIST1</td>
<td>687</td>
</tr>
<tr>
<td>ZONE1</td>
<td>994</td>
</tr>
<tr>
<td>PROMPT</td>
<td>686</td>
</tr>
<tr>
<td>RETURN1</td>
<td>619</td>
</tr>
<tr>
<td>RETURN2</td>
<td>435</td>
</tr>
<tr>
<td>RETURN3</td>
<td>380</td>
</tr>
<tr>
<td>RETURN4</td>
<td>785</td>
</tr>
<tr>
<td>RETURN5</td>
<td>273</td>
</tr>
<tr>
<td>RETURN6</td>
<td>711</td>
</tr>
<tr>
<td>SIX1</td>
<td>1002</td>
</tr>
<tr>
<td>THREE1</td>
<td>998</td>
</tr>
<tr>
<td>ILOG1</td>
<td>996</td>
</tr>
<tr>
<td>XCHR1</td>
<td>1047</td>
</tr>
<tr>
<td>XIDAT1</td>
<td>502</td>
</tr>
<tr>
<td>XEST1</td>
<td>1091</td>
</tr>
<tr>
<td>XILIT1</td>
<td>1146</td>
</tr>
<tr>
<td>XILS1</td>
<td>835</td>
</tr>
<tr>
<td>XILSS</td>
<td>973</td>
</tr>
<tr>
<td>XILST</td>
<td>338</td>
</tr>
<tr>
<td>XINIX</td>
<td>113</td>
</tr>
<tr>
<td>XINPT</td>
<td>664</td>
</tr>
<tr>
<td>XINTE</td>
<td>35</td>
</tr>
<tr>
<td>XIPAR1</td>
<td>752</td>
</tr>
<tr>
<td>XIPMT</td>
<td>276</td>
</tr>
<tr>
<td>XIPRM</td>
<td>193</td>
</tr>
<tr>
<td>XISUB</td>
<td>415</td>
</tr>
</tbody>
</table>
CD************
CD
CD RTE RUN PROCEDURE FOR LIBRARY MAINTENANCE:
CD
CD  :RUN,XLMAN,LU,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,AMPAR
CD
CD FDS FUNCTIONs AND ROUTINES USED:
CD
CD XELDS, XLDBF, XDEL, XLMNT, XLMOD, XLMSG,
CD XLRAM, XLPK, XTCOM, XRMOD, XRMG
CD
CD XE COMMON USED:
CD
CD EQUIVALENCE (XE(1)), (LU ),
CD +XE(2), ICCLASS), (XE(3), ISECU ),
CD +XE(4), FLAGS ), (XE(5), MA$5T ),
CD +XE(6), SUBSTA ), (XE(7), NUMBER ),
CD +XE(8), VALFLG), (XE(9), TOKENS ),
CD +XE(142), ICR ), (XE(145),COMBUF)
CD************
1 BEGIN XLIMAN
2 CALL RMPAR TO GET INPUT PARAMETERS
3 INITIALIZE COMMON TO ZERO
4 SET CLASS NUMBER TO ZERO
5 CALL EXEC TO GET A CLASS NUMBER
6 EXIT XLIMAN IF SECURITY CODE IS NOT VALID
7 CALL XMOV TO INITIALIZE TOKENS IN COMMON
8 DO FOREVER
9 :PROMPT:
10 :INITIALIZE MASTER AND SUBSTATE FLAGS
11 CALL XTCCM TO PROMPT FOR OPTION
12 IF XTom RETURN CODE IS NOT ZERO OR FIRST TOKEN IS NOT AN INTEGER OR INTEGER > 7 THEN
13 CALL XMSG TO WRITE INVALID RESPONSE
14 GO TO :PROMPT:
15 ENUF:
16 CASE INTEGER (:XLPRM;:XLPRM;:XLPRM;:XLPRM;:XLDEL;:XLMOD;
17 :XLMSG;:XLDBF;:XLDBF;:EXIT):
18 :XLPRM:
19 :SET NUMBER TO INTEGER
20 CALL XELS TO LOAD XLPRM TO CREATE SYSTEM PROMPT FILE
21 :XLPRO:
22 :SET VALFLG TO SAY ORIGINAL XLPRO REQUEST
23 CALL XELS TO LOAD XLPRO TO ADD A PROCESSOR
24 DO UNTIL VALFLG SAYS EXIT (X)
25 CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
26 CALL XELS TO LOAD XLPRO TO COMPLETE PROCESSING
27 ENDDO
28 :XLDEL:
29 CALL XELS TO LOAD XLDEL TO DELETE A PROCESSOR
30 :XLMOD:
31 :SET VALFLG TO SAY OPTIMAL XLMOD REQUEST
32 CALL XELS TO LOAD XLMOD TO MODIFY A PROCESSOR
33 DO UNTIL VALFLG SAYS EXIT (X)
34 CALL XELS TO LOAD XLINT TO ENTER DEFAULT VALUES
35 CALL XELS TO LOAD XLMOD TO COMPLETE PROCESSING
36 ENDDO
37 :XLMSG:
38 CALL XELS TO LOAD XLMSG TO ADD A MESSAGE
39 :XLDBF:
40 CALL XELS TO LOAD XLDBF TO HANDLE DATA BASE FILES
41 ENUF:
42 EXIT:
43 CALL EXEC TO RELEASE CLASS NUMBER
44 END XLIMAN
**FORTRAN CALLING PROCEDURE:**

```fortran
CALL XELOD('XLPRM')
```

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

```fortran
INPUT FROM COMMON:
```

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

**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 XREMSG**
- **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** - (INTEGER) PROM2, PROM3, PROM4 ARE ALL USER PROMPT ARRAYS

**FUNCTIONS AND ROUTINES USED:**

- **XREMOV, XREMSG, XTCOM**

**COMMON USED:**

- **ERIVALENCE**
  - `+(XE(3), ISECU,(XE(7), NUMBR),`
160 1 BEGIN XLPRM
161 2 :PRM1:
162 3 CALL XCOM TO PROMPT USER FOR 0 DIRECTIVES
163 4 ERREXIT IF RESPONSE NOT INTEGER 1-63 TO :ERR1:
164 5 CALL CREAT TO CREATE PROMPT FILE
165 6 IF FILE ALREADY EXISTS THEN
166 7 CALL XCOM TO PROMPT USER FOR SCRATCH/CANCEL
167 8 EXIT XLPRM IF RESPONSE IS CANCEL
168 9 CALL PURGE TO PURGE PROMPT FILE
170 5 ERREXIT IF PURGE ERROR TO :FILERR:
171 6 GO TO :PRM1:
177 3 ELSE (CREATE NEW FILE)
172 4 ERREXIT IF CREATION ERROR TO :FILERR:
173 5 CALL XCOM TO PROMPT USER FOR LIST OF DIRECTIVES
174 6 ERREXIT IF LIST IS INCONSISTENT WITH 0 OF DIRECTIVES TO :ERR1:
175 7 DO FOR EACH DIRECTIVE
176 8 ERREXIT IF RESPONSE IS NOT VALID DIRECTIVE TO :ERR1:
177 9 CALL XMOV TO MOVE DIRECTIVE INTO BUFFER
178 END0
179 5 CALL WRITF TO WRITE LIST OF DIRECTIVES TO FILE
180 6 ERREXIT IF WRITF ERROR TO :FILERR:
181 7 DO FOR EACH DIRECTIVE
182 8 CALL XCOM TO PROMPT USER FOR DEFINITION
183 9 DO UNTIL EOS IS REACHED IN RESPONSE
184 5 ERREXIT IF RESPONSE IS NOT A CHARACTER STRING TO :ERR1:
185 6 ERREXIT IF RESPONSE IS TOO LONG (>128) TO :ERR1:
186 7 MOVE RESPONSE INTO BUFFER
188 9 SET CONTROL CHARACTERS IN BUFFER
189 5 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
190 END0
191 4 SET REMAINDER OF BUFFER TO NULL
192 4 CALL WRITF TO WRITE DEFINITION
193 5 ERREXIT IF WRITF ERROR TO :FILERR:
194 END0
195 4 CALL CLOSE TO CLOSE FILE
196 5 ERREXIT IF CLOSE ERROR TO *FILERR:
197 6 CALL XMSG TO DISPLAY FILE CREATED MESSAGE
198 2 ENDF
199 1 EXIT XLPRM
200 2:ERR1: (ERROR IN RESPONSE)
201 5 CALL XMSG TO DISPLAY ERROR IN RESPONSE
202 6 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
202 2:FILEERR: (FILE ACCESS ERROR)
203 5 CALL XMSG TO DISPLAY ERROR
204 6 CALL CLOSE TO CLOSE FILE
205 6 CALL PURGE TO PURGE FILE
206 1 END XLPRM
BEGIN XLPRO

IF THIS ENTRY IS THE ORIGINAL XLPRO ENTRY FOR THIS PROCESSOR THEN

INITIALIZE MASTER AND SUB STATES

PERFORM LED TO UPDATE LIBRARY DIRECTORY

IF PROCESSOR HAS AN INTERFACE TABLE THEN

SET RETN = 2

:PRMPT2:

CALL XTCOM TO PROMPT FOR 8 PARAMETERS

ERREXIT IF XTCOM RETURN CODE IS NOT ZERO OR

ERREXIT IF 8 PARAMETERS NOT INTEGER 1-63 TO :PRMERR:

CREATE HEADEP ENTRY WITH 8 PARAMETERS AND PROCESSOR NAME

CALL XLINS TO WRITE INSTRUCTIONS FOR ENTERING SPECS

DO FOR 8 PARAMETERS

CALL XLSPS TO CREATE ONE PARAMETER ENTRY

ENDDO

SET CODES ARRAY TO ADD ABSTRACT AND PARAMETER DEFINITIONS

CALL XLIFL TO CREATE PROMPT FILE

SET RETN = 3

:PRMPT3:

CALL XTCOM FOR DEFAULT VALUES DECISION

ERREXIT IF RETURN CODE IS NOT ZERO OR

ERREXIT IF RESPONSE IS NOT YE OR NO TO :PRMERR:

IF RESPONSE WAS YE THEN

SET FLAG TO CALL INTERFACE TABLE EDITOR

EXIT XLPRO

ENDIF

ELSE

SET CODES ARRAY TO ADD ONLY ABSTRACT

CALL XLIFL TO CREATE PROMPT FILE

PERFORM XLPRO - NO RETURN EXPECTED

ENDIF

END

:PRMERR:

CALL XRMGR TO DISPLAY ERROR MESSAGE

GO TO (:PRMPT1:, :PRMPT2:, :PRMPT3:), RETN

END XLPRO
1 BEGIN LIBD
2 CALL OPEN TO OPEN LIBRARY DIRECTORY
3 IF RETURN CODE SAYS FILE NOT FOUND THEN
4 SET RECORD 1 TO ALL ZEROS
5 ELSE
6 ERREXIT IF FILE ERROR TO :FILERR:
7 CALL READF AND CLOSE TO READ IN LIBRARY DIRECTORY
8 ERREXIT IF FILE ERROR TO :FILERR:
9 IF # PROCESSORS + 1 > 50 THEN
10 CALL XRMSE TO WRITE ERROR: TOO MANY PROCESSORS
11 EXIT XLPRO
12 ENDIF
13 SET RETN = 1
14 :PRMT1:
15 CALL XTCOM TO PROMPT FOR PROCESSOR NAME, VERSION, INT TABLE
16 EXIT XLPRO IF RETURN CODE SAYS X ENTERED
17 ERREXIT IF RETURN CODE > ZERO OR
18 ERREXIT IF PROCESSOR NAME IS NOT 6-CHAR NAME TO :PRMERR:
19 CALL XRMOV TO MOVE PROCESSOR NAME INTO ENTRY
20 ERREXIT IF VERSION IS NOT INTEGER VALUE 0-127 TO :PRMERR:
21 CALL XRSET TO SET VERSION IN ENTRY
22 ERREXIT IF INTERFACE TABLE OPTION IS NOT YE OR NO TO :PRMERR:
23 SET IT BIT = 0
24 IF RESPONSE IS YES THEN
25 SET IT BIT = 1
26 ENDIF
27 CALL XRSET TO SET BIT ON/OFF
28 ERREXIT IF PROCESSOR NAME ALREADY EXISTS TO :PRMERR:
29 INCREMENT # PROCESSORS BY 1
30 CALL XRMOV TO MOVE NEW ENTRY INTO XLIBD
31 IF # PROCESSORS > 1 THEN
32 CALL PURGE TO PURGE OLD FILE
33 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
34 ELDIF
35 CALL CREATE, WRTF AND CLOSE TO CREATE NEW LIBRARY DIRECTORY
36 ERREXIT IF FILE ERROR TO :FILERR:
37 ELDIF
38 1 END LIBD
410 1 BEGIN XCBDB
411 2 CALL OPEN TO OPEN OLD FIL
412 3 ERREXIT IF OPEN ERROR TO :ERR3:
413 4 CALL READ TO READ IN HEADER RECORD
414 5 ERREXIT IF READ ERROR TO :ERR2:
415 6 COMPUTE NUMBER OF TOC RECORDS (NTOC) AS(# ENTRIES + 16)/16
416 7 IF NTOC > 1 THEN
417 8 CALL READ TO READ IN REMAINING TOC RECORDS
418 9 ERREXIT IF READ ERROR TO :ERR2:
419 10 ENDF
420 11 SET # RECORDS LEFT TO READ (SIZE) AS TOTAL SIZE - NTOC
421 12 CALL CREATE TO CREATE NEW FIL
422 13 ERREXIT IF CREATE ERROR TO :ERR2:
423 14 DO FOR EACH TOC ENTRY
424 15 IF REQUEST WAS PDF TO PDF THEN
425 16 ERREXIT IF NAME > 4 CHAR OR
426 17 ERR IF CLASS IS DNC AND NAME > 2 CHAR TO :ERR1:
427 18 APPEND PDF CONV. TO FRONT OF NAME
428 19 ELSE
429 20 REMOVE PDF CONV. FROM NAME
430 21 ENDF
431 22 ENDDO
432 23 CALL WRITE TO WRITE NEW TOC RECORDS TO NEW FIL
433 24 ERREXIT IF WRITE ERROR TO :ERR1:
434 25 SET TOCPTR TO FIRST TOC ENTRY
435 26 SET FIRST RECORD NUMBER AND LAST RECORD NUMBER
436 27 CONVERT SIZE FROM BLOCKS TO WORDS
437 28 DO UNTIL ALL RECORDS ARE CYPED (SIZE = 0)
438 29 SET LENGTH TO MAXIMUM SIZE OF 1 READ/WRITE (1024)
439 30 IF SIZE < LENGTH THEN
440 31 SET LENGTH TO SIZE
441 32 ENDF
442 33 CALL READ TO READ LENGTH WORDS OF DATA
443 34 ERREXIT IF READ ERROR TO :ERR1:
444 35 DECREMENT SIZE BY LENGTH READ
445 36 UPDATE FIRST AND LAST RECORD NUMBERS
446 37 START SEARCH FOR TOCPTR = TOCPTR TO LAST TOC ENTRY
447 38 EXIT IF RECORD # IN TOC ENTRY > LAST RECORD NUMBER
448 39 IF CLASS IS AN INTERFACE TABLE THEN
450 40 IF RECORD IN TOC ENTRY = INTERFACE INDEX IN TOC ENTRY
451 41 AM ALL INDEX IN TOC ENTRY
452 42 SET NAME IN INTERFACE TABLE TO NAME IN TOC ENTRY
453 43 ENDF
454 44 ENDSARCH
455 45 CALL WRITE TO WRITE LENGTH WORDS OF DATA TO NEW FIL
456 46 ERREXIT IF WRITE ERROR TO :ERR1:
457 47 ENDDO
CALL CLOSE TO CLOSE NEWFIL
CALL CLOSE TO CLOSE OLDFIL
CALL XDDDB TO DELETE OLDFIL FROM PDB LOG FILE
CALL PURGE TO PURGE OLDFIL FROM SYSTEM
1 EXIT XLCDB

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

:ERR2:
CALL CLOSE TO CLOSE OLDFIL

:ERR3:
IF REQUEST WAS PDB TO M66 THEN
SET QUAL TO SAY DELETE M66 FILE
ELSE ( REQUEST WAS M66 TO PDB )
SET QUAL TO SAY DELETE PDB FILE
ENDIF
CALL XDDDB TO DELETE M66/PDB FROM LOG FILE
IF ERROR WAS FILE MANAGER THEN
CALL XRMST TO DISPLAY ERROR AND RETURN CODE
ELSE
CALL XRMST TO DISPLAY ERROR
ENDIF
1 END XLCDB
FORTRAN CALLING SEQUENCE:

CALL XLD0F ('XLD0F')

XLD0F IS THE DATA BASE FILES MAIN ROUTINE. IT DETERMINES WHICH
DATA BASE ROUTINE IS NEEDED AND GATHERS INPUTS NEEDED BY THAT
ROUTINE. HANDLES ALL PDB LOG FILE RELATED TRANSACTIONS.

INPUTS IN COMMON:

XE(3) ISECU, XE(7) NUMBR,
XE(65) TOKENS, XE(142) ICR

INTERNAL VARIABLES IN COMMON

XE(3) QUAL - USER QUALIFIER REQUIRED BY XDB ROUTINES AND XRFWM
XE(6) SECU - TEMPORARY SLOT FOR ISECU DURING THIS OVLAY
X(5) OLDFILE - OLD FILE NAME
X(6) NEWFIL - NEW FILE NAME
X(9) TOTSLZ - SIZE OF OLDFILE/NEWFIL IN BLOCKS
X(10) FILCHR - 4 BASE CHARACTERS OF FILE NAME
X(12) QUALIF - QUALIFIER ENTERED BY USER

RTE ROUTINES USED:

KCVT, OPENW

FDS ROUTINES USED:

XDDRA, XDDRY, XERW, XLD0B, XLPCR

COMMON USERS:

(EQUIVALENCE (XE(3), ISECU),
+XE(3), QUAL ),
+XE(6), SECU ),
+XE(65), TOKENS),
+XE(142), ICR ),
+XE(145), COMMDF),
+XE(6), NEWFIL),
+XE(9), TOTSLZ),
+XE(10), FILCHR),
+XE(12), QUALIF),
+XE(100), IECB)
1 BEGIN XLDBF
2   IF REQUEST IS TO CREATE/MODIFY LOG FILE THEN
3     CALL XPDR TO OPEN MDB/PDB LOG FILE
4   ELSE
5     ERREXIT IF OPEN ERROR TO :FILE ERR:
6     CALL XPMD TO MODIFY MDB/PDB LOG FILE
7     ENDIF
8   ELSE
9     DO UNTIL USER REQUESTS EXIT (X)
10    CALL XTOM TO PROMPT USER FOR NAME AND USER ID
11    IF RESPONSE IS NOT EXIT (PERCENT) THEN
12       ERREXIT IF RESPONSE IS INVALID TO :ERR1:
13       SAVE A CHARACTER NAME AND ID IN COMMON
14       IF REQUEST WAS PDB TO MDB THEN
15         SET QUALIFIER TO SEARCH FOR PDB NAME
16       ELSE (REQUEST WAS FOR MDB TO PDB)
17         SET QUALIFIER TO SEARCH FOR MDB NAME
18       ENDIF
19       CALL XDOBV TO VERIFY EXISTENCE OF MDB/PDB DEPENDING ON QUALIFIER
20       ERREXIT IF NAME WAS NOT FOUND TO :ERR1:
21       ERREXIT IF FILE MANAGER ERROR TO :FILE ERR:
22       IF REQUEST WAS PDB TO MDB THEN
23         SET QUALIFIER TO ADD MDB TO LOG FILE
24       ELSE (REQUEST WAS MDB TO PDB)
25         SET QUALIFIER TO ADD PDB TO LOG FILE
26       ENDIF
27       CALL XDOBA TO ADD MDB/PDB NAME TO LOG FILE DEPENDING ON QUALIFIER
28       ERREXIT IF DUPLICATE NAME OR FILE MANAGER ERROR TO :FILE ERR:
29       IF REQUEST WAS PDB TO MDB THEN
30         CALL XOFM TO SET OLD FIL TO PDB NAME
31         SET NEWFIL TO MBD NAME
32       ELSE (REQUEST WAS MBD TO PDB)
33         SET OLD FIL TO MBD NAME
34         SET NEWFIL TO PDB NAME
35       ENDIF
36       CALL XLCDB TO COPY OLD FIL TO NEWFIL
37     ENDIF
38   ENDIF
39  END XLDBF
40 1 EXIT XLDBF
41
42 2 :ERR1:
43 2 CALL XRMSG TO DISPLAY ERROR
44 3 RETURN TO PROMPT FOR ANOTHER 4 CHARACTERS AND USER ID
45 2 :FILE ERR:
46 2 CALL XRMSG TO DISPLAY FILE ACCESS ERROR
47 1 END XLDBF
CD**********
CD FORTRAN CALLING PROCEDURE FOR DELETE PROCESSOR
CD CALL XELD'S ('XLDEL')
CD**********
CD XLDEL DELETES A PROCESSOR FROM THE LIBRARY DIRECTORY AND
CD THE PROMPT FILE. IF THE PROCESSOR HAS AN INTERFACE TABLE,
CD IT DELETES THE DEFAULT INTERFACE TABLE FILE ALSO.
CD**********
CD INTERNAL VARIABLES:
CD COMM - (INTEGER, 7 WORDS) IS A TEMPORARY WORK AREA
CD WHERE ENTRY IN LIBRARY DIRECTORY IS STRIPPED
CD DOWN TO PROCESSOR NAME
CD PRNAM - (INTEGER, 2 WORDS) PROCESSOR NAME TO BE
CD DELETED.
CD**********
CD RTE/ FMGR ROUTINES USED:
CD IAND, XCVT, CREAT, OPEN, READF, WRITF, CLOSE, PURGE
CD FDS ROUTINES USED:
CD XRCP, XNEXT, XRMOV, XRMSG, XRPK, XRSET, XRUPK, XTCOM
CD XE AND XB COMMON USED
CD EQUIVALENCE (XE(3), ISECU),
CD + (XE(142), ICR), (XE(145), COMBUF),
CD + (XH(48), LIBD1), (XH(51), LIBD2)
CD**********
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:
XEXT, XMSG

RTE ROUTINES USED:
CLOSE, CREAT, WRITF

COMMON USED:

EQUIVALENCE (X(2), ISECU),
+ (X(142), ICR ), (X(290), LITPR),
+ (X(91), LITLEN),
+ (X(96), MOPARM), (X(101), MDPR ),
+ (X(108), PARMS ), (X(1), IERR ),
+ (X(2), IDCB )
1 BEGIN XLIPL
2 EXTRACT LITERAL LENGTH FROM HEADER
3 COMPUTE # BLOCKS FOR THIS FILE
4 CALL CREAT TO CREATE DEFAULT INT TABLE FILE
5 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
6 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
7 IF THERE IS A LITERAL RECORD THEN
8 CALL WRITF TO WRITE LITERALS
9 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
10 ENDIF
11 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
12 CALL CLOSE
13 ERREXIT IF RETURN CODE < ZERO TO :FILERR:
14 CALL XMSG TO DISPLAY FILE (NAME) CREATED MESSAGE
15 EXIT XLIPL
16 :FILERR:
17 CALL XMSG TO WRITE FILE ACCESS ERROR
18 CALL XEXIT TO RETURN TO MAIN PROGRAM
19 END XLIPL
1 CD**********
2 CD0        FCRTAN CALLING SEQUENCE:
3 CD0        CALL XLNS
4 CD0
5 CD**********
6 CD1        XLNS DISPLAYS INSTRUCTIONS FOR ENTERING PARAMETER SPECS
7 CD1
8 CD1
9 CD**********
10 CD4        INTERNAL VARIABLES:
11 CD4        PROMPT - (INTEGER, 120 WORDS) ALL 7 LINES IN AN ARRAY
12 CD4        TO BE DISPLAYED
13 CD4
14 CD4
15 CD**********
16 CD5        RTE ROUTINES USED:
17 CD5        RTE ROUTINES USED:
18 CD5        EXEC
19 CD5        COMMON USED:
20 CD5        EQUIVALENCE (REC(1), LU )
21 CD5
22 CD**********
23 #
24 #
25 #
26 Begin XLNS
27 2 CALL EXEC TO DISPLAY ALL 7 LINES
28 End XLNS
FORTRAN CALLING PROCEDURE:

CALL XELDS (XLINT)

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

FDS FUNCTIONS AND SUBROUTINES USED:

XEINT, XENTW, XINIX, XRMOV

COMMON USED:

+EQUIVALENCE (RECI, MASSTA),
+RECI(6), SUBSTA, (RECI(7), NUMDIR),
+RECI(8), DIREC*, (RECI(18), LTFLG),
+RECI(37), ARCHO, (RECI(47), PRNMP),
+RECI(73), SIZES*, (RECI(83), LITDOWN),
+RECI(90), LITPR, (RECI(91), LITLEN),
+RECI(92), HAP*,
+RECI(96), NUMARG, (RECI(97), HEITAB),
+RECI(100), WKBNG, (RECI(101), WKBUF),
+RECI(1400), EMN*
7881 BEGIN XINT
7892 SET MASTER STATE AS EXEC
7902 SET SUBSTATE AS INTERFACE TABLE EDITOR
7912 SET UP LIST OF VALID DIRECTIVES
7922 SET UP LIST FLAG TO GET ENTIRE LIST
7932 SET CURRENT ARGUMENT TO ZERO
7942 SET PROMPT MODE TO ALL
7952 SET UP INDEX INTO PARS OF SHORT PROMPTS
7962 SET NEW TABLE NAME TO ZERO
7972 INITIALIZE ARRAY OF TYPE LENGTHS
7982 IF LITERAL LENGTH IS ZERO THEN
7993 SET LITERAL POINTER TO FIRST AVAILABLE WORD
8002 ELSE
8012 SET LITERAL POINTER TO FIRST WORD OF LITERALS
8022 CALL XEINT TO UNPACK LITERALS
8032 ENDIF
8042 CALL XNIX TO ACCEPT DEFAULT VALUES
8052 CALL ZERINT TO RETURN
8061 END XINT

3-141
FORTRAN CALLING SEQUENCE:

CALL XELBS ('XLMDJ')

XLMDJ 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, POIN1, READ, WRIT

FDS ROUTINES USED:

XLIIF, XLINS, XLISP, XREPD, XREXT,
XRN0V, XRN0G, XRPED, XRFET, XRCM

COMMON USED:

EQUIVALENCE (XCE3), (SECU),

* (XGE5), (HASTA), (XE(6)), (SUBSTA),

* (XGE(7)), (VAL0), (XE(135)), (每年),

* (XGE(85)), (PROM), (XGE(156)), (PROM),

* (XGE(86)), (INTOT), (XCE(249)), (HANG),

* (XGE(242)), (ICh),

* (XCE(240)), (COMNAT), (XGE(260)), (VRS),

* (XGE(35)), (XFL-()), (XGE(35)), (CFL),

* (XGE(37)), (SPFL), (XGE(38)), (IT),

* (XGE(38)), (JEN), (XGE(44)), (JENFL)

* (XIB(33)), (SUBFLAG), (XIB(35)), (CFL),

* (XIB(36)), (SUBF), (XIB(38)), (IT),

* (XIB(39)), (JEN), (XIB(44)), (JENFL)

* (XIB(88)), (LITBSP), (XIB(90)), (LETB)

* (XIB(92)), (IMOD), (XIB(96)), (HOPAN1),

* (XIB(100)), (REDR), (XIB(100)), (RED1)

* (XIB(104)), (LIB02), (XIB(104)), (PARTS)

* (XIB(104)), (LIB02), (XIB(104)), (PARTS)
BEGIN XLMOD
   IF VALFLAG SAYS THIS IS AN ORIGINAL REQUEST TO MODIFY THEN
      SET RTN = 1
   ENDIF

   CALL XLMOD TO DISPLAY SPEC INSTRUCTIONS
   INITIALIZE MASTER AND SUB STATE FLAGS
   CALL XTCOP TO PROMPT FOR PROCESSOR NAME
   EXIT XLMOD IF RETURN CODE SAYS X ENTERED
   ERREXIT IF XCOM 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 NEST AND SPEC
      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 PARSER TABLE NAME
   CALL OPEN, READF AND CLOSE TO READ IN SHORT PROMPTS
   ERREXIT IF THERE WAS A FILE ERROR TO :FILERR:
   CALL NAMF TO RENAME PROMPT FILE >XLMAP
   ERREXIT IF NAMF ERROR TO :FILERR:
   SET CODES ARRAY TO MODIFY/ABSTRACT AND NO CHANGES TO PARAMETER SPEC
   PERFORM DELPAR TO DELETE PARAMETERS
   PERFORM MODPAR TO MODIFY PARAMETERS
   PERFORM ADDPAR TO ADD PARAMETERS
   CALL XLPFL TO CREATE NEW PROMPT FILE
   PERFORM DEFAULT TO ADD/MODIFY/DELETE ANY DEFAULT VALUES
   ELSE
      CALL NAMF TO RENAME PROMPT FILE >XLMAP
      ERREXIT IF NAMF ERROR TO :FILERR:
      SET CODES ARRAY TO MODIFY ABSTRACT ONLY
      CALL XLPFL TO CREATE NEW PROMPT FILE
      PERFORM XLMOD - NO RETURN EXPECTED
   ENDIF
   CALL PURGE TO PURGE OLD DEFAULT INTERFACE TABLE FILE
   ERREXIT IF FILE ERROR TO :FILER:
   CALL XLMOD TO CREATE NEW DEFAULT INTERFACE TABLE FILE
   SET VALFILE TO SAY ORIGINAL REQUEST TO MODIFY
   PERFORM XLMOD - NO RETURN EXPECTED
   :PRMERR:
   CALL XRMOD TO DISPLAY ERROR MESSAGE
   GO TO (:PRM1, :PRM2, :PRM3, :PRM4, :PRM5, :PRM6, :PRM7), RTN
   :FILERR:
   CALL XRMOD TO DISPLAY FILE ERROR
END XLMOD
BEGIN VERSION
SET RTN = 2
CALL XCOM TO PROMPT USER FOR VERSION NUMBER
IF RETURN CODE IS NOT CR THEN
ERROR IF RETURN CODE IS NON-ZERO ON
ERROR IF VERSION IS INVALID TO :PRMR:
IF INPUT VERSION IS NOT EQUAL TO OLD VERSION THEN
CALL XRSET TO PUT NEW VERSION IN ENTRY
CALL OPEN, WRITF, CLOSE TO UPDATE LIBRARY DIRECTORY
ERROR IF FILE ERROR TO :FILERR:
ENDIF
ENDIF
END VERSION
BEGIN DLP RM
SET RTN = 3
:PRM3:
DO UNTIL RETURN CODE IS CR ENTERED
IF NUMBER OF PARAMETERS = 1 THEN
CALL XCOM TO PROMPT FOR DELETE PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPRD TO INTERPRET RESPONSE
PERFORM CHAT TO CHECK FOR EXISTING DATA
SET ARCM TH WORD IN CODES TO SAY 'DELETED'
CALL XREMOV TO MOVE DATA TO DELETE PARAMETER
DECREMENT # PARAMETERS BY 1
ENDIF
ELSE
CALL XRMSG TO DISPLAY NO PARAMETERS CAN BE DELETED
EXIT DLP RM
ENDIF
END DLP RM
END MODPRM
BEGIN MODPRM
SET RTN = 4
:PRM4:
DO UNTIL RETURN CODE IS CR ENTERED
CALL XCOM TO PROMPT FOR MODIFY PARAMETER NAME
IF RETURN CODE IS NOT CR ENTERED THEN
PERFORM RSPRD TO INTERPRET RESPONSE
PERFORM CHAT TO CHECK FOR EXISTING DATA
SET ARCM TH NON-DELETED WORD IN CODES TO SAY 'MODIFIED'
CALL VLMP TO PROMPT USER FOR SPECIFICATIONS
SET IT COMPLETE BIT OFF
ENDIF
ENDIF
END MODPRM
CD**********
1 CD0 FORTRAN CALLING PROCEDURE
1 CD0 CALL XMSG ('XMLSG ')
1 CD0 CALL XHELDS ('XMLSG ')
1 CD0**********
1 CD1 XMLSG PROVIDES MAINTENANCE OF THE FDS MESSAGE FILE XRMSE
1 CD1**********
1 CD2 INPUT
1 CD2 XE COMMON - LU, ISEQU, FLAGS, TOKENS, ICR
1 CD2**********
1 CD3 TERMINAL - CREATING MODE, AREA AND MAXIMUM NUMBER OF MESSAGES
1 CD3 UPDATE MODE, MESSAGE NUMBER AND TEXT
1 CD3**********
1 CD4 MESSAGE FILE - DIRECTORY AND OLD TEXT
1 CD4**********
1 CD5 OUTPUT
1 CD5 XE COMMON - COMBUN
1 CD5**********
1 CD6 MESSAGE FILE - DIRECTORY AND TEXT UPDATES
1 CD6**********
1 CD7 LOCAL
1 CD7**********
1 CD8 AREA - NUMERICAL AREA INDICATOR FOR MESSAGE
1 CD8**********
1 CD9 DIRECT - MESSAGE DIRECTORY (SEE SDD 6.2.4.12)
1 CD9**********
1 CD10 I - INDEX TO BEGINNING OF CURRENT DIRECTORY ENTRY
1 CD10**********
1 CD11 IDCB - FILE MANAGER DATA CONTROL BLOCK
1 CD11**********
1 CD12 IERR - FILE MANAGER & XTOM RETURN CODE
1 CD12**********
1 CD13 NUMB - MESSAGE NUMBER WITHIN MESSAGE AREA
1 CD13**********
1 CD14 NBLIN - BLOCK NUMBER WITHIN FILE
1 CD14**********
1 CD15 CREATE MODE - NEXT BLOCK AVAILABLE FOR ALLOCATION
1 CD15**********
1 CD16 UPDATE MODE - NUMBER OF BLOCK CONTAINING MESSAGE
1 CD16**********
1 CD17 RECPOS - MESSAGE LOCATION WITHIN 128 WORD BLOCK (1, 33, 65 OR 97)
1 CD17**********
1 CD18 LOCAL
1 CD18**********
1 CD19 NOTES
1 CD19**********
1 CD20 USES APOS, CLOSE, CREATE, EXEC, IAND, KCVT, OPEN, READF, WRITF,
1 CD20**********
1 CD21 XERIN, XIRG, XRMIN, XRMSE, XTCOM, XUDFG
1 CD21**********
1 CD22 WHEN REPLACING AN EXISTING MESSAGE, A NULL RESPONSE WILL LEAVE THE
1 CD22**********
1 CD23 EXISTING TEXT IN PLACE.
1 CD23**********
1 CD24 MESSAGE UPDATING MAY BE TERMINATED AT ANY TIME BY ENTERING A X
1 CD24**********
1067 1 BEGIN XLMG
1068 2 OPEN XLMG
1069 3 IF FILE NOT FOUND
1070 4 THEN
1071 5 OUTPUT 'MESSAGE FILE CREATION'
1072 6 DO FOR EACH OF THE 32 DIRECTORY ENTRIES
1073 7 PROMPT FOR AREA ID AND MAXIMUM NUMBER OF MESSAGES FOR THIS ENTRY NUMBER
1074 8 IF RESPONSE WAS NULL
1075 9 THEN
1076 10 CLEAR ENTRY
1077 11 ELSE
1078 12 STORE ID
1079 13 COMPUTE AREA ORIGIN AND STORE
1080 14 CLEAR LAST MESSAGE NUMBER
1081 15 COMPUTE NUMBER OF BLOCKS AND STORE
1082 16 ENDIF
1083 17 ENDOD
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 XLMG
1149 1 BEGIN XL CRE
1150 2 CALL CREATE TO CREATE MDB/PDB LOG FILE
1151 2 ERREXIT IF CREATE ERROR TO :FILERR:
1152 2 INITIALIZE LOG RECORD BUFFER TO ZEROS
1153 2 SET # MDB FILES CURRENTLY USED TO ZERO
1154 2 SET MAXIMUM NUMBER MDBS TO 20
1155 2 CALL WRITE TO WRITE MDB RECORD TO LOG FILE
1156 2 ERREXIT IF WRITF ERROR TO :FILERR:
1157 2 DO FOR EACH REMAINING LOG RECORD
1158 3 DO FOR THIS PAIR OF USER ID'S
1159 4 CALL XCTGM TO PROMPT FOR MAXIMUM ALLOWED # PDB'S
1160 4 SET MAXIMUM # PDB FILES TO RESPONSE
1161 4 SET # PDB FILES CURRENTLY USED TO ZERO
1162 3 ENDDO
1163 3 CALL WRITE TO WRITE 1 PDB RECORD TO LOG FILE
1164 3 ERREXIT IF WRITF ERROR TO :FILERR:
1165 3 ENDDO
1166 2 CALL CLOSE TO CLOSE FILE
1167 2 ERREXIT IF CLOSE ERROR TO :FILERR:
1168 1 EXIT XL CRE
1169 2 :FILERR:
1170 2 CALL XRMSG TO DISPLAY FILE ERROR
1171 2 CALL CLOSE TO CLOSE FILE
1172 1 END XL CRE
FORTRAN CALLING SEQUENCE:

CALL XLPFL (NOCOD, CODES)

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

INPUTS IN CALLING SEQUENCE:

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

INPUTS IN COMMON:

XE(1) LU, XE(3) ISECU, XE(16) PRCNAM,
XE(12) ICR, XB(96) NOPARM, XB(108) PARMS

RTE FUNCTIONS USED:

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

FDS FUNCTIONS USED:

XERTN, XRPCP, XPMOV, XRMSSG,
XRPK, XRUPK, XTCOM

COMMON USED:

EQUIVALENCE (XE(1), LU ),
+ (XE(3), ISECU ), (XE(7), VALFL3),
+ (XE(16), PRCNAM ), (XE(85), TOKEMS),
+ (XE(12), ICR ), (XE(144), COMPAR),
+ (XE(145), COMBUS ), (XB(96), NOPARM),
+ (XB(108), PARMS )
1227 1 BEGIN XLPL
1228 2 COMPUTE SIZE OF FILE AS 0 PARAMETERS +3
1229 3 CALL CREATE TO CREATE PROMPT FILE
1230 4 ERREXIT IF CREATE ERROR TO :FILERR:
1231 5 STUFF SYNTAX RECORD 0 AND 0 PARAMETERS INTO LIST OF SHORT PROMPTS
1232 6 CALL WRITF TO WRITE SHORT PROMPT RECORD(S)
1233 7 ERREXIT IF WRITF ERROR TO :FILERR:
1234 8 IF ABSTRACT CODE IS MODIFY THEN
1235 9 CALL OPEN TO OPEN OLD PREAMPT FILE >XTMP
1236 10 ERREXIT IF OPEN ERROR TO :FILERR:
1237 11 CALL READ TO READ EXISTING ABSTRACT
1238 12 ERREXIT IF READ ERROR TO :FILERR:
1239 13 CALL EXEC TO DISPLAY EXISTING ABSTRACT
1240 14 ERREXIT IF WRITF ERROR TO :FILERR:
1241 15 IF RESPONSE IS CR (NO MODIFICATION) THEN
1242 16 CALL WRITF TO WRITE EXISTING ABSTRACT TO NEW FILE
1243 17 ELSE (NEW ABSTRACT WAS ENTERED)
1244 18 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1245 19 ENDIF
1246 20 ELSE (ABSTRACT CODE IS AD)
1247 21 CALL XTCOM TO PROMPT USER TO ENTER NEW ABSTRACT
1248 22 PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
1249 23 ENDIF
1250 24 IF PARAMETERS IS NOT ZERO THEN
1251 25 PERFORM EXTPRM TO GET DEFINITIONS FOR EACH PARAMETER
1252 26 ENDIF
1253 27 CALL CLOSE TO CLOSE PREAMPT FILE
1254 28 ERREXIT IF CLOSE ERROR TO :FILERR:
1255 29 IF THERE WAS AN OLD FILE THEN
1256 30 CALL CLOSE TO CLOSE OLD FILE >XTMP
1257 31 CALL PURGE TO PURGE OLD FILE >XTMP
1258 32 ENDIF
1259 33 EXIT XLPL
1260 34 :FILERR:
1261 35 CALL XPMUG TO DISPLAY ERROR CODE
1262 36 CALL CLOSE TO CLOSE NEW FILE
1263 37 CALL close to close OLD FILE >XTMP
1264 38 CALL PURGE TO PURGE OLD FILE >XTMP
1265 39 SET VALFLG = 0
1266 40 CALL XERTM TO RETURN TO MAIN ***NO RETURN TO HERE***
1267 41 END XLPL
1270 1 BEGIN FORMAT
1271 2 INITIALIZE TOKEN POINTER AND TOTAL WORD COUNT
1272 3 DO UNTIL EOS IS DETECTED IN RESPONSE
1273 4 ERREXIT IF RESPONSE IS NOT CHARACTER STRING TO :ERR1:
1274 5 ERREXIT IF RESPONSE IS TOO LONG (>128 WORDS) TO :ERR1:
1275 6 CALL INMOV TO MOVE RESPONSE TO BUFFER
1276 7 SET CONTROL CHARACTERS IN BUFFER
1277 8 INCREMENT TOTAL WORD COUNT BY THIS RESPONSE
1278 9 ERREXIT IF NEXT RESPONSE IS NOT A COMMA TO :ERR1:
1279 0 INCREMENT TOKEN POINTER TO NEXT CHARACTER STRING
1280 1 ENDDO
1281 2 SET REMAINDER OF BUFFER TO NULL
1282 3 CALL WRITF TO WRITE NEW RESPONSE TO PROMPT FILE
1283 4 ERREXIT IF WRITF ERROR TO :FILEERR:
1284 5 I END FORMAT
1285 1 :ERR1:
1286 1 CALL ERRMSG TO DISPLAY INVALID RESPONSE
1287 1 RETURN TO REPROMPT USER FOR ANOTHER RESPONSE
BEGIN EXTRM
DO FOR EACH ENTRY IN CODES ARRAY
BEGIN CASE (:NOCHNG; :MOD; :DEL; :ADD); CODES+1
:NOCHNG: (SPECs HAD NO CHANGES)
CALL READ TO READ EXISTING DEFINITION FROM >XLTP
ERREXIT IF READF ERROR TO :FILERR:
CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
ERREXIT IF WRITF ERROR TO :FILERR:
:MOD: (SPECs WERE MODIFIED)
CALL READ TO READ EXISTING DEFINITION FROM >XLTP
ERREXIT IF READF ERROR TO :FILERR:
CALL EXEC TO DISPLAY EXISTING DEFINITION
IF RESPONSE IS CR (NO RESPONSE) THEN
CALL WRITF TO WRITE EXISTING DEFINITION TO NEW FILE
ELSE (NEW DEFINITION WAS ENTERED)
PERFORM FORMAT TO FORMAT DEFINITION INTO FILE
ENDIF
:DEL: (SPECs WERE DELETED)
CALL POSNT TO POSITION >XLTP OVER THIS ENTRY
ERREXIT IF POSNT ERROR TO :FILERR:
:ADD: (SPECs WERE ADDED)
CALL XCTOM TO PROMPT USER TO ENTER NEW DEFINITION
PERFORM FORMAT TO FORMAT RESPONSE INTO FILE
END CASE
DO UNTIL RESPONSE IS CR (EXIT)
EXIT EXTRM IF RESPONSE IS CR
SEARCH SHOPT PROPTS FOR RESPONSE
ERREXIT IF NOT FOUND TO :ERR:
CALL READ TO READ EXISTING DEFINITION FROM NEW FILE
ERREXIT IF READF ERROR TO :FILERR:
CALL EXEC TO DISPLAY EXISTING DEFINITION
CALL XCTOM TO PROMPT USER TO MODIFY DEFINITION
IF RESPONSE IS POSITIVE
CALL POSNT TO REPOSITION NEW FILE BACK TO PREVIOUS RECORD
ERREXIT IF POSNT ERROR TO :FILERR:
PERFORM FORMAT TO FORMAT NEW DEFINITION INTO FILE
ENDIF
END
END EXTRM
FORTRAN CALLING SEQUENCE:

CALL XLPHO

XLPHO MODIFIES THE EXISTING PDB LOG FILE

INPUTS FROM COMMON:

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

INTERNAL VARIABLES IN COMMON:

XE(5) UNO - NUMBER OF USER ID A-Z (1-26)
XE(51) RECN - RECORD # WHERE UNO'S PDB LIST IS (2-14)
XE(52) PHAM - FILE NAME OF USER'S PDB (FULL 6 CHAR)
XE(55) DIFS - NUM-BER OF FILES THAT MUST BE DELETED
XE(2244) OUTBUF-AREA WHERE DISPLAY OF CURRENT FILES IS BUILT.

RTE ROUTINES USED:
CLOSE, EXEC, KEVT, PURGE, KREAD, WRITF

FDS ROUTINES USED:
XRCRA, XREXT, XRMVI, XRMG,
XROF, XRFN, XRCM

COMMON USED:

EQUIVALENCE (XEC(1), LU ), (XEC(3), QUA ), (XEC(8), SECU ),
+ (XEC(142), ICA ), (XEC(99), XER ), XEC(5) , XEC(51) , XEC(52) ,
+ (XEC(56), ATN ), (XEC(57), HSC ), + (XEC(99), IERR ), (XEC(100), INCB ),
+ (XEC(116), IBUF ), (XEC(2244), OUTBUF ),
+ XEC(304), IORC 

CO************
1387    1 BEGIN XLMHO
1388    2 DO UNTIL USER RESPONSES EXIT (2)
1389    3 CALL XCOM TO PROMPT USER FOR USER ID
1390    4 ERREXIT IF RESPONSE IS INVALID TO :ERR1:
1391    5 SAVE USER ID AND NUMBER (1-26) IN COMMON
1392    6 CALL READ TO READ IN THE RECORD CONTAINING THIS ID
1393    7 ERREXIT IF READ ERROR TO :FILERR:
1394    8 DISPLAY MAX # ALLORED FOR THIS USER AND # CURRENTLY USED
1395    9 CALL XCOM TO PROMPT USER FOR NEW MAXIMUM
1396   10 IF RESPONSE IS A MODIFICATION (NOT CR) THEN
1397   11 ERREXIT IF RESPONSE IS NOT VALID TO :ERR1:
1398   12 COMPUTE DIFFERENCE AS CURRENTLY USED - NEW MAXIMUM
1399   13 IF DIFFERENCE <= 0 THEN
1400   14 CALL WRITE TO REWRITE UPDATED RECORD TO FILE
1401   15 ERREXIT IF WRITE ERROR TO :FILERR:
1402   16 ELSE
1403   17 DISPLAY LIST OF CURRENT FILES
1404   18 SET SPURGED = 0
1405   19 DO UNTIL DIFFERENCE <= 0 OR RESPONSE IS EXIT (PERCENT)
1406   20 CALL XCOM TO PROMPT USER TO DELETE SPURGED FILES
1407   21 DO FOR EACH FILE NAME IN RESPONSE
1408   22 SEARCH LIST FOR FILE NAME
1409   23 ERREXIT IF INVALID NAME TO :ERR1:
1410   24 MOVE FILE NAME TO PURGE LIST
1411   25 INCREMENT SPURGED BY 1
1412   26 DECREMENT DIFFERENCE BY 1
1413   27 COMPRESS OLD FILE NAME OUT OF LIST
1414   28 ENDDO
1415   29 ENDDO
1416   30 IF RESPONSE WAS NOT EXIT THEN
1417   31 CALL WRITE TO REWRITE UPDATED LOG RECORD
1418   32 ERREXIT IF WRITE ERROR TO :FILERR:
1419   33 DO FOR SPURGED FILES
1420   34 CALL XCOM TO CREATE FILE NAME
1421   35 CALL PURGE TO PURGE FILE
1422   36 ENDDO
1423   37 ENDDIF
1424   38 ENDDIF
1425   39 ENDIF
1426   40 ENDIF
1427   41 CALL CLOSE TO CLOSE NDB/PDB LOG FILE
1428   42 EXIT XLMHO
1429   43 :ERR1:
1430   44 CALL XMSG TO DISPLAY ERROR IN RESPONSE
1431   45 RETURN TO REISSUE LAST PROMPT
1432   46 :FILERR:
1433   47 CALL XMSG TO DISPLAY FILE ERROR
1434   48 CALL CLOSE TO CLOSE NDB/PDB FILE
1435   49 END XLMHO
FORTRAN CALLING SEQUENCE FOR SPEC PROCESSOR:

CALL XLSPS

XLSPS PROCESSES INPUTS FOR ONE PARAMETER.

INTERNAL VARIABLES:

CLASS = (INTEGER, : WORD, XE(10)) 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 SPECS ARE TO BE STORED
MSG = (INTEGER, 1 WORD) IS THE CURRENT ERROR CONDITION
TYPE = (INTEGER, 1 WORD, XE(11)) IS THE PARAMETER TYPE

Routines Used:
KCVT, XRCPR, XRMV, XRMG, XRSET, XTCOM

XE COMMON USED:

EQUIVALENCE (XE(B), RETC ), (XE(9), RETN ),
+ (XE(10), CLASS ), (XE(11), TYPE ),
+ (XE(145), COMBUF ), (XE(37), ARGNO ),
+ (XB(96), NOPARM ), (XB(108), PARMS )
1474 1 BEGIN XSPS
1475 2 CALL KCVT TO CONVERT PARAMETER NUMBER TO ASCII
1476 2 DO: PRMP1:
1477 2 SET RETURN TO 1
1478 2 CALL XTCOM TO PROMPT FOR PROMPT,CLASS,TYPE AND I/O FLAGS
1479 2 ERREXIT IF RETURN CODE IS NOT ZERO TO :PRMERR:
1480 2 INITIALIZE THIS SHORT PROMPT AND ENTRY
1481 2 ERREXIT IF SHORT PROMPT IS NOT VALID (6 CHAR NAME) TO :PRMERR:
1482 2 ERREXIT IF THIS IS A DUPLICATE SHORT PROMPT
1483 2 CALL XRMOV TO MOVE SHORT PROMPT INTO SPECS
1484 2 ERREXIT IF CLASS IS NOT VALID TO :PRMERR:
1485 2 CALL XRSET TO SET CLASS IN ENTRY
1486 2 ERREXIT IF TYPE IS NOT VALID TO :PRMERR:
1487 2 ERREXIT IF CLASS IS ORDER AND TYPE IS SYMBOLIC STRING TO :PRMERR:
1488 2 CALL XRSET TO SET TYPE IN ENTRY
1489 2 ERREXIT IF I/O FLAGS ARE NOT VALID TO :PRMERR:
1490 2 ERREXIT IF TYPE IS SYMBOLIC STRING AND I/O FLAGS ARE NOT INPUT TO :PRMERR:
1491 2 CALL XRSET TO SET I/O FLAGS IN ENTRY
1492 2 IF CLASS IS DATU ELEMENT THEN
1493 3 IF TYPE IS NOT SYMBOLIC STRING THEN
1494 4 SET RETURN TO 2
1495 4 DO: PRMP2:
1496 4 CALL XT COM 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 4 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 5 ENDIF
1508 4 ENDIF
1509 4 SET SIZE =I DIMENSION * J DIMENSION * TYPE LENGTH
1510 4 ERREXIT IF SIZE IS NOT VALID TO :PRMERR:
1511 3 ELSE
1512 4 SET RETURN TO 3
1513 4 DO: PRMP3:
1514 4 CALL XT COM 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 XSPS
1521 2 :PRMERR: CALL XRMSGE TO DISPLAY ERROR MESSAGE
1522 2 GO TO (:PRMP1:,PRMP2:,PRMP3,), RETURN
1523 1 END XSPS
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
61  6  *  II=REQUEST & WM IS LU
62  6  *  SET UP FOR FDS TERMINATION
63  5   EXIT TO :XEND
64  5   ENDIF
65  4   ENDIF
66  3   ENDIF
67  2   ENDDO
68  2   :XEND
69  2   IF DUMP OPTION THEN
70  3   CALL XDUMP TO DUMP OUR PARTITION
71  2   ENDIF
72  2   RELEASE DWA TRACK SPACE
73  2   FREE EXEC'S & PROCESSOR'S CLASS NUMBERS
74  2   CALL RNRQ SET LOCK ON STATUS TABLE
75  2   UNLOCK THE MANAGER FROM THE PARTITION
76  2   CALL SLIBR DISABLE
77  2   CLEAR EXEC'S ID SEGMENT
78  2   CALCULATE EQT ADDRESS FOR ATTENTION ID
79  2   CLEAR THE STATUS TABLE ENTRY
80  2   CALL SLIBR ENABLE
81  2   CALL RNRQ CLEAR THE LOCK ON THE STATUS TABLE
82  2   ISSUE MESSAGE '***XMDZ SIGN OFF FOR LU "WM"'
83  2   CALL SLIBR GET PRIVLEGED AND DISABLED
84  2   CALL SLIST TO MAKE US DOORMANT
85  2   CLEAR OUR OWN ID
86  2   RESET EQT TO REMOVE ATTENTION ID
87  1   EXIT :SEIQ TO THE RTE DISPATCHER
88  1   END   XMG
80 1 *D0 CALLING PROCEDURE
81 1 *D0 JSB XMST
82 1 *D0
83 1 *********
84 1 *D1 SEQUENCE TABLE EXECUTION FROM &SEQTB
85 1 *D1
86 1 *********
87 1 *D2 INPUT
88 1 *D2 XMAM, XMPRM, XVSTA
89 1 *D2
90 1 *********
91 1 *D3 OUTPUT
92 1 *D3 XMCTP, XMPRM, XVSTA
93 1 *D3
94 1 *********
95 1 *D5 ROUTINES USED
96 1 *D5 CNUMP, EXEC, XMAG, XMKIL, XMFW (XMSCH),
97 1 *D5 XMSS, XMFIN, SLBR, SLBM
98 1 *D5
99 1 *********
BEGIN XMGT

1 CALL XMFT TO FIND BSEG TOC ENTRY
2 CALL XMST TO FIND ENDING SEQUENCE NUMBER INTO ENDING DISPLACEMENT
3 CALL XMST TO CONVERT ENDING SEQUENCE NUMBER INTO CURRENT DISPLACEMENT
4 COMPUTE CURRENT ENTRY ADDRESS
5 DO UNTIL THE LAST TABLE ENTRY IS EXECUTED OR
6 UNTIL THE TERMINATION ENTRY IS EXECUTED
7 IF PROCESSOR REQUIRES AN INTERFACE TABLE (WORD 3 BIT 8 IS SET) THEN
8 EXIT TO :ERROR IF INTERFACE TABLE NOT SPECIFIED (WORD 4 = 0) (PARMS = 1)
9 CALL XMFT TO SEARCH AWAY FOR INTERFACE TABLE (CHAIN 4)
10 EXIT TO :ERROR IF TABLE NOT FOUND (PARMS = 2)
11 IF TABLE NOT IN AWAY THEN
12 CALL INDIR TO RETRIEVE FROM AWAY
13 EXIT TO :ERROR IF SPACE IS NOT FOUND
14 ENDF
15 EXIT TO :ERROR IF INTERFACE TABLE NOT COMPLETE (WORD 3 BIT 8 CLEAR) (PARMS = 3)
16 EXIT TO :ERROR IF PROCESSOR NAME FIELDS DIFFERENT (BYTES 3-7) (PARMS = 4)
17 EXIT TO :ERROR IF VERSION FIELDS NOT EQUAL (WORD 3 BITS 9-15) (PARMS = 5)
18 ENDF
19 SEARCH SEGMENT TABLE FOR PROCESSOR TO BE SCHEDULED
20 ERREXIT TO :ERROR IF NOT FOUND
21 CALL BLIB TO BE PRIVILEGED
22 SET CURRENT TASK TO this PROCESSOR
23 CALL BLIB X TO BE UN-PRIVILEGED
24 IF PROCESSOR REQUIRES AN INTERFACE TABLE THEN
25 CALL EXEC TO CLASS I/O WRITE HEADER AND SPEC
26 CALL EXEC TO CLASS I/O WRITE HEADER AND SPEC
27 ENDF
28 CALL EXEC TO SCHEDULE PROCESSOR WITH WAIT
29 CALL XASCH TO RETRIEVE PARAMETERS FROM PROCESSOR
30 CALL XMFT TO FIND BSEG TOC ENTRY
31 FIND ADDRESS OF BSEG TO
32 COMPUTE CURRENT ENTRY ADDRESS
33 DO WHILE PROCESSOR REQUESTS AWAY MANAGEMENT (PARM = 1)
34 CALL XHANG TO HONOR AWAY REQUEST
35 ENDDO
36 CLEAR OUT CLASS BUFFERS FROM LAST PROCESSOR
37 ERREXIT TO :ERROR IF PROCESSOR REQUESTED TERMINATION (PARM = 8)
38 ERREXIT TO :ERROR IF PROCESSOR ABENDED (PARM = -32768)
39 IF REQUEST IS TO RESET CURRENT SEQUENCE ENTRY (PARM = 3) THEN
40 ERREXIT TO :ERROR IF RESET NUMBER IS ZERO
41 CALL XMST TO CONVERT SEQUENCE NUMBER INTO CURRENT ENTRY
42 ERREXIT TO :ERROR IF RESET SEQUENCE NUMBER IS NOT FOUR
43 IF TERMINAL ENTRY WAS JUST EXECUTED THEN
44 SET UP PARMS TO SHOW RESET SEQUENCE NUMBER
45 PERFORM :END - **NO RETURN EXPECTED**
46 ENDF
47 ELSE
48 IF REQUEST IS NOT NORMAL COMPLETION (PARM = 0) THEN
49 DISPLAY ERROR MESSAGE - INVALID REQUEST
50 ERREXIT TO :ERROR IF CURRENT AT IS DORMANT
51 CALL XMST TO SET PARAMETERS TO ABEND ASSOCIATED TASK
52 CALL YMPAN TO RESCHEDULE PROCESSOR
53 ERREXIT TO :ERROR TO TERMINATE SEQUENCE
54 ENDF
55 ENDF
56 INCREMENT CURRENT DISPLACEMENT TO NEXT ENTRY
57 ENDDO
171  SET PARM1 = 0 (NORMAL COMPLETION)
172  PERFORM :END: - NO RETURN EXPECTED
173  :ERROR1: SET PARM1 = 1
174  :ERROR5: SET PARM1 = 8 AND PARM5 TO APPROPRIATE REASON CODE
175  :END:
176  CALL SLIBR TO BECOME PRIVLEDGED
177  SET CURRENT TASK IN MGR, STATUS TABLE, AND ANA TO EXEC
178  CALL SLIBX TO BECOME UN-PRIVLEDGED
179  CALL XMPAW TO POST EXEC AND WAIT FOR NEXT REQUEST
180  END XMXQT
**CALLING PROCEDURE**

182 1 *00
183 1 *00
184 1 *00
185 1 *00
186 1 *00
187 1 *00
188 1 *00
189 1 *********
190 1 *01
191 1 *01
192 1 *01
193 1 *********
194 1 *02
195 1 *02
196 1 *02
197 1 *02
198 1 *02
199 1 *02
200 1 *02
201 1 *02
202 1 *02
203 1 *********
204 1 *03
205 1 *03
206 1 *03
207 1 *********
208 1 *05
209 1 *05
210 1 *05
211 1 *********
1 BEGIN XMAFR
2  SET NEW FE SIZE FIELD TO MAX(SIZE, 3)
3  INCREMENT TOTAL FREE SPACE BY NEW FE SIZE
4  INDEX TO FORWARD CHAIN POINTER (FCP) HEAD
5  START SEARCH WHILE FCP NOT = END-OF-CHAIN (-32768)
6  EXIT IF 'ADDR' < FCP VALUE
7  SET NEW FE FCP TO CURRENT FCP VALUE
8  SET CURRENT FCP VALUE TO 'ADDR'
9  SET NEW FE BACKWARD CHAIN POINTER (BCP) TO NEXT FE BCP VALUE
10  SET NEXT FE'S BCP VALUE TO 'ADDR'
11  PERFORM MERGE TO ATTEMPT COMBINATION OF NEW FE AND NEXT FE
12  OR ELSE
13  INDEX TO NEXT FE FCP
14  END LOOP
15  SET NEW FE'S FCP VALUE TO CURRENT FE'S FCP VALUE (-32768)
16  SET CURRENT FE'S FCP VALUE TO 'ADDR'
17  SET NEW FE BCP TO BCP HEAD VALUE
18  SET BCP HEAD TO 'ADDR'
19  END SEARCH
20  IF NEW FE BCP NOT = END-OF-CHAIN
21  THEN
22  PERFORM MERGE TO ATTEMPT COMBINATION OF PREVIOUS FE AND NEW FE
23  END IF
24  IF TOC SPACE FENCE IS WITHIN BOUNDARIES OF THE NEW
25  OR CONSOLIDATED FE, THEN
26  MOVE THE TOC SPACE FENCE TO BE ORIGIN OF THIS FE
27  ENDIF
28  END XMAFR
29 1 BEGIN MERGE
30  IF FE 1 IS ADJACENT TO FE 2
31  THEN
32  INCREMENT FE 1 SIZE FIELD BY FE 2 SIZE FIELD
33  SET FE 1 FCP TO VALUE OF FE 2 FCP
34  IF FE 1 FCP NOT = END-OF-CHAIN
35  THEN
36  SET FE 3 BCP TO ADDRESS OF FE 1
37  ELSE
38  SET BCP HEAD TO ADDRESS OF FE 1
39  ENDIF
40  ENDIF
41 1 END MERGE
CALLING PROCEDURE

JBB XMAGT
DEF +3
DEF OPTM
DEF SIZE

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

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

EXTERNAL SYMBOLS FROM XMAWA
XMFCP, XMFC, XMFRE

OUTPUT
A-REG - ADDRESS OF ALLOCATED BLOCK OR -32768 (OCTAL 100000)
INDICATING NONE AVAILABLE
EXTERNAL SYMBOLS FROM XMWA
XMFCP, XMFC, XMFRE
XMFC1, XMFC2

NOTES
IF SOME FREE 'SIZE' WORDS OR >= SIZE+3 WORDS IS NOT FOUND AN ERROR RETURN (A-REG = -32768) IS TAKEN
USES .EMIR
1 BEGIN XNAGT
  2 IF TOTAL FREE SPACE < MAX(SIZE,3)
  3 THEN
  4 CALL XNPK1 PURGE DNA ELEMENTS FROM AMA
  5 SET PHASE1 FLAG
  6 IF TOTAL FREE SPACE < MAX(3,SIZE), THEN
  7   RETURN VALUE= -32768
  8 EXIST TO:XMAGR
  9 ENDIF
 10  ENDIF
 11  ENDIF
 12 IF RETURN CODE IS NOT FOUND, THEN
 13 CALL XMAP
 14 IF RETURN CODE IS NOT FOUND, THEN
 15 RETURN VALUE= -32768
 16 EXIT TO:XMAGR
 17 ENDIF
 18 RETURN VALUE= AREA
 19 :XMAGR
 20 ENDIF
 21 CLEAR PHASE1 FLAG
 22 1 END XNAGT
318 1 BEGIN XMISR
319 2 SET INDEX TO APPROPRIATE CHAIN HEAD, I.E., FHEAD(OPTH)
320 3 START SEARCH WHILE POINTER NOT = END-OF-CHAIN (-32768), AND
321 4 WHILE TOC SPACE FENCE HAS NOT BEENcrossed
322 5 EXIT IF FE SIZE = MAX('SIZE', 3)
323 6 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
324 7 DECHAIN FE
325 8 RETURN ADDRESS OF AREA
326 9 EXIT IF FE SIZE >= MAX('SIZE', 3) + 3
327 10 DECREMENT TOTAL FREE SPACE BY MAX('SIZE', 3)
328 11 IF ALLOCATING FROM HEAD OF SPACE (OPTN = 0)
329 12 THEN
330 13 CREATE CHAIN POINTERS AND SIZE FIELDS IN BOTTOM OF SPACE
331 14 RECHAIN NEW FE
332 15 RETURN ADDRESS OF AREA
333 16 ELSE
334 17 CHANGE SIZE FIELD TO FE SIZE - MAX('SIZE', 3)
335 18 COMPUTE AND RETURN ADDRESS OF AREA
336 19 ENDIF
337 20 OR ELSE
338 21 INDEX TO NEXT FE
339 22 END LOOP
340 23 SET RETURN CODE TO 0(NOT FOUND)
341 24 END SEARCH
342 25 IF TOC SPACE WAS FOUND AT THE TOC SPACE FENCE, THEN
343 26 INCREMENT TOC SPACE FENCE OR 'SIZE'
344 27 ENDIF
345 28 END XMISR
**CALLING PROCEDURE**

1 *00 JSB XMAPG
1 *00 DEF CLSNQ

1 *01 PROVIDE AWS MANAGEMENT BASED ON REQUEST LIST (SEE TABLE 6.2-III)

1 *01 TABLE 6.2-III
1 *01

1 *02 INPUT CLSNQ - CLASS I/O NUMBER CONTAINING REQUEST LIST
1 *02 REQUEST LIST (SEE TABLE 6.2-III)
1 *02

1 *03 ID SEGMENT PARAMETERS (SEE TABLE 6.2-IV)
1 *03 REQUEST LIST FIELD 8
1 *03

1 *05 NOTES
1 *05 USES EXCON,XMAPR,XMACT,XMAPK,XMN
1 *05 XGON,XMSAL,XMSDA,XMSST,XMDST
1 *05

1 *05 NEITHER EXCON NOR THE PROCESSORS WILL MAKE AWS MANAGEMENT REQUESTS
1 *05 FOR DWA DATA (CLASS 3 & 5). ONLY THE MANAGER IS AWARE OF THE DWA
1 *05 AND IT WILL DUPLICATE INTERFACE TABLES AND SEQUENCE TABLES IN TO
1 *05 THE DWA AND COPY THEM BACK TO THE AWS AS NECESSARY.
1 BEGIN XMAP
2 RETRIEVE AWA MANAGEMENT REQUEST LIST FROM CLASS I/O NUMBER
3 CLEAR RETURN PARM
4 CLEAR REWRITE FLAG
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 > TOC SIZE, THEN
10 CALL XMAPK TO COLLAPSE AWA IF TOC TOO SCATTERED FOR XEXEC BUFFER SIZE
11 ENDIF
12 WRITE CHAIN HEADS, TOTAL FREE SPACE AND TOC TO CLASS I/O
13 STORE CLASS I/O NUMBER IN REQUEST WORD EIGHT
14 SET REQUESTED SIZE FROM TOC SIZE
15 SET REWRITE FLAG
16 ELSE
17 IF REQUEST TO CLEAR (CODE 17)
18 THEN
19 GET A(DWA) FROM XMDWA
20 SAVE HEADER AND DIRECTORY-SIZE
21 CLEAR XMDWA THRU XMD02
22 BUILD AN FE AT XMDWA FOR AWA SIZE
23 CALL XMAST TO ALLOCATE A TOC ENTRY FOR DWA
24 CHAIN IN TOC ENTRY TO XMD03
25 CALL XMAST TO ALLOCATE SPACE FOR DWA
26 SET LOCATION, SIZE, & KEY IN THE TOC
27 SET DIRECTORY ADDRESS AT XMDWA
28 CLEAR THE DIRECTORY
29 SET LU, TRACK NUMBER, & NUMBER OF TRACKS IN THE DIRECTORY
30 ELSE
31 CALL XMTFN TO SEARCH TOC FOR INDICATED ENTRY
33 (:RETRAVE:, :RETRAVE:) REQUEST CODE
34 :VERIFY:
35 IF ENTRY NOT FOUND
36 THEN
37 SET RETURN PARM1 AND PARM2 (2 & INDEX)
38 ENDIF
39 :VERALO:
40 IF ENTRY ALREADY EXISTS
41 THEN
42 IF ALLOCATE REQUEST (33)
43 THEN
44 SET RETURN PARM1 AND PARM2 (3 & INDEX)
45 ELSE
46 IF TYPE, SIZE AND 1-DIM FIELDS DO NOT MATCH
47 THEN
48 SET RETURN PARM1 AND PARM2 (4 & INDEX)
49 ENDIF
50 ENDIF
51 ELSE
52 CALL XMAST 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 XMAST TO ALLOCATE DATA SPACE
437 8
438 8
439 8
440 8
441 9
442 9
443 10
444 10
445 11
446 10
447 11
448 10
449 9
450 9
451 8
452 8
453 8
454 8
455 7
456 6
457 6
458 6
459 6
460 7
461 6
462 7
463 6
464 7
465 8
466 7
467 7
468 8
469 8
470 9
471 8
472 9
473 9
474 9
475 8
476 7
477 6
478 6
479 6
480 6
481 7
482 7
483 7
484 7
485 7
486 7
487 7
488 7
489 6
490 7
491 7
492 8
493 7

IF SPACE NOT AVAILABLE THEN
SET RETURN PARM1 AND PARM2 (1 & INDEX) ELSE IF DATA ELEMENT (CLASS 2) THEN IF CHARACTER STRING (TYPE 4 - 8) THEN INITIALIZE AREA TO BLANKS ELSE INITIALIZE AREA TO ZEROS ENDIF ENDIF CHAIN IN NEW TOC ENTRY ENDIF IF CLASS EQ 4 OR 6 AN SEQUENCE TABLE OR INTERFACE TABLE THEN CALL XMDAL DMA ALLOCATION ENDIF ENDIF :RENAME: IF ENTRY NOT FOUND THEN SET RETURN PARM1 AND PARM2 (2 & INDEX) ELSE CALL XMTFN TO SEARCH TOC FOR NEW ENTRY AND DETERMINE CHAIN POSITION IF ENTRY FOUND THEN SET RETURN PARM1 AND PARM2 (3 & INDEX) ELSE CALL XMAGT TO ALLOCATE NEW TOC ENTRY IF SPACE NOT AVAILABLE THEN SET RETURN PARM1 AND PARM2 (1 & INDEX) ELSE COPY OLD ENTRY ATTRIBUTES INTO NEW ENTRY AND CHAIN IN TO TOC DECHAIN OLD ENTRY CALL XMAGF TO RETURN OLD ENTRY TOC SPACE TO FE POOL ENDIF ENDIF :DELIVER: IF ENTRY FOUND THEN GENERATE KEY 1 LESS THAN FOUND KEY CALL XMTFN FOR GENERATED KEY DECHAIN TOC ENTRY CALL XMAGF TO RETURN TOC ENTRY SPACE TO FE POOL IF CLASS EQ 4 OR 6, THEN CALL XMDDA DMA DEALLOCATION ENDIF ELSE IF DELETE REQUEST ($)
THEN SET RETURN PARM1 AND PARM2 (2 & INDEX)
ENDIF
ENDIF

:STORE:

IF ENTRY NOT FOUND

THEN

SET RETURN PARM1 AND PARM2 (2 & INDEX)

ELSE

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

DISPLACEMENT OR SPECIFIED SIZE < 0, OR

DISPLACEMENT + REQUESTED SIZE > ALLOCATED SIZE

THEN

SET RETURN PARM1 AND PARM2 (4 & INDEX)

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 (1 & INDEX)

ELSE

IF REQUESTED SIZE = ZERO

THEN

CALCULATE AMOUNT OF DATA TO RETRIEVE AS ACTUAL SIZE MINUS DISPLACEMENT

STORE COMPUTED SIZE IN REQUEST WORD SIX

ENDIF

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

THE ELEMENT EXISTS ONLY ON THE DMA

CALL XMDRT MOVE INTO AXA

IF NO SPACE THEN

SET RETURN PARM1 AND PARM2 TO (1, INDEX)

EXIT TO XMR EX

ENDIF

ENDIF

WRITE VALUES TO CLASS I/O

STORE TYPE IN LOW BYTE OF REQUEST WORD 1

STORE CLASS NUMBER IN REQUEST WORD 8 CH T

SET NEWWRITE FLAG

ENDIF

ELSE

WRITE TOC ENTRY TO CLASS I/O

STORE CLASS NUMBER IN REQUEST WORD EIGHT

SET NEWWRITE FLAG

ENDIF

: XMR EX

ENDIF
CALLING PROCEDURE

JSB XMTFN

DEF KEYS

EXAMINE THE TOC FOR AN ENTRY EQUAL TO 'KEYS'

INPUT

KEYS - ADDRESS OF FOUR WORD KEY TO BE LOCATED IN THE TOC

EXTERNAL SYMBOLS FROM XAMAN

XHMD

OUTPUT

A-REG - ADDRESS OF TOC ENTRY MATCHING 'KEY' OR

ADDRESS (WITH INDIRECT BIT SET) OF PREVIOUS TOC ENTRY

WHERE 'KEYS' COULD BE CHAINED IN

BEGIN XMTFN

ISOLATE CLASS FROM KEY AND INDEX "O A-PRIORITY CHAIN HEAD

START SEARCH WHILE CHAIN POINTER .EQ. END-OF-CHAIN (-32768)

COMPARE LAST THREE WORDS OF 'KEYS' TO TOC ENTRY

EXIT IF MATCH

RETURN ADDRESS OF ENTRY

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

RETURN ADDRESS OF PREVIOUS ENTRY WITH INDIRECT BIT SET

OR ELSE

INDEX TO NEXT TOC ENTRY ON CHAIN

END LOOP

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

END XMTFN
CALLING PROCEDURE

JSB XMSST

ADJUST SEQUENCE LOCATION COUNTER

INPUT
A-REG - INTEGER VALUE OF SEQUENCE NUMBER TO BE LOCATED
EXTERNAL SYMBOLS
SEAD - ADDRESS OF EXECUTING SEQUENCE TABLE IN AW
ENDTB - DISPLACEMENT TO LAST ENTRY OF EXECUTING SEQUENCE TABLE

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

BEGIN XMSST
SAVE RESET SEQUENCE NUMBER IN LOCAL VARIABLE SEGNO
SET DISPLACEMENT IN A-REG TO ZERO
GET ADDRESS OF FIRST ENTRY IN SEQUENCE TABLE INTO X-REG
START SEARCH UNTIL CURRENT ENTRY DISPLACEMENT > LAST ENTRY DISPLACEMENT
EXIT IF INDICATED NUMBER FOUND
END LOOP OR ELSE
INCREMENT DISPLACEMENT IN A-REG TO NEXT ENTRY
SET A-REG TO -32768 TO INDICATE NUMBER NOT FOUND
END SEARCH

END XMSST
1 BEGIN XMPAW
2  *0 D
3   ENTRY: JSB XMPAW OR CALL XMPRM
4       DEF RETURN ADDRESS
5  *0 D
6  *0 D
7   INPUTS: XMPRM HAS REPLY
8  *0 D
9  *0 D
10  *0 D
11  OUTPUTS: XMPRM HAS REQUEST,
12       AND XVSTB IS UPDATED.
13  *0 D
14  *0 D
15   DO UNTIL GOOD PARMS RECEIVED OR TOP AT TERMINATES
16  *0 D
17    IF ABORT CURRENT FLAG CN(- STBLU) THEN
18  *0 D
19      SET ABORT FLAG IN XMPRM
20  *0 D
21      TURN OFF ABORT CURRENT FLAG
22  *0 D
23      ENDF
24  *0 D
25      GET CURRENT TOP ASSOCIATED TASK(AT)
26  *0 D
27      SET IN PARM 1 FIELD OF MANAGER'S ID SEGMENT
28  *0 D
29      JSD XVPAW
30  *0 D
31      DEF *3 RETURN
32  *0 D
33      DFC 0 MANAGER CALL
34  *0 D
35      DEF XMPRM PARM FIELD
36  *0 D
37      THIS IS AN IMPLIED WAIT
38  *0 D
39    3  I/XMSCN GET CURRENT XVSTB ENTRY (XUSTA)
40  *0 D
41    3  IF THERE HAS BEEN A CALL TO PRNM (P1 FIELD IS NOT TOP AT) OR
42  *0 D
43      IL SEGMENT IS DORMANT OR
44  *0 D
45      ID SEGMENT IS NOT OUR SON THEN (TOP AT HAS TERMINATED)
46  *0 D
47    4  IF PARM IS NOT 0, 3, 8, 9, OR -32768 THEN
48  *0 D
49      SET PARM TO -32768
50  *0 D
51      PRINT ERROR "INVALID REQUEST"
52  *0 D
53      ELSE
54  *0 D
55      SET PARM FIELD FROM MANAGER'S ID SEGMENT
56  *0 D
57      ENDF
58  *0 D
59      ELSE (TOP AT IS STILL ACTIVE AND RETURNED VIP PARM)
60  *0 D
61    3  IF PARM NOT 1 OR 2 THEN
62  *0 D
63      CALL XMKIL TO ABORT TOP AT
64  *0 D
65      PRINT ERROR "INVALID REQUEST"
66  *0 D
67      ELSE
68  *0 D
69      SET PARM FROM CURRENT ID SEGMENT
70  *0 D
71      ENDF
72  *0 D
73  *0 D
74  *0 D
75  *0 D
76  *0 D
77  *0 D
78  *0 D
79  *0 D
80  *0 D
81  *0 D
82  *0 D
83  *0 D
84  *0 D
85  *0 D
86  *0 D
87  *0 D
88  *0 D
89  *0 D
90  *0 D
91  1 END XMPAW
693 1 F00 4
694 1 F00 4
695 1 F00 4
696 1 F00 4
697 1 F01 4
698 1 F01 4
699 1 F01 4
700 1 F01 4
701 1 F03 4
702 1 F03 4
703 1 F03 4
704 1 F03 4
705 1 F05 4
706 1 F05 4
707 1 F05 4
708 1 F05 4
709 1 BEGIN 4
710 2 GET NUMBER OF DMA TRACKS FROM P5 4
711 2 CALCULATE SIZE OF EDWA(3+6H) N IS # OF TRACKS 4
712 2 CALL XMTFN (EDWA) 4
713 2 CALL XNAGT (0,6) TOC ENTRY FOR EDWA 4
714 2 CALL XNAGT (1,SIZE) DATA AREA FOR EDWA 4
715 2 INITIALIZE & CHAIN EDWA TOC ENTRY 4
716 2 SET # OF TRACKS FOR EDWA 4
717 2 CLEAR EDWA 4
718 2 CALL EXEC (DISC TRACK ALLOCATION) 4
719 2 IF DISC ADDRESS .EQ. -1 TRACKS NOT AVAILABLE 4
720 2 THEN 4
721 2 ISSUE MESSAGE '***XMDA "N" TRACKS NOT AVAILABLE' 4
722 2 SET ERROR RETURN 4
723 2 ELSE 4
724 3 SET DISC ADDRESS IN EDWA 4
725 3 SET ADDRESS OF EDWA FOR DMA MANAGEMENT 4
726 2 ENDIF 4
727 1 END XMDIN
BEGIN XDBAL

DWA ALLOCATION
CALLING Procedure
JSB XDBAL

FUNCTION
ALLOCATE DWA SPACE

INPUTS
A(TOC ENTRY OF THE AMA ELEMENT)

IN TRM

OUTPUTS
UPDATE TOC ENTRY FOR "WA

RETURNS BREG= ZERO- ALLOCATION COMPLETE
BREG= -1 ERROR CONDITION

NOTES
CALLS XXNDS, XMBST

IF (DWA) NE. 0 THEN
SET START-ADDRESS TO FIRST TRACK WORD IN DWA DIRECTORY
GET SIZE(IN WORDS FROM TOC ENTRY)
SIZE(IN SECTORS) = (SIZE+63/64)
DO FOR NUMBER OF TRACKS IN DWA OR DWA SIZE .GE. SIZE
SET BITMUM TO ZERO
DO UNTIL DWA SIZE .GE. SITE OR BITMUM .GE. 96
CALL XRNBD (0,BITMUM,START-ADDRESS)
STARTBIT =BITMUM
CALL XRNDB (1,BITMUM,START-ADDRESS)
DWA-SIZE=BITMUM-STARTBIT
END-0
IF BITMUM .GE. 96, THEN
START-ADDRESS=START ADDRESS+1 TRACK ADDRESS
ENDIF
ENDIF

IF D:A"C AREA FOUND, THEN
* START=A:"F" & BITMUM DEFINES THE TRACK & SECTOR
SET DISC ADDRESS & SIZE IN TOC ENTRY
CALL XPST (TOC-ENTRY)
SET NORMAL RETURN
ELSE
ENDIF

ISSUE MESSAGE "***XNOS NO DWA SPACE REMAINING"
SET ERROR RETURN
ENDIF

END XDBAL
1 BEGIN XMODA
2 * DMADealocation
3 CALLING PROCEDURE
4 JSB XMODA
5 * *
6 FUNCTION
7 DELETE DMA ELEMENT WHICH CORRESPONDS
8 TO THE ANA ELEMENT
9 * *
10 INPUTS
11 TOC ENTRY ADDRESS OF ANA ELEMENT
12 IN YREG
13 * *
14 NOTES
15 USES XMBST
16 * *
17 IF A(SDWA) .NE. 0, THEN
18 GET DISC ADDRESS
19 GET DATA SIZE
20 * FREE THE DISK AREA
21 CALL XMBST (TOC ENTRY)
22 * *
23 ENDF
24 * *
25 1 ENO XMODA
1 BEGIN XHOST
2 * DWA STORE DATA
3 * CALLING PROCEDURE
4 JSB XHOST
5 * FUNCTION
6 * UPDATE DWA ELEMENT WHICH CORRESPONDS
7 TO THE ANA ELEMENT
8 * INPUTS
9 TO ENTRY ADDRESS OF ANA ELEMENT
10 IN TREG
11 *
12 * OUTPUTS
13 * UPDATE DWA ELEMENT ON DISC TRACKS
14 *
15 * NOTES
16 * USES WRITE
17 *
18 IF A(EDUA) NE 0, THEN
19 SET DISC ADDRESS FOR THE WRITE
20 SET DATA ADDRESS FOR THE WRITE
21 WRITE DATA
22 ENDIF
23 END XHOST
029  1  BEGIN XH9RT
030    2  *  DNA RETRIEVE
031    3  * Calling Procedure
032    4  JSB XH9RT
033    5  *  FUNCTION
034    6  *  RETRIEVE DNA DATA INTO AWA
035    7  *  INPUTS
036    8  * A(TOC ENTRY) IN VREG
037    9  10  *  OUTPUTS
038     11  * ADDRESS OF DATA IN THE TOC
039     12  * REG=0, RETRIEVE SUCCESSFUL
040     13  * MINUS, ERROR NO DNA DATA
041     14  15  *  NOTES
042     16  * USES XHAGT, XHOMA, EXECREAD
043     17  18  IF NO DNA DIRECTORY, THEN
044     19  20  SET ERROR CODE -5
045     21  22  ELSE
046     23  24  CALL XHAGT, GET DATA SPACE
047     25  26  IF NO SPACE, THEN
048     27  28  SET ERROR CODE -1
049     29  30  ELSE
050     31  32  SET DATA ADDRESS IN TOC
051     33  34  GET DISC ADDRESS
052     35  36  READ DATA INTO AWA
053     37  38  READ DATA INTO AWA
054     39  40  SET RETURN CODE TO 0
055     41  42  ENDF1
056     43  44  ENDF1
057     45  46  ENDF1
058     47  48  ENDF1
059     49  50  ENDF1
060     51  52  END XH9RT
BEGIN XMST
CALLING PROCEDURE
JSB XMST

FUNCTION
SET ALLOCATION & DEALLOCATION IN DWA DIRECTORY

INPUTS
AWA TOC ENTRY

OUTPUTS
UPDATES DWA DIRECTORY

GET TRACK-ADDRESS FROM TOC
SUBTRACT START OF TRACKS FROM DWA DIRECTORY FOR RELATIVE TRACK
DIVIDE SECTOR ADDRESS BY 16(NUMBER OF BITS/WORD)
QUOTIENT IS NUMBER OF RELATIVE WORDS
REMAINDER IS BIT-POSITION(BP)
WORD ADDRESS=DWA ADDRESS*RELATIVE TRACK+RELATIVE WORDS
SUBTRACT BIT-POSITION FROM 16 GIVING NBFW(NUMBER BITS IN FIRST WORD)
GET DATA SIZE, ADD 63, DIVIDE BY 64 GIVING NUMBER OF SECTORS(BITS)
BITS-NBFW=REMAINING BITS(RB)
IF RB =0, THEN
SET NUMBER OF WORDS(NW) TO ZERO
SET NUMBER OF BITS LAST WORD(NBLW) TO ZERO
SET NBFW TO BITS
ELSE
DIVIDE RB BY 16
SET NW TO QUOTIENT
SET NBLW TO REMAINDER
ENDIF
L-LOAD DATA POINTED TO BY WORD ADDRESS
ROTATE LEFT (BP+NBFW-1 BITS) SAVE BP BITS & POSITION NBFW BITS
EXCLUSIVE OR SIGN BIT (ON TO OFF; OFF TO ON)
IF NBFW .GT. 1, THEN
SHIFT RIGHT (ARITHMETIC) BY NBFW-1 PROPAGATE BITS
ENDIF
IF DATA IS NEGATIVE, THEN
SET FULL WORD TO -1
ELSE
SET FULL WORD TO 0
ENDIF
ROTATE RIGHT BY BP, RESET SAVED BITS
STORE WHERE WORD-ADDRESS POINTS
DO WHILE NW.GT. 0
WORD-ADDRESS=WORD-ADDRESS+$
SET FULL WORD INTO WHERE WORD-ADDRESS POINTS
ENDIF
IF NBLW.GT. 0, THEN
WORD-ADDRESS=WORD-ADDRESS+$
LOAD DATA POINTED TO BY WORD-ADDRESS
IF NBLW.GT. 1, THEN
ROTATE LEFT NBLW-1 BITS
ENDIF
EXCLUSIVE OR SIGN BIT
IF NBLW.GT. 1, THEN
SHIFT RIGHT ARITHMETIC BY NBLW-1
ENDIF
STORE DATA WHERE WORD-ADDRESS POINTS
1 BEGIN XMAPK
2 *00 CALLING PROCEDURE
2 *00 JSB XMAPK
2 *00 DEF OPTION
2 *00 DEF PHASE 1 FLAG
2 *00 GENERAL COLLAPSE INTERFACE
2 *00 OUTPUT
2 *00 RETURNS AREA ADDRESS FOR AREA FOUND
2 *00 RETURNS -32768 AREA NOT FOUND
2 *00 NOTES
2 *05 USES XMPK1,XMPK2,XMPK3,XMSRC
2 *05 IF PHASE1 FLAG NOT SET, THEN
2 *05 CALL XMPK3 PURGE DWA ELEMENTS FROM AWA
2 *05 PERFORM XMSRC(OPTN,SIZE)
2 ELSE
2 *05 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 *05 ENDIF
2 SET RETURN VALUE TO RETURN CODE
1 END XMAPK
BEGIN XMPKI
CALLING PROCEDURE
JSB XMPKI

PHASE 1 OF COLLAPSE(PURGE DWA ELEMENTS)

OUTPUT

UPDATES ADDRESS FIELD IN TOC
FOR EACH DWA DATA ELEMENT

NOTES

USES XMAFP,XMD6,XMLD6,XMLP1
CURRENTLY ONLY CLASS 4 & 6 ELEMENTS
INTERFACE TABLES & SEQUENCE TABLES
ARE DWA ELEMENTS.

INCREMENT XMLP1 UPDATE PHASE 1 COUNT
IF DMA DIRECTORY ADDRESS NOT EQ 0, THEN
SAVE X & Y REGS
DO FOR ALL CLASS 4 ELEMENTS
CALL XMAFR(DATA,SIZE)
ENDDB
DO FOR ALL CLASS 4 ELEMENTS
CALL XMAFR(DATA,SIZE)
ENDDB
RESTORE X & Y REGS
ENDIF
END XMPKI
900 1 BEGIN XMFK2    CALLING PROCEDURE
901 2 *00 JSB XMFK2
902 2 *00 JSB XMFK2
903 2 *00 JSB XMFK2
904 2 *01 \ PHASE 2 OF COLPASE(COLPASE TOC ENTRIES)
905 2 *01 \ REORDER TOC ENTRIES
906 2 *01 \ UPDATES XMFC(TOC FENCE ADDRESS)
907 2 *03 \ NOTES
908 2 *05 \ USES XMFC, XMFCP, XMFCM, XMFCM, XMFCM, XMFCM, XMFCM
909 2 *05 \ INCREMENT XMFP2 UPDATE PHASE 2 COUNT
910 2 *06 \ ONLY IF THERE ARE FREE ELEMENTS AND THEY ARE IN THE TOC
911 2 *07 \ WILL A TOC COMPRESS BE DONE.
912 2 *08 \ GET FIRST-FREE(FREE=TRUE) FROM XMFCP
913 2 *09 \ GET JLSZ FROM THE FREE ELEMENT
914 2 *10 \ NWSIZ=OLSIZ-B
915 2 *11 \ IF NWSIZ .GT. 0, THEN THE FREE ELEMENT WILL BE DEPLETED
916 2 *12 \ GET NEXT-FREE FROM FIRST-FREE'S FCP
917 2 *13 \ SET INTO XMFCP DELETE FROM THE FORWARD CHAIN
918 2 *14 \ IF XMFCP.NE.-32768, THEN IF NOT THE LAST FREE ELEMENT
919 2 *15 \ SET NEXT-FREE'S BCP TO -32768
920 2 *16 \ ELSE
921 2 *17 \ SET XMFCP TO -32768 DELETE FROM BACKWARD CHAIN.
922 2 *18 \ ENDIF
923 *19 \ ELSE
924 2 *20 \ OLSIZ=NWSIZ UPDATE LENGTH IN FREE ELEMENT
925 2 *21 \ ENDIF
926 2 *22 \ NWTFC=A(2STP)+NWSIZ THE NEW TOC IS LAST 8 WORDS OF FIRST-FREE
927 2 *23 \ OLTFC=XMFC-8 OLD TOC IS THE ENTRY MOVE THE FENCE
928 2 *24 \ TMPKET=OLTTC'S KEY -1 CONSTRUCT A KEY TO FIND PREVIOUS ENTRY
929 2 *25 \ CALL XMFCM(TMPKET) FIND PREVIOUS
930 2 *26 \ SET PTROC FROM AREG
931 2 *27 \ COPY TOC ENTRY FROM OLTTC TO NWTTC
932 2 *28 \ STORE NWTTC ADDRESS INTO PTROC'S CHAIN
933 *29 \ CALL XMFCM(OLTTC,TOC-SIZE) FREE THE OLD TOC ENTRY
934 *30 \ ENDB
935 1 END XMFK2
BEGIN XMPK3
  CALLING PROCEDURE
  JSB XMPK3
  FUNCTION
  COMPRESS ASA DATA AREA BY MIG ? ATING DATA AREAS
  TO THE HIGHER ADDRESSES AND FREE AREAS TO LOWER ADDRESSES
  OUTPUT
  UPDATES XMBCP, XMFCP, XMFNC, XMFRE, XMP3
  NOTES
  USES XMAFR
  INCREMENT XMP3 (NUMBER OF PHASE 3's)
  IF XMBCP .NE. -32768, THEN THERE ARE FREE AREAS
  CALL XMPK3, INSURE TOC IS COMPRESSED
  DO WHILE XMBCP .GT. XMFCN UNTIL 1 FREE AREA ADJACENT TO THE FENCE
  IF RCY OF LAST-FREE .EQ. -32768 ONLY 1 FREE AREA
  SET HIGH-WATER TO XMFCN
  ELSE
     MULTIPLE AREAS
     SET HIGH-WATER TO BCP OF LAST-FREE
     ENDIF
  ENDIF
  PERFORM XMCAP(LAST-FREE)
  PERFORM XMCAP(HIGH-WATER)
  IF TOC-ADDRESS .EQ. 0
     EXIT
     PERFORM XMAVP(TOC-ADDRESS)
  ENDIF
END XMPK3
1064 1 BEGIN XMTSC  TOC SEARCH
1065 2 * FIND TOC ENTRY WHICH HAS DATA ADDRESS GREATER THAN
1066 2 * HIGH-WATER AND LESS THAN LOW-WATER, AND HAS A DATA
1067 2 * ADDRESS GREATER THAN ANY OTHER FOUND ON THIS SEARCH.
1068 2 * RETURN THE TOC-ADDRESS OR (NONE FOUND).
1069 2 * FIRST TOC ENTRY IS AT SYMBOL XMMA, THE
1070 2 * LAST TOC-ENTRY IS AT XMFC-8.
1071 2 TOC-ENTRY=A(XMMA)
1072 2 TEST-AD=0; TEST-TOC=0
1073 2 DO UNTIL TOC-ENTRY GE XMFC
1074 3 IF DATA ADDRESS IN TOC-ENTRY IS: 
1075 4 NE 0, AND IS
1076 4 LT LOW-WATER, AND IS
1077 4 GT HIGH-WATER, AND IS
1078 4 GT TEST-AD,
1079 3 THEN
1080 4 TEST-AD=DATA ADDRESS
1081 4 TEST-TOC=TOC-ENTRY
1082 3 ENDIF
1083 3 ADD 8 TO TOC-ENT
1084 2 ENDDO
1085 2 TOC-ADDRESS=TEST-TOC  RETURN 0 OR A TOC ADDRESS
1086 1 END XMTSC
1088 1 BEGIN XMAIN  AXA MOVE
1089 2 * MOVE THE DATA DEFINED BY THE TOC(WHICH IS
1090 2 * IMMEDIATELY ABOVE THE LAST FREE AREA) INTO
1091 2 * THE BOTTOM OF THE LAST FREE
1092 2 * UPDATE THE LENGTH OF THE RESULTING FREE AREA.
1093 2 * GET DATA ADDRESS FROM THE TOC
1094 2 SAVE THE FIRST THREE WORDS OF THE DATA AREA
1095 2 CALL XMAIN(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 XMAIN
FORTRAN CALLING PROCEDURE

CALL XPATR (LU, INBUF, INTLOG, MRBUF, INUM, 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
INTLOG - INPUT/OUTPUT BUFFER OF 7+(8 PARAMETERS + 1) WORDS
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTLOG - LENGTH OF INTLOG
MRBUF - MANAGER REQUEST BUFFER (6A WORDS) USED TO COMMUNICATE
WITH THE FDS MANAGER. MAY BE USED AS A SCRATCH AREA BY
THE PROCESSOR EXCEPT ACROSS PROCESSOR SERVICE CALLS.
NUM - 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 DRIVE WHERE THE DATA IS TO
BE OBTAINED/STORED. A ZERO ENTRY INDICATES INPUT DATA
WHICH IS LITERAL DATA STORED WITHIN THE INTERFACE TABLE.
TYPE - DATA TYPE CODE OF THE PARAMETER
SIZE - TOTAL NUMBER OF WORDS OF LITERAL DATA,
TOTAL NUMBER OF WORDS OF REFERENCED INPUT DE,
TOTAL NUMBER OF WORDS OF REFERENCED OUTPUT DE IF
SUBSCRIPTED OR ZERO IF NOT SUBSCRIPTED,
TOTAL NUMBER OF BLOCKS OF REFERENCED INPUT DE OR
ZERO IF OUTPUT DE.
IDIN - COLUMN LENGTH OF A DATA ELEMENT, MAXIMUM RECORD SIZE OF
AN INPUT DE OR THE LENGTH OF A SYMBOLIC STRING. ZERO
IF AN UNSUBSCRIPTED OUTPUT.
DSPTY - DISPLACEMENT FROM THE BEGINNING OF THE DATA FOR
SUBSCRIPTED DATA ELEMENTS ELSE ZERO.
FOR AN INPUT DE THE RTE FILE MANAGER TYPE CODE IS
RETURNED IN DSPRT.

*********

EXTERNAL SYMBOLS
(SEE XPEG)

*********

INTERNAL VARIABLES

*********

NOTES

USES .ENTRY, XPEG, XPIN, XPATR, XPFMT, XPR, APSUC, XPATR, XVSUB

XPATR IS IMPLEMENTED AS A SINGLE MODULE CONTAINING THE ENTRY
POINTS XPATR, XPEG AND XPUT
BEGIN XPATR
1 PERFORM XPINI(XPGET) TO INITIALIZE GLOBALS AND INTERFACE TABLE
2 EXIT TO :XPs13; (XPGET) IF PARAMETER IS OUT OF RANGE
3 EXTRACT NAME FROM INTERFACE TABLE
4 SET DSPIT TO ZERO
5 IF LITERAL PARAMETER (NAME IS ZERO)
6 THEN
7 COPY TYPE, SIZE AND IDIM FROM INTERFACE TABLE
8 ELSE
9 IF SUBSCRIPTED (INTERFACE TABLE DISP OR S FIELDS ARE NON-ZERO)
10 THEN
11 PERFORM XPBIC(XPGET) TO RETRIEVE TOC ENTRY AND COMPUTE DISPLACEMENT
12 STORE TYPE, SIZE, IDIM AND DSPIT
13 ELSE
14 IF BORE
15 THEN
16 PERFORM XPFSN(XPGET) TO QUALIFY FILE NAME
17 ENDIF
18 IF INPUT PARAMETER
19 THEN
20 CALL XPREG TO RETRIEVE TOC ENTRY
21 COPY TYPE, SIZE, IDIM AND DSPIT FROM TOC ENTRY
22 ELSE
23 SET TYPE, SIZE AND IDIM TO ZERO
24 ENDIF
25 ENDIF
26 ENDIF
27 END XPATR
FORTRAN CALLING PROCEDURE

CALL XPGET (LU, INBUF, INTLVG, RBUFF, N, INUMS, IN(1), ...
IN(IN(N))

XGET ALLOWED PROCESSORS TO OBTAIN DATA FROM THE DDIS. IT
VERIFIES THE EXISTENCE OF THE DATA ELEMENTS AND ORDER'S
REFERENCE OF THE INTERFACE TABLE AND RETRIEVES INPUT DATA
BY ONE OF THE FOLLOWING WAYS:
1) REQUESTED DATA FROM LITERAL DATA STORED WITHIN THE
INTERFACE TABLE IS RETURNED OR
2) REQUESTED DATA FROM A DATA ELEMENT REFERENCED BY
THE INTERFACE TABLE IS RETURNED OR
3) THE NAME OF THE 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.
4) DATA ELEMENT AND CODE NAMES ARE SEARCHED FOR IN THE DDA TOC,
IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING TERMINATED.

---

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL
INBUF - INPUT/OUTPUT BUFFER OF THE PARAMETERS + 1) WORDS.
INTLVG - ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION.
INTLV - LENGTH OF INBUF
RBUFF - MANAGER REQUEST BUFFER (4 WORDS) USED TO COMMUNICATE
WITH THE MANAGER. BUFFER MAY BE USED AS PROGRAM SCRATCH
AREA BUT NOT ACROSS PROCESSOR SERVICE CALLS.
N - NUMBER OF PARAMETERS TO BE RETRIEVED. IF N=0, ALL
PARAMETERS ARE REQUESTED AND INUMS IS NOT USED.
INUMS - ARRAY (N WORDS) OF RELATIVE NUMBERS OF PARAMETERS IN THE
INTERFACE TABLE TO BE RETRIEVED. THE ORDER OF THE
NUMBERS INDICATES THE STORAGE ORDER FOR THE IN(1)

OUTPUT

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

---

INTERNAL VARIABLES

AMULT - ARRAY OF 30 BYTES CONTAINING MULTIPLIERS FOR USE IN THE
PROCEDURE SUMSCP IN DETERMINING WORD DISPLACEMENTS FOR
VARIOUS TYPES OF SUBSCRIPTED ARRAYS
CLSTL - ADDRESS OF THE CLASS FIELD (SECOND WORD) OF THE REQUEST
ARRAY RETR.
STPT - FLAG INDICATING GET OR PUT PROCESSING. VALUES CORRESPOND
TO BIT SETTINGS FOR INPUT OR OUTPUT FLAGS IN THE FIFTH
WORD OF INTERFACE TABLE ENTRIES
COUNTER OF THE NUMBER OF PASSES THROUGH THE PARAMETER
PROCESSING LOOP

VALUE OF "P", WHERE P IS THE NUMBER OF PARAMETERS IN THE INTERFACE TABLE

NAME FIELD (THIRD WORD) OF THE REQUEST

ARRAY REQUEST

NEGATIVE OF THE NUMBER OF IN/OUT ARRAYS IN CALLING SEQUENCE

EIGHT WORD ARRAY USED FOR CONSTRUCTING ANA MANAGEMENT REQUESTS FOR XPRE

FILE IDENTIFIER CHARACTER (Hимв3))

EXTERNAL VARIABLES (SEE XPRE)

XPRCL

XPRU

XPRQ

**********

NOTES

USES .ENTRY, EXEC, XPRE, XPIT, XVSTB

XPGET MUST BE INCLUDED IN PROCESSOR AT FDS BUILD TIME.

XPME INTRUFS MUST BE USED BY XPGET, XPUT, AND XPATH AND XCEOS TO BE INITIALIZED ONLY ONCE BY ANY OF THE THREE ROUTINES.

SINCE REQUESTS FOR INPUT DATA FROM THE ANA MAY BE MADE FOR UP TO EIGHT AT A TIME, XPGET RUNS MOST EFFICIENTLY WHEN PARAMETERS ARE REQUESTED IN MULTIPLES OF EIGHT.

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

CALL XPPUT (LU, INTBUF, INTLIB, NBUF, N, IMUXS, OUT(1), ...,
OUT(N))

1  *00
2  *00
3  *00
4  *00
5  *00
6  *00
7  *00
8  *00
9  *00
10  *00
11  *00
12  *00
13  *00
14  *00
15  *00
16  *00
17  *00
18  *00
19  *00
20  *00
21  *00
22  *00
23  *00
24  *00
25  *00
26  *00
27  *00
28  *00
29  *00
30  *00
31  *00
32  *00
33  *00
34  *00
35  *00
36  *00
37  *00
38  *00
39  *00
40  *00
41  *00

XPPUT ALLOWS PROCESSORS TO STORE DATA INTO DATA ELEMENTS AND
SHARE'S REFERENCED IN THE INTERFACE TABLE AS FOLLOWS:
1) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC.
2) IF FOUND, DATA IS STORED
3) IF NOT FOUND, A MESSAGE IS ISSUED AND PROCESSING
TERMINATES.
2) DATA ELEMENT NAME IS SEARCHED FOR IN THE TOC. IF THE NAME IS
NOT FOUND, IT IS ADDED TO THE TOC. IF THE NAME IS
FOUND OR AFTER BEING ADDED TO THE TOC, THE NAME IS
PREPARED WITH A / SYMBOL, SUFFIXED WITH A ONE
201  *01
202  *01
203  *01
204  *01
205  *01
206  *01
207  *01
208  *01
209  *01
210  *01
211  *01
212  *01
213  *01
214  *01
215  *01
216  *01
217  *01
218  *01
219  *01
220  *01
221  *01
222  *01
223  *01
224  *01
225  *01
226  *01
227  *01
228  *01
229  *01
230  *01
231  *01
232  *01
233  *01
234  *01
235  *01
236  *01
237  *01
238  *01
239  *01
240  *01
241  *01

INPUT
1  *02
2  *02
3  *02
4  *02
5  *02
6  *02
7  *02
8  *02
9  *02
10  *02
11  *02
12  *02
13  *02
14  *02
15  *02
16  *02
17  *02
18  *02
19  *02
20  *02
21  *02
22  *02
23  *02
24  *02
25  *02
26  *02
27  *02
28  *02
29  *02
30  *02
31  *02
32  *02
33  *02
34  *02
35  *02
36  *02
37  *02
38  *02
39  *02
40  *02

LENGTH OF INTBUF
1  *02
2  *02
3  *02
4  *02
5  *02
6  *02
7  *02
8  *02
9  *02
10  *02
11  *02
12  *02
13  *02
14  *02
15  *02
16  *02
17  *02
18  *02
19  *02
20  *02
21  *02
22  *02
23  *02
24  *02
25  *02
26  *02
27  *02
28  *02
29  *02
30  *02
31  *02
32  *02
33  *02
34  *02
35  *02
36  *02
37  *02
38  *02
39  *02
40  *02

FLAG INDICATING XPPUT/XPGF NORMAL PROCESSING (-1) OR
XPITI/XPPITI SPECIAL PROCESSING BY-PASSING SUBSCRIPT
RESOLUTION (-2)

OUTPUT
1  *03
2  *03
3  *03
4  *03
5  *03
6  *03
7  *03
8  *03
9  *03
10  *03
11  *03
12  *03
13  *03
14  *03
15  *03
16  *03
17  *03
18  *03
19  *03
20  *03
21  *03
22  *03
23  *03
24  *03
25  *03
26  *03
27  *03
28  *03
29  *03
30  *03
31  *03
32  *03
33  *03
34  *03
35  *03
36  *03
37  *03
38  *03
39  *03
40  *03

INTERNAL VARIABLES
1  *04
2  *04
3  *04
4  *04
5  *04
6  *04
7  *04
8  *04
9  *04
10  *04
11  *04
12  *04
13  *04
14  *04
15  *04
16  *04
17  *04
18  *04
19  *04
20  *04
21  *04
22  *04
23  *04
24  *04
25  *04
26  *04
27  *04
28  *04
29  *04
30  *04
31  *04
32  *04
33  *04
34  *04
35  *04
36  *04
37  *04
38  *04
39  *04
40  *04

(SEE XPGF)
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>243</td>
<td>BEGIN XPGET</td>
</tr>
<tr>
<td>245</td>
<td>SET FOR 'GET'</td>
</tr>
<tr>
<td>246</td>
<td>PERFORM ACCESS TO RETRIEVE DATA</td>
</tr>
<tr>
<td>247</td>
<td>END XPGET</td>
</tr>
<tr>
<td>248</td>
<td>BEGIN XPUT</td>
</tr>
<tr>
<td>249</td>
<td>SET FOR 'PUT'</td>
</tr>
<tr>
<td>250</td>
<td>PERFORM ACCESS TO STORE DATA</td>
</tr>
<tr>
<td>251</td>
<td>END XPUT</td>
</tr>
<tr>
<td>252</td>
<td>BEGIN ACCESS XPINI TO INITIALIZE GLOBALS AND INTERFACE TABLE</td>
</tr>
<tr>
<td>253</td>
<td>DO FOR EACH PARAMETER REQUESTED</td>
</tr>
<tr>
<td>254</td>
<td>IF SELECTED PARAMETER IS OUT OF RANGE</td>
</tr>
<tr>
<td>255</td>
<td>CALL XPRED TO PURGE QUEUED REQUESTS</td>
</tr>
<tr>
<td>256</td>
<td>EXIT TO XPE13:</td>
</tr>
<tr>
<td>257</td>
<td>ENSIF</td>
</tr>
<tr>
<td>258</td>
<td>IF INPUT/OUTPUT TYPE DOES NOT MATCH 'GET'/'PUT' PROCESSING</td>
</tr>
<tr>
<td>259</td>
<td>THEN</td>
</tr>
<tr>
<td>260</td>
<td>CALL XPRED TO PURGE QUEUED REQUESTS</td>
</tr>
<tr>
<td>261</td>
<td>EXIT TO XPE12:</td>
</tr>
<tr>
<td>262</td>
<td>ENSIF</td>
</tr>
<tr>
<td>263</td>
<td>IF OVER RUNNING CALLING SEQUENCE</td>
</tr>
<tr>
<td>264</td>
<td>THEN</td>
</tr>
<tr>
<td>265</td>
<td>CALL XPRED TO PURGE QUEUED REQUESTS</td>
</tr>
<tr>
<td>266</td>
<td>EXIT TO XPE13:</td>
</tr>
<tr>
<td>267</td>
<td>ENSIF</td>
</tr>
<tr>
<td>268</td>
<td>IF PROCESSING FOR &quot;GET&quot;</td>
</tr>
<tr>
<td>269</td>
<td>THEN</td>
</tr>
<tr>
<td>270</td>
<td>IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)</td>
</tr>
<tr>
<td>271</td>
<td>THEN</td>
</tr>
<tr>
<td>272</td>
<td>IF INPUT IN LITERAL FORM</td>
</tr>
<tr>
<td>273</td>
<td>THEN</td>
</tr>
<tr>
<td>274</td>
<td>BUILD REQUEST WITH INTERFACE TABLE NAME AND DISPLACEMENT</td>
</tr>
<tr>
<td>275</td>
<td>ELSE</td>
</tr>
<tr>
<td>276</td>
<td>IF NORMAL XPGET/PUT PROCESSING ('XPGPF' = -1)</td>
</tr>
<tr>
<td>277</td>
<td>THEN</td>
</tr>
<tr>
<td>278</td>
<td>IF SUBSCRIPTED (DISPLACEMENT &gt; 0 OR DOUBLE SUBSCRIPT FLAG SET)</td>
</tr>
<tr>
<td>279</td>
<td>THEN</td>
</tr>
<tr>
<td>280</td>
<td>PERFORM XPSEC TO COMPUTE DISPLACEMENT = (DIM, SUBS, TYPE)</td>
</tr>
<tr>
<td>281</td>
<td>ELSE</td>
</tr>
<tr>
<td>282</td>
<td>DISPLACEMENT IS ZERO</td>
</tr>
<tr>
<td>283</td>
<td>ENDIF</td>
</tr>
<tr>
<td>284</td>
<td>ELSE</td>
</tr>
<tr>
<td>285</td>
<td>USE DISPLACEMENT FROM INTERFACE TABLE ENTRY</td>
</tr>
<tr>
<td>286</td>
<td>ENDIF</td>
</tr>
<tr>
<td>287</td>
<td>BUILD REQUEST WITH DE NAME AND DISPLACEMENT</td>
</tr>
<tr>
<td>288</td>
<td>ENSIF</td>
</tr>
<tr>
<td>289</td>
<td>CALL XPRED TO QUEUE RETRIEVAL AND STORAGE OF INPUT</td>
</tr>
<tr>
<td>290</td>
<td>ELSE PARAMETER IS DROE (CLASS 3)</td>
</tr>
<tr>
<td>291</td>
<td>THEN</td>
</tr>
<tr>
<td>292</td>
<td>BUILD REQUEST FOR TOC ENTRY</td>
</tr>
<tr>
<td>293</td>
<td>CALL XPRED TO IMMEDIATELY RETRIEVE TOC ENTRY</td>
</tr>
<tr>
<td>294</td>
<td>PERFORM XPFLM TO CONSTRUCT AND STORE QUALIFIED FILE NAME</td>
</tr>
<tr>
<td>295</td>
<td>&quot;STORE FILE ATTRIBUTES&quot;</td>
</tr>
<tr>
<td>296</td>
<td>ENSIF</td>
</tr>
<tr>
<td>297</td>
<td>ELSE PROCESSING FOR &quot;PUT&quot;</td>
</tr>
<tr>
<td>298</td>
<td>THEN</td>
</tr>
<tr>
<td>299</td>
<td>IF PARAMETER IS MEMORY RESIDENT DE (CLASS 2)</td>
</tr>
<tr>
<td>300</td>
<td>THEN</td>
</tr>
<tr>
<td>301</td>
<td>IF NORMAL XPGET/PUT PROCESSING ('XPGPF' = -1)</td>
</tr>
</tbody>
</table>
IF SUBSCRIBED
THEN
PERFORM XPSBC TO COMPUTE DISPLACEMENT = F(DIM, SUBS, TYPE)
ELSE
DISPLACEMENT IS ZERO
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD REQUEST TO REALLOCATE DRDE
CALL XPREQ TO QUEUE ALLOCATION
END IF
ELSE
USE DISPLACEMENT FROM INTERFACE TABLE ENTRY
BUILD REQUEST TO OUTPUT DATA TO CLASS I/O AND STORE DATA IN AHA
CALL XPREQ TO QUEUE STORAGE OF DATA
ELSE PARAMETER IS DRDE (CLASS 3)
BUILD REQUEST TO DELETE ANY EXISTING DRDE WITH THIS NAME
CALL XPREQ TO QUEUE DELETION
BUILD RF IST TO REALLOCATE DRDE WITH NEW ATTRIBUTES
EXIT TO XERR14 IF FILE TYPE NOT 1-13, # BLOCKS < 1 OR MAX REC SIZE NOT 1-1200
BUILD RF IST TO QUEUE REALLOCATION OF DRDE
END IF
ENDIF
ENDDO
CALL XPREQ TO COMPLETE QUEUED REQUESTS
EXIT ACCESS
:ERR12: TERMINATE PROCESSOR FOR INPUT/OUTPUT TYPE INCONSISTENCY
:XPERR: TERMINATE PROCESSOR FOR INVALID PARAMETER REQUEST
:ERR14: CALL XPREQ TO PURGE QUEUED REQUESTS
:ERR15: TERMINATE PROCESSOR FOR INVALID DRDE FILE TYPE, BLOCK COUNT OR MAX RECORD SIZE
BEGIN XPINI
1.3 INITIALIZ GLOBAL VALUES FROM LU AND XVSTB
1.6 TERMINATE PROCESSOR WITH XP10 ERROR IF LU NOT IN XVSTB
1.3 IF INTERFACE TABLE BUFFER NOT INITIALIZED
1.4 THEN
1.5 RETRIEVE INTERFACE TABLE FROM MANAGER CLASS I/O NUMBER
1.6 IF RETRIEVAL NOT SUCCESSFUL
1.7 THEN
1.8 TERMINATE PROCESSOR WITH 'XP10 PROCESSOR INITIALIZATION ERROR'
1.9 ENDIF
1.10 ENDIF
1.11 EXIT TO XPE13: IF M $ 0
1.12 END XPINI
1.13

BEGIN XPIN
1.14 SET / IN FIRST CHARACTER POSITION
1.15 MOVE FOUR WORD NAME INTO MIDDLE POSITIONS
1.16 LOCATE FIRST BLANK CHARACTER
1.17 REPLACE BLANK WITH USER ID
1.18 END XPIN
1.19

BEGIN XPBC
1.22 IF DOUBLE SUBSCRIPTED
1.23 THEN
1.24 BUILD REQUEST FOR SUBSCRIPTS STORED IN LITERAL BLOCK
1.25 CALL XPER TO QUEUE RETURN OF SUBSCRIPTS
1.26 ELSE
1.27 SINGLE SUBSCRIPT IS CONTAINED IN DISPLACEMENT FIELD & J-SUBSCRIPT IS $ 1
1.28 ENDIF
1.29 BUILD REQUEST TO RETURN TOC ENTRY
1.30 CALL XPER TO IMMEDIATELY RETRIEVE TOC ENTRY
1.31 IF DOUBLE DIMENSIONED (IDIM $ 0)
1.32 THEN
1.33 DISPLACEMENT = (IDIM*(JSUB-1) + JSUB-1) * # WORDS PER ELEMENT
1.34 ELSE
1.35 DISPLACEMENT = SIZE*(JSUB-1) + (JSUB-1) * # WORDS PER ELEMENT
1.36 ENDIF
1.37 END XPBC
1.38 1 END ACCESS
FORTRAN CALLING PROCEDURE

CALL XPGET (LU, INBUF, INLNG, MBUFF, INUM, IN, SIZE, DISP)

*********

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

*********

INPUT

LU - LOGICAL UNIT NUMBER OF USER TERMINAL

INBUF - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 1) WORDS, ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON FIRST USE TO CAUSE INITIALIZATION

INLNG - LENGTH OF INBUF

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

INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE FROM WHICH TO RETURN DATA

SIZE - TOTAL NUMBER OF WORDS TO BE RETURNED

DISP - DISPLACEMENT FROM THE BEGINNING OF THE INPUT PARAMETER AT WHICH TO BEGIN DATA RETRIEVAL. A VALUE OF ZERO INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT BE SPECIFIED. I.E., RETRIEVAL FROM AN AREA PREVIOUS TO THE USERS SUBSCRIPTS IS NOT SUPPORTED

*********

OUTPUT

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

*********

INTERNAL

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

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

JGTP - TWO WORD VECTOR CONTAINING 'JSB XPGET' OR 'JSB XPPUT' INSTRUCTIONS. USED TO DYNAMICALLY PRODUCE XPGET AND XPPUT CALLS

*********

USES EXEC, .ENTR, XPAR, XPE13(XPATR), XPGET(XPATR), XPPUT(XPATR), XPERR(XPATR), XPE13(XPATR), XPGET(XPATR), XPAR, .ENTR(XPATR), XPPUT(XPATR)
FORTRAN CALLING PROCEDURE

CALL XPPTI (LU, INBUF, INTLG, MBUFF, INUM, OUT, SIZE, DISP)

********

XPPTI 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

INTBUF - INPUT/OUTPUT BUFFER OF 7*(# PARAMETERS + 1) WORDS,
ALLOCATED WITHIN THE CALLING PROGRAM TO HOLD THE
INTERFACE TABLE HEADER. FIRST WORD MUST BE ZERO ONLY ON
FIRST USE TO CAUSE INITIALIZATION

INTLG - 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 ACROSS PROCESSOR SERVICE CALLS

INUM - RELATIVE NUMBER OF PARAMETER IN INTERFACE TABLE INTO
WHICH TO STORE DATA

OUT - ARRAY OF AT LEAST SIZE WORDS FROM WHICH OUTPUT DATA IS TO
BE TAKEN

SIZE - TOTAL NUMBER OF WORDS TO BE STORED

DISP - DISPLACEMENT FROM THE BEGINNING OF THE OUTPUT PARAMETER
AT WHICH TO BEGIN DATA STORAGE. A VALUE OF ZERO
INDICATES THE BEGINNING OF THE AREA SPECIFIED BY THE USER
IN THE INTERFACE TABLE. NEGATIVE VALUES OF DISP MUST NOT
BE SPECIFIED, I.E., STORAGE INTO AN AREA PREVIOUS TO THE
USERS SUBSCRIPTS IS NOT SUPPORTED.

********

OUT

NONE

********

INTERNAL

SEE XPPTI

********

NOTES

XPPTI IS AN ENTRY POINT INTO XPPTI

********
BEGIN XPGTI
SET FOR GET PROCESSING
PERFORM XPXII TO RETRIEVE DATA
END XPGTI
BEGIN XPPTI
SET FOR PUT PROCESSING
PERFORM XPXII TO STORE DATA
END XPPTI
BEGIN XPXII
SET XPSF(XPSET) TO BY-PASS NORMAL XPGT/XPPUT SUBSCRIPT PROCESSING
CALL XPATR TO ASSURE INITIALIZATION OF XINTF AND RETURN BASE DISPLACEMENT
EXIT TO :XPE13: (XPATH) IF INDICATED PARAMETER IS A DROG FILE
SAVE INTERFACE TABLE ENTRY CLASS/TYPE WORD, FLAG/DISP WORD AND SIZE WORD
IF LITERAL (NAME = 0)
THEN
EXIT TO :ERR15: IF DISP + SIZE > SAVED SIZE
RESET BASE DISPLACEMENT TO ACTUAL DISP FIELD IN INTERFACE TABLE ENTRY
ELSE
INCREMENT BASE DISPLACEMENT TO CONVERT TO SUBSCRIPT
SET TYPE FIELD TO FREE
ENDIF
CLEAR ENTRY SUBSCRIPT BIT
SET DISP FIELD TO SUM OF BASE DISPLACEMENT AND DISP
SET SIZE FIELD TO SIZE
CALL XPGT/XPPUT TO TRANSFER DATA
RESTORE ORIGINAL INTERFACE TABLE ENTRY
RESTORE XPGP(XPSET) TO NOMINAL VALUE
EXIT XPXII
:ERR15: TERMINATE PROCESSOR FOR ATTEMPT TO RETRIEVE TOO MUCH DATA
END XPXII
CALLING PROCEDURE

JSR XPREQ
DEF *H+1 WHERE H IS THE NUMBER OF ACTUAL ARGUMENTS
DEF OPTN REQUIRED OPTION
DEF RST REQUIRED REQUEST
DEF ADRES OPTIONAL ADDRESS (SEE OPTN)

******
** XPGET/XPPUT BUFFERED AVA MANAGEMENT SERVICE **
** ******
** IM-UT **
** ******
** OPTN - OPTION WORD **
** SIGN BIT - 0 QUEUE REQUEST AND RETURN **
** 1 QUEUE REQUEST AND CLOSE BUFFER **
** RIGHT BIT - 0 QUEUE REQUEST AND TRANSFER DATA TO/FROM **
** ADRES **
** 1 QUEUE REQUEST ONLY **
** RST - EIGHT WORD REQUEST (SEE SDD 6.2.6.2) **
** ADRES - STORAGE AREA TO RECEIVE RETURNED VALUES OR SUPPLY OUTPUT **
** VALUES USED ONLY WHEN OPTN(RIGHT BIT) = 0 **
** ******
** EXTERNAL SYMBOLS **
** XPLU - TERMINAL LOGICAL UNIT NUMBER **
** XPREQ - ADDRESS OF 64 WORD MANAGER REQUEST BUFFER **
** XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS **
** ******
** OUTPUT **
** IF A REQUEST FAILS, A MESSAGE IS ISSUED AND PROCESSING IS TERMINATED, OTHERWISE INCOMING DATA WILL BE STORED IN THE ADDRESS(es) SUPPLIED. **
** ******
** LOCAL VARIABLES **
** ADDS - ADDRESS OF STORAGE AREA ADDRESS TABLE (ADDS) **
** ADDS - TABLE OF ADDRESSES FOR STORAGE OF FETCHED DATA **
** APRMS - ADDRESS OF PARMS AREA **
** CLASS - CLASS NUMBER USED TO TRANSMIT DATA TO MANAGER **
** PARAMETER RETURN AND SCRATCH AREA **
** PTR - SAVE AREA FOR ADDS AND XPREQ POINTER **
** ******
** NOTES **
** USES .ENTR, EXEC, XPIIT, XPAW **
** IF DATA IS TO BE TRANSFERRED TO/FROM 'ADRES', THEN 'RST' 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 (8)
6 THEN
7 STORE ADDRESS IN TABLE
8 ELSE SHOULD BE A REQUEST TO STORE DATA (7)
9 OUTPUT DATA TO CLASS I/O
10 STORE CLASS NUMBER IN REQUEST WORD &
11 ENDIF
12 EMBDF
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 AWA MANAGEMENT
19 RETRIEVE RETURN PARAMETERS
20 IF REWRITE FLAG SET (PARM5)
21 THEN
22 RETRIEVE REQUEST BUFFER
23 ENDIF
24 IF REQUESTS WERE SUCCESSFUL
25 THEN
26 DO FOR EACH REQUEST IN BUFFER
27 IF REQUEST TO RETURN DATA (8, 9 OR 16)
28 THEN
29 RETRIEVE AND STORE DATA IN ADDRESS CONTAINED IN TABLE
30 ENDIF
31 EMBDO
32 CLEAR POINTER AND LOCAL CLASS NUMBER
33 ELSE
34 OUTPUT FAILURE MESSAGE (XP11)
35 DO FOR EACH REQUEST IN BUFFER
36 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
37 IF REQUEST SUCCESSFUL FOR DATA RETRIEVAL OR UNSUCCESSFUL STORE
38 THEN
39 FREE CLASS I/O NUMBER AND SAM BUFFER
40 END
41 ENDIF
42 EMBDO
43 EXIT PROCESSOR WITH REQUEST FOR SEQUENCE TERMINATION
44 ENDIF
45 ENDIF
46 THEN
47 END XPREG
FORTRAN CALLING PROCEDURE FOR PROCESSOR TC SPECIFIC TYPE

CALL XPDRS (LU,PRMLEN,PROMPT,TYPE,DATLEN,IDIM,DATA,RETC)

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

INPUTS FROM CALLING SEQUENCE:
- LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
- PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER STRING USED FOR THE USER PROMPT
- PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING USED AS THE USER PROMPT
- TYPE - (INTEGER, 1 WORD) CODE FOR THE DATA TYPE EXPECTED AS THE USER'S RESPONSE AS FOLLOWS:
  0 - FREE
  1 - INTEGER
  2 - REAL
  3 - DOUBLE PRECISION
  4 - CHARACTER STRING LENGTH 2
  5 - CHARACTER STRING LENGTH 4
  6 - CHARACTER STRING LENGTH 8
  7 - CHARACTER STRING LENGTH 16
  8 - CHARACTER STRING LENGTH 32
  9 - CHARACTER STRING LENGTH 64
  10 - CHARACTER STRING LENGTH 128
  11 - CHARACTER STRING LENGTH 256
  12 - CHARACTER STRING LENGTH 512
  13 - CHARACTER STRING LENGTH 1024
- DATLEN - (INTEGER, 1 WORD) NUMBER OF LOGICAL ELEMENTS IN THE DATA AREA. IF TYPE IS FREE, DATLEN IS THE NUMBER OF WORDS.
- IDIM - (INTEGER, 1 WORD) THE COLUMN LENGTH OF THE DATA AREA IF IT IS A 2 DIMENSION ARRAY, ELSE 1 OR 0 IF IT IS A VECTOR.

OUTPUTS FROM CALLING SEQUENCE:
- DATA - (INTEGER, DIM DEPENDS ON DATLEN AND TYPE) AREA TO CONTAIN THE USERS RESPONSE
- RETCS - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO CALLER:
  0 - NORMAL RETURN, DATA AREA CONTAINS USER'S RESPONSE
  1 - USER ENTERED A CHAR. THERE IS NO RESPONSE.
  2 - USER ENTERED A CHAR. THERE IS NO RESPONSE.
  3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS 34 CHARACTERS OR 17 WORDS.
  4 - PARAMETER LIST IS INVALID. EITHER:
    A. IDIM LESS THAN ZERO
    B. DATLEN LESS THAN ONE
    C. TYPE NOT VALID. SPECIFIED.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>667</td>
<td>DATPTR - INDEX IN WORDS INTO DATA ARRAY WHERE NEXT</td>
</tr>
<tr>
<td>668</td>
<td>Element is to be stored</td>
</tr>
<tr>
<td>669</td>
<td>EFFTP - THE EFFECTIVE INDEX (TYPE+1) OF THE CHARACTER</td>
</tr>
<tr>
<td>670</td>
<td>STRING RESPOND, IF TYPE IS FREE, IT IS THE</td>
</tr>
<tr>
<td>671</td>
<td>CLOSEST SUPPORTED LENGTH, ELSE IT IS THE LENGTH</td>
</tr>
<tr>
<td>672</td>
<td>SPECIFIED BY TYPE.</td>
</tr>
<tr>
<td>673</td>
<td>IND - CONTAINS THE INDEX INTO MSGS ARRAY OF THE CURRENT</td>
</tr>
<tr>
<td>674</td>
<td>ERROR MESSAGE.</td>
</tr>
<tr>
<td>675</td>
<td>LEN - LENGTH OF CURRENT ERROR MESSAGE.</td>
</tr>
<tr>
<td>676</td>
<td>LENGTH - 9 WORD ARRAY REPRESENTING THE LENGTH IN WORDS</td>
</tr>
<tr>
<td>677</td>
<td>OF THE 9 DATA TYPES 0-8 RESPECTIVELY</td>
</tr>
<tr>
<td>678</td>
<td>STKIND - COUNT OF NUMBER OF NESTED REPEATS AND INDEX</td>
</tr>
<tr>
<td>679</td>
<td>TO THE CURRENT TOP OF THE STACK</td>
</tr>
<tr>
<td>680</td>
<td>STKPM- - A 9 WORD ARRAY FOR STACK OF PARENTHESES FLAGS</td>
</tr>
<tr>
<td>681</td>
<td>STKREP - A 9 WORD ARRAY FOR STACK OF REPEAT COUNTS</td>
</tr>
<tr>
<td>682</td>
<td>STKTOK - A 9 WORD ARRAY FOR STACK OF TOKEN POINTERS</td>
</tr>
<tr>
<td>683</td>
<td>WHERE EACH REPEAT GROUP BEGINS</td>
</tr>
<tr>
<td>684</td>
<td>TOKPTR - POINTER TO CURRENT TOKEN BEING PROCESSED</td>
</tr>
<tr>
<td>685</td>
<td>TYPE1 - INDEX 1-9 INTO LENGTH ARRAY BASED ON 0-8 TYPE</td>
</tr>
<tr>
<td>686</td>
<td>TYPE1 = TYPE +1</td>
</tr>
<tr>
<td>687</td>
<td>*********</td>
</tr>
<tr>
<td>688</td>
<td>*********</td>
</tr>
<tr>
<td>689</td>
<td>10C5</td>
</tr>
<tr>
<td>690</td>
<td>10D5</td>
</tr>
<tr>
<td>691</td>
<td>SUBROUTINES AND FUNCTIONS CALLED:</td>
</tr>
<tr>
<td>692</td>
<td>EXEC, XPRDS, XPRDS</td>
</tr>
<tr>
<td>693</td>
<td>*********</td>
</tr>
<tr>
<td>694</td>
<td>10C5</td>
</tr>
<tr>
<td>695</td>
<td>10C5</td>
</tr>
<tr>
<td>696</td>
<td>POL ROUTINES INCLUDED:</td>
</tr>
<tr>
<td>697</td>
<td>XPRDS, STRING, SUBSCR, REPET, ERRMSG</td>
</tr>
<tr>
<td>698</td>
<td>XPRDS</td>
</tr>
<tr>
<td>699</td>
<td>XPRDS</td>
</tr>
</tbody>
</table>
1 BEGIN XPRODS
2 IF TYPE, DATA LENGTH, AND I DIMENSION ARE VALID THEN
3 CALL XPRM TO PROMPT USER AND DECIDE RESPONSE
4 IF XPRM RETURN CODE IS ZERO THEN
5 DO UNTIL EOS TOKEN IS FOUND
6 SET MESSAGE NUMBER TO XPOZ
7 CASE Token (:NUMBER::NUMBER::NUMBER::STRING::NULL::)
8 :SUBSCR::REPET:
9 
10 :NUMBER:
11 ERREXIT IF TYPE DOES NOT MATCH OR IS NOT FREE PERFORM ERRMSG
12 SET MESSAGE NUMBER TO XPOZ
13 ERREXIT IF THERE IS NO ROOM IN DATA AREA PERFORM ERRMSG
14 SET PREVIOUS TOKEN TO DATA
15 CALL XMOV TO MOVE DATA INTO DATA AREA
16 INCREMENT POINTER IN DATA AREA
17 INCREMENT POINTER TO NEXT TOKEN
18 
19 :STRING:
20 PERFORM STRING
21 :NULL:
22 SET MESSAGE NUMBER TO XPOZ
23 ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN PERFORM ERRMSG
24 SET PREVIOUS Token TO DATA
25 INCREMENT POINTER IN DATA AREA
26 INCREMENT POINTER TO NEXT TOKEN
27 :SUBSCR:
28 PERFORM SUBSCR
29 :REPET:
30 PERFORM REPET
31 ENCASE
32 DO UNTIL TOKEN IS NOT A CLOSED PARENTHESIS
33 IF STACK IS NOT EMPTY AND
34 (PREVIOUS Token IS DATA AND TOP OF STACK PAREN FLAG =0) OR
35 (TOKEN IS CLOSE PAREN AND TOP OF STACK PAREN FLAG =1) THEN
36 DECREMENT TOP OF STACK REPEAT COUNT BY 1
37 IF TOP OF STACK REPEAT COUNT > 0 THEN
38 RESET Token POINTER TO TOP OF STACK INDEX
39 ELSE
40 POP TOP ENTRY ON STACK
41 IF TOKEN IS A CLOSED PAREN THEN
42 INCREMENT TO NEXT TOKEN
43 ENDIF
44 ENDOF
45 ENDIF
46 ENDOF
47 ENDIF
48 SET XPRMS RETURN CODE = XPRM RETURN CODE
49 EXIT
50 SET XPRMS RETURN CODE TO SAY INVALID PARAMETER 135T
51 END XPRODS
BEGIN STRING
ERROR: TYPE IS NOT CHARACTER OR FREE PERFORM ERRMSG
ERROR: DETERMINE EFFECTIVE LENGTH OF RESPONSE AS NEXT LARGER SUPPORTED LENGTH
ERROR: IF TYPE IS NOT FREE THEN
ERROR: SET EFFECTIVE LENGTH = LENGTH REQUESTED
END IF
SET MESSAGE NUMBER TO XPOS
ERROR: IF THERE IS NO ROOM IN DATA AREA FOR THIS ELEMENT PERFORM ERRMSG
CALL XNRYZ TO MOVE BLANKS INTO DATA AREA FOR EFFECTIVE LENGTH
CALL XNRYZ TO MOVE CHARACTER STRING INTO DATA AREA FOR REAL LENGTH
SET PREVIOUS TOKEN IS DATA
INCREMENT POINTER IN DATA AREA
INCREMENT TO NEXT TOKEN
END STRING

BEGIN SUBSCR
INCREMENT POINTER TO NEXT TOKEN
SET MESSAGE NUMBER TO XPOS
ERROR: IF TOKEN IS NOT AN INTEGER TO PERFORM ERRMSG
IF I-DIMENSION > 1 THEN
SET 1 TO INTEGER VALUE
INCREMENT POINTER TO NEXT TOKEN
ERROR: IF TOKEN IS NOT AN INTEGER OR
ERROR: IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
SET MESSAGE NUMBER TO XPOS
ERROR: IF INTEGER > I-DIMENSION TO PERFORM ERRMSG
ERROR: IF SUBSCRIPT IS OUT OF RANGE TO PERFORM ERRMSG
ELSE
ERROR: IF NEXT TOKEN IS NOT A CLOSE PAREN TO PERFORM ERRMSG
SET MESSAGE NUMBER TO XPOS
ERROR: IF SUBSCRIPT IS OUT OF RANGE
END IF
ADJUST INDEX INTO DATA AREA ACCORDING TO SUBSCRIPT
INCREMENT POINTER BY 3 TOCKES
SET PREVIOUS TOKEN = SUBSCRIPT
SET MESSAGE NUMBER TO XPOS
ERROR: IF TOKEN IS AN EOS OR
ERROR: IF TOKEN IS A REPEAT OR
ERROR: IF TOKEN IS A CLOSE PAREN OR
ERROR: IF TOKEN IS A SUBSCRIPT TO PERFORM ERRMSG
END SUBSCR
END
799 1 BEGIN REPET
800 2 SET MESSAGE NUMBER TO XPOS
801 2 ERROR IF TOKEN IS AN EOS TO PERFORM ERRMSG
802 2 INCREMENT STACK POINTER
803 2 SET MESSAGE NUMBER TO XPOS
804 2 ERROR IF THERE ARE TOO MANY NESTED REPEATS PERFORM ERRMSG
805 2 PUSH REPEAT COUNT ON STACK
806 2 SET PARENTHESES FLAG TO 0
807 2 IF TOKEN IS AN OPEN PARENTHESES THEN
808 3 INCRTMENT POINTER TO NEXT TOKEN
809 3 SET PARENTHESES FLAG TO 1
810 2 ENDIF
811 2 PUSH TOKEN INDEX AND PAREN FLAG ON STACK
812 2 SET PREVIOUS TOKEN IS A REPEAT
813 1 END REPET
814 1 *
815 1 *
816 1 *
817 1 BEGIN ERRMSG
818 2 CALL EXEC TO WRITE ERROR MESSAGE
819 2 PERFORM XPRDS - NO RETURN
820 1 END ERRMSG
CD**************
CD CALLING PROCEDURE FOR PROCESSOR TC MIXED TYPE
CD
CD CALL XPRM (LU,PRMLEN,PROMPT,COMLEN,COMBUF,RETC)
CD
CD**************
CD1 WRITES "PRMLEN" WORDS OF "PROMPT" TO USER READS THE USER'S
CD1 RESPONSE; CONVERTS RESPONSE TO TOKENS IN "COMBU" AND PASSES
CD1 BACK A RETURN CODE "RETC" INDICATING THE SUCCESS OF XPRM
CD1 AND THE USEFULNESS OF "COMBUF".
CD**************
CD2 INPUTS FROM CALLING SEQUENCE:
CD2
CD2 LU - (INTEGER, 1 WORD) LOGICAL UNIT OF USER'S TERMINAL
CD2 PRMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE CHARACTER
CD2 PROMPT - (INTEGER, PRMLEN WORDS) IS THE CHARACTER STRING
CD2 COMLEN - (INTEGER, 1 WORD) LENGTH IN WORDS OF THE
CD2 COMMUNICATIONS BUFFER (COMBU)
CD2
CD2**************
CD3 OUTPUTS FROM CALLING SEQUENCE:
CD3
CD3 COMBUF - (INTEGER, COMLEN WORDS) ARRAY TO CONTAIN THE
CD3 ENCODED USER RESPONSE
CD3 RETC - (INTEGER, 1 WORD) RETURN CODE PASSED BACK TO THE
CD3 CALLING PROGRAM AS FOLLOWS:
CD3 0 - NORMAL RETURN BUFFER CONTAINS USERS RESPONSE
CD3 1 - USER ENTERED CHARACTERS OF BUFFER UNPREDICTABLE.
CD3 2 - USER BUFFER CONTAINS NO INFORMATION
CD3 3 - USER PROMPT WAS TOO LONG. MAXIMUM LENGTH IS
CD3 34 CHARACTERS OR 17 WORDS.
CD3
CD**************
CD4 INTERNAL VARIABLES
CD4
CD4 COUNT - COUNTER USED FOR COUNTING NUMBER CHARACTERS IN
CD4 A CHARACTER STRING AND NUMBER DIGITS IN A NUMBER.
CD4 DBLHT - DOUBLE WORD USED TO ACCUMULATE AN INTEGER VALUE
CD4 DBLV0 - DOUBLE WORD USED TO ACCUMULATE A REAL OR DOUBLE
CD4 VALUE
CD4 FLCOM - COMMA FLAG
CD4 Q - LAST CHARACTER NOT A COMMA
CD4 T - LAST CHARACTER WAS A COMMA
CD4 FLCOM - CONTINUE FLAG
CD4 Q - THIS IS NOT A RESPONSE TO A CONTINUE
CD4 T - THIS IS A CONTINUE RESPONSE
CD4 FLEM - NUMBER FLAG
CD4 Q - POSITIVE NUMBER
CD4 T - NEGATIVE NUMBER
CD4 FLGPOW - POWER FLAG
CD4 Q - POSITIVE POWER
CD4
BEGIN XPRDM
IF PROMPT IS NOT TOO LONG THEN
CALL XMOV "" MOVE PROMPT INTO OUTPUT AREA
CALL EXEC TO WRITE PROMPT
SET XPRDM RETURN CODE TO NORMAL RETURN
SET CONTINUE FLAG OFF
INITIALIZE COMMUNICATIONS BUFFER
COMLPO:
CALL XMOV TO INITIALIZE INPUT BUFFER TO BLANKS
CALL EXEC TO READ RESPONSE
CALL XUPK TO CONVERT A2 RESPONSE TO R1 FORMAT
IF NUMBER OF WORDS READ IS NOT ZERO THEN
SET COMMA FLAG ON
DO WHILE CONTINUE FLAG IS OFF
IF INPUT BUFFER IS COMpletely SCANNED THEN
SET CONTINUE FLAG ON
IF COMMA FLAG IS ON THEN
CALL EXEC TO WRITE CONTINUE
GO TO :COMLPO:
ELSE
EXIT IF COMUF IS FULL PERFORM COMFUL
IF INPUT CHARACTER IS A COMMA THEN
IF COMMA FLAG IS ON THEN
ERROID IF COMUF CANNOT HOLD TOKEN PERFORM COMFUL
STORE NULL FIELD TOKEN IN COMUF
INCREMENT #WORDS IN COMUF BY 1
INCREMENT #TOKENS IN COMUF BY 1
ENDIF
SET COMMA FLAG ON
SET NEXT INPUT CHARACTER
ELSE
SET COMMA FLAG OFF
PERFORM TOKENS
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ELSE
SET XPRDM RETURN CODE TO SAY USER ENTERED CR
ENDIF
ENDIF
ELSE
SET XPRDM RETURN CODE TO SAY PROMPT IS TOO LONG
ENDIF
END XPRDM
BEGIN TOKENS
SET NEGATIVE NUMBER FLAG OFF
SET NEGATIVE POWER FLAG OFF
SET POWER = 0
IF INPUT CHARACTER IS A DIGIT THEN
    PERFORM DIGIT
ELSE
    CASE J ((A;B;C;D;E;F;G;H))
    :A:
        SET XPARM RETURN CODE TO SAY I ENTERED
    :B:
        PERFORM QUOTE
    :C:
        SET PAREN INDICATOR = 0
        GO TO :E:
    :D:
        SET PAREN INDICATOR = 1
        GO TO :E:
    :E:
        ERREXIT IF THERE IS NO ROOM FOR THIS TOKEN TO PERFORM COMFULL
        SET TOKEN (OPAR + PAREN INDICATOR) IN COMBUF
        INCREMENT #WORDS IN COMBUF BY 1
        INCREMENT #TOKENS IN COMBUF BY 1
        GET NEXT CHARACTER
    :F:
        ERREXIT IF NEXT CHARACTER IS NOT A DIGIT PERFORM INVAL
    :G:
        PERFORM DECP
    :H:
        IF INPUT CHARACTER IS A - THEN
        SET NEGATIVE NUMBER FLAG ON
        ENDIF
        GET NEXT CHARACTER
        IF INPUT CHARACTER IS A DIGIT THEN
        PERFORM DIGIT
        ELSE
            IF INPUT CHARACTER IS A . THEN
                GO TO :F:
            ELSE
                PERFORM INVAL - NO RETURN
            ENDIF
            ENSDF
        ENDIF
        ENDCASE
    ENDIF
END TOKENS
BEGIN QUOTE
GET NEXT CHARACTER
SET #CHARACTERS = 0
DO WHILE (INPUT CHARACTER IS NOT A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED) OR
(INPUT CHARACTER IS A QUOTE AND
NEXT CHARACTER IS A QUOTE AND
INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED)
INCREMENT #CHARACTERS BY 1
MOVE CHARACTER INTO TEMPORARY BUFFER (#CHARACTERS)
IF INPUT CHARACTER IS A QUOTE THEN
GET NEXT CHARACTER
ENDIF
GET NEXT CHARACTER
ENDDO
ERROR IF LENGTH OF CHARACTER STRING IS 0 OR
ERROR IF INPUT CHARACTER IS NOT A QUOTE PERFORM INVALID
ERROR IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
STORE CHARACTER STRING TOKEN IN COMBUF
STORE NUMBER OF CHARACTERS IN COMBUF
CALL XRPCK TO CONVERT CHARACTERS FROM R1 TO A2 FORMAT
INCREMENT #WORDS IN COMBUF BY 2* (#CHARACTERS + 1)/2
INCREMENT #TOKENS IN COMBUF BY 1
GET NEXT CHARACTER
END QUOTE
1019 1 BEGIN DIGIT
1020 2 PERFORM DEC
1021 3 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1022 4 IF INPUT CHARACTER IS A . THEN
1023 5 PERFORM EORD
1024 6 ELSE
1025 7 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1026 8 PERFORM EORD
1027 9 ELSE
1028 10 IF INPUT CHARACTER IS AN "R" THEN
1029 11 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1030 12 ERREXIT IF INTEGER IS ZERO PERFORM INVAL
1031 13 STORE REPEAT TOKEN IN COMBUF
1032 14 INCREMENT #WORDS IN COMBUF BY 2
1033 15 INCREMENT TOKENS BY 1
1034 16 GET NEXT CHARACTER
1035 17 ELSE
1036 18 PERFORM INTEGR
1037 19 ENDIF
1038 20 ENDIF
1039 21 ELSE
1040 22 PERFORM INTEGR
1041 23 ENDIF
1042 24 ENDIF
1043 25 END DIGIT
1044 26 *
1045 27 *
1046 28 BEGIN DCOL
1047 29 SET INTEGER=0
1048 30 SET COUNTER=0
1049 31 DO WHILE CHARACTER IS A DIGIT AND
1050 32 WHILE INPUT BUFFER IS NOT EXHAUSTED
1051 33 SET INTEGER+(INTEGER * 10) + INPUT CHARACTER - 48
1052 34 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1053 35 INCREMENT COUNTER BY 1
1054 36 GET NEXT CHARACTER
1055 37 END DO
1056 38 END DCOL
1057 39 END

REPRODUCIBILITY OF THE ORIGINAL PAGE FOR

1059 1 BEGIN DECP
1060 2 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
1061 3 GET NEXT CHARACTER
1062 4 IF INPUT BUFFER IS NOT EXHAUSTED THEN
1063 5 IF INPUT CHARACTER IS A DIGIT THEN
1064 6 PERFORM DEC
1065 7 ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
1066 8 EXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1067 9 ENDIF
1068 3 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
1069 4 PERFORM EORD
1070 3 ELSE
1071 4 PERFORM REAL
1072 3 ENDIF
1073 2 ELSE
1074 3 PERFORM REAL
1075 2 ENDIF
1076 1 END DECP
1078  1 BEGIN EORD
1079  2 IF INPUT CHARACTER IS AN "E" THEN
1080  3 SET TYPE FLAG TO "E"
1081  2 ELSE
1082  3 SET TYPE FLAG TO "D"
1083  2 ENDIF
1084  2 GET NEXT CHARACTER
1085  2 ERREXIT IF INPUT BUFFER IS EXHAUSTED PERFORM INVAL
1086  2 IF INPUT CHARACTER IS A - THEN
1087  3 SET NEGATIVE POWER FLAG ON
1088  2 GET NEXT CHARACTER
1089  2 ELSE
1090  3 IF INPUT CHARACTER IS A + THEN
1091  4 SET NEGATIVE POWER FLAG OFF
1092  4 GET NEXT CHARACTER
1093  3 ENDIF
1094  2 ENDIF
1095  2 ERREXIT IF INPUT BUFFER IS EXHAUSTED OR
1096  2 ERREXIT IF INPUT CHARACTER IS NOT A DIGIT PERFORM INVAL
1097  2 PERFORM DCOL
1098  2 IF NEGATIVE POWER FLAG IS ON THEN
1099  3 SET POWER = -POWER
1100  2 ENDIF
1101  2 IF TYPE FLAG IS "E" THEN
1102  3 PERFORM REAL
1103  3 ELSE
1104  3 BEGIN DBL
1105  4 ERREXIT IF NEXT TOKEN IS NOT A COMMA AND
1106  4 ERREXIT IF NEXT TOKEN IS NOT A CLOSED PAREN AND
1107  4 ERREXIT IF INPUT BUFFER IS NOT EXHAUSTED PERFORM INVAL
1108  4 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
1109  4 SET DOUBLE = DUVALUE * 10 ** POWER
1110  4 ERREXIT IF INTEGER OVERFLOW HAS OCCURRED TO PERFORM OVFLOW
1111  4 IF NEGATIVE NUMBER FLAG IS ON THEN
1112  5 SET DOUBLE = - DOUBLE
1113  4 ENDIF
1114  4 STORE DOUBLE TOKEN IN COMBUF
1115  4 INCREMENT #WORDS IN COMBUF BY 4
1116  4 INCREMENT #TOKENS IN COMBUF BY 1
1117  3 END DBL
1118  2 ENDIF
1119  1 END EORD
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1153</td>
<td>1 BEGIN INVAL</td>
</tr>
<tr>
<td>1154</td>
<td>2 CALL CVT TO CONVERT OCTAL CHARACTER NUMBER TO ASCII</td>
</tr>
<tr>
<td>1155</td>
<td>2 CALL EXEC TO WRITE ERROR MESSAGE</td>
</tr>
<tr>
<td>1156</td>
<td>2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN</td>
</tr>
<tr>
<td>1157</td>
<td>1 END INVAL</td>
</tr>
<tr>
<td>1158</td>
<td>1 *</td>
</tr>
<tr>
<td>1159</td>
<td>1 *</td>
</tr>
<tr>
<td>1160</td>
<td>1 *</td>
</tr>
<tr>
<td>1161</td>
<td>1 BEGIN CONFUL</td>
</tr>
<tr>
<td>1162</td>
<td>2 CALL EXEC TO WRITE ERROR MESSAGE</td>
</tr>
<tr>
<td>1163</td>
<td>2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN</td>
</tr>
<tr>
<td>1164</td>
<td>1 END CONFUL</td>
</tr>
<tr>
<td>1165</td>
<td>1 *</td>
</tr>
<tr>
<td>1166</td>
<td>1 *</td>
</tr>
<tr>
<td>1167</td>
<td>1 *</td>
</tr>
<tr>
<td>1168</td>
<td>1 BEGIN OVFLOW</td>
</tr>
<tr>
<td>1169</td>
<td>2 CALL CVT TO CONVERT OCTAL CHARACTER NUMBER TO ASCII</td>
</tr>
<tr>
<td>1170</td>
<td>2 CALL EXEC TO WRITE ERROR MESSAGE</td>
</tr>
<tr>
<td>1171</td>
<td>2 PERFORM XPRM TO DISPLAY ORIGINAL PROMPT - NO RETURN</td>
</tr>
<tr>
<td>1172</td>
<td>1 END OVFLOW</td>
</tr>
</tbody>
</table>
FORTRAN CALLING PROCEDURE

CALL XPXIT (LU, RPAMS)

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

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

OUTPUT

RPAMS IS RETURNED TO THE FATHER TASK

NOTES

USES EXEC, PRIN

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
CALL BIT (BIT, OUTFM, STRING)

BIT SET/CLEAR ROUTINE FOR MULTIPLE WORD BIT STRINGS.

SET THE BIT SET/CLEAR ADDRESS OF WORD CONTAINING BIT

CALL bitset TO SET/CLEAN BIT

END BIT
INTEGER FUNCTION
1 40 XRCPR(LENGTH, ARRAY1, ARRAY2)
41 40
42 40
43 40
44 40
45 40
46 40 COMPARE 'LENGTH' WORDS OF 'ARRAY1' TO 'ARRAY2'. IF THE ARRAYS
47 40 ARE IDENTICAL A VALUE OF 0 IS RETURNED. IF THE COMPARE FAILS
48 40 A VALUE IS RETURNED INDICATING WHETHER THE FIRST UNEQUAL
49 40 ELEMENT OF ARRAY1 IS LESS THAN ARRAY2 (-1) OR GREATER (+1)
50 40
51
52 40 INPUT
53 40 LENGTH - POSITIVE INTEGER NUMBER OF WORDS TO BE COMPARED
54 40 ARRAY1 - ARRAY OF INTERGERS OR CHARACTERS TO BE COMPARED TO ARRAY2
55 40 ARRAY2 - ARRAY OF INTERGERS OR CHARACTERS TO BE COMPARED TO ARRAY1
56 40
57
58 40 OUTPUT
59 40 FUNCTION VALUE - 0, ARRAYS EQUAL
60 40 -1, ARRAYS NOT EQUAL, ARRAY1 < ARRAY2
61 40 +1, ARRAYS NOT EQUAL, ARRAY1 > ARRAY2
62 40
63
64 40 NOTES
65 40 USES .ENTR
66 40
67
68
69
70
71
72
73 40 BEGIN XRCPR
74 40 TRANSFER CALLING SEQUENCE
75 40 SETUP COMPARE
76 40 COMPARE ARRAY1 AND ARRAY2
77 40 RETURN RESULT FLAG
78 40 END XRCPR
FORTRAN CALLING PROCEDURE.

CALL XR18 (DOUBLE, ASCII)

CONVERT A DOUBLE PRECISION REAL NUMBER TO AN ASCII STRING IN

DOUBLE - THREE WORD DOUBLE PRECISION REAL NUMBER TO BE CONVERTED

OUTPUT - NINE WORD ASCII CHARACTER STRING REPRESENTATION OF

DOUBLE LOCAL

D - WORKING LOCATION FOR ABSOLUTE VALUE OR 'DOUBLE'

REPEATEDLY MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES

USES DOUBLE, FLOAT, IAI0, IDINT, IO8, KCVT, XRET

BEGIN XR18

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 XR18
FORTRAN CALLING PROCEDURE

CALL XREQ

MAKE A WORK AREA MANAGEMENT REQUEST AND WAIT FOR RESPONSE

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

ID SEGMENT PARAMETERS RETURNED FROM THE MANAGER

COMMON XE - REQPTR

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

REQBUF AND RESPONSE IS PRINTED IF REQUESTED

COMMUNICATES WITH FDS MANAGER FATHER TASK

USES EXEC, IAND, XRMOV, XRMG, XRO6, XRSFL, XUDPL, XVPAW

BEGIN XREQ

PERFORM TRACE

OUTPUT REQUESTS TO MANAGER

REQUEST AWX MANAGEMENT AND WAIT FOR RESPONSE

RETRIEVE MANAGER RESPONSE

PERFORM TRACE

RETURN RESPONSE IN REQPTR

END XREQ

BEGIN TRACE

IF TRACE REQUESTED

THEN

DO FOR EACH REQUEST

CALL XUDPL TO FORMAT LINE

OUTPUT LINE

ENDDO

OUTPUT PARM1 AND PARM2

ENDIF

END TRACE
1  *INT*  INTEGER FUNCTION
2  *INT*  XREX(START, LENGTH, SOURCE)
3  *INT*  ********
4  *INT*  EXTRACT 'LENGTH' BITS OF 'SOURCE' BEGINNING WITH BIT 'START'
5  *INT*  AND RIGHT ADJUST
6  *INT*
7  *INT*  ********
8  *INT*  INPUT
9  *INT*  START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE EXTRACTED
10  *INT*  (SIGN BIT = 0)
11  *INT*  LENGTH - POSITIVE INTEGER SIZE OF FIELD TO BE EXTRACTED
12  *INT*  SOURCE - WORD FROM WHICH FIELD IS TO BE EXTRACTED
13  *INT*  ********
14  *INT*  NOTES
15  *INT*  USES .EMT
16  *INT*  
17  *INT*  
18  *INT*  
19  *INT*  ********
20  1  *IO*  
21  1  *IO*  
22  1  *IO*  
23  1  *IO*
24  1  *IO*
25  1  *IO*
26  1  *IO*
27  1  *IO*
28  1  *IO*
29  1  *IO*
30  1  *IO*
31  1  *IO*
32  1  *IO*
33  1  *IO*
34  1  *IO*
35  1  *IO*
36  1  *IO*
37  1  *IO*
38  1  *IO*
39  1  *IO*
40  1  *IO*
41  1  *IO*
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
R - WORKING LOCATION FOR ABSOLUTE VALUE OR 'REAL' REPEATEDLY
MODIFIED TO EXTRACT REMAINING DECIMAL DIGITS

NOTES
USES FLOAT, IAND, IFIX, IOR, RCVT, XREAT, 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

1 CD0 CALL XR16 (INTEGER, ASCII)

1 CD0

1 CD1 CONVERT A SIXTEEN BIT SIGNED BINARY INTEGER TO A SIX CHARACTER

1 CD1 ASCII STRING

1 CD2 INPUT

1 CD2 INTEGER - SIXTEEN BIT INTEGER TO BE CONVERTED

1 CD2

1 CD3 OUTPUT

1 CD3 ASCII - THREE WORD CHARACTER STRING REPRESENTATION OF 'INTEGER'

1 CD3

1 CD4 LOCAL

1 CD4 I - INTERNAL LOCATION FOR 'INTEGER' REPEATEDLY MODIFIED TO

1 CD4 PRODUCE 'ASCII'

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

1 CD4

1 CD5 NOTES

1 CD5 USES XRMV AND XRPC

1 CD5

1 CD6

1 CD6 *

1 CD6 *

1 CD6 *

1 CD6 *

1 CD6 BEGIN XR16

1 CD6 BLANK WORKING SPACE

1 CD6 CONSTRUCT 'ASCII' LEAST SIGNIFICANT DIGITS FIRST USING REMAINING

1 CD6 SET SIGN OF 'INTEGER' IN 'ASCII'

1 CD6 CALL XRCPCK TO CONVERT FROM R1 TO A2 FORMAT

1 CD6 END XR16
302 1*0) FORTRAN CALLING PROCEDURES
303 1*0) CALL XRLCK (RCODE)
304 1*0) CALL XRULK (RCODE)
305 1*0)********
306 1*0)********
307 1*0)********
308 1*01 XRLCK AND XRULK PROVIDE A MECHANISM FOR SERIALIZING THE UPDATE
309 1*01 OF FDS GLOBAL SYSTEM TABLES AND FILES. THE RESOURCE NUMBER
310 1*01 STORED IN THE XVSTB RESIDENT STATUS TABLE IS USED AS THE
311 1*01 LOCKING MECHANISM
312 1*01********
313 1*01********
314 1*02 INPU
315 1*02 XVSTB RESOURCE NUMBER
316 1*02********
317 1*02********
318 1*02********
319 1*03 OUTPUT
320 1*03 RCODE - INTEGER RETURN CODE (0 - SUCCESSFUL, 1 - FAILURE)
321 1*03********
322 1*03********
323 1*04 LOCAL
324 1*04 STAT - STATUS OF THIS COPY OF XEXEC USE OF XVSTB RN
325 1*04 1 - RN LOCKED
326 1*04 4 - RN UNLOCKED
327 1*04********
328 1*04********
329 1*05 NOTES
330 1*05 USES .ENTRY, RNRR.
331 1*05********
332 1*05********
333 1*05 THIS ROUTINE MAY NOT BE OVERLAYERED
334 1*05********
335 1*05********

REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
1 BEGIN XRLOCK
2 SET REQUEST FOR LOCK FUNCTION
3 PERFORM RLOCK
4 END XRLOCK
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 RN IN STB IS DEFINED, I.E., FDS HAS INITIALIZED SINCE IBL
14 THEN
15 CALL RNRQ TO ACCOMPLISH FUNCTION (WAIT IF NECESSARY ON LOCK)
16 ENDIF
17 CLEAR RETURN CODE
18 ELSE
19 SET RETURN CODE
20 ENDIF
21 END RLOCK
22 END XRLOCK
INTEGER FUNCTION
360 1 *00
361 1 *00 XRLOC(A)
362 1 *00
363 1 *00
364 1 *01
365 1 *01 RETURN THE 16-BIT MAPPED ADDRESS OF A
366 1 *01
367 1 *00
368 1 *02 INPUT
369 1 *02 A - VARIABLE, ROUTINE, ETC. FOR WHICH THE ADDRESS IS DESIRED
370 1 *02
371 1 *00
372 1 *03 OUTPUT
373 1 *03 XRLOC - 16-BIT ADDRESS OF A
374 1 *03
375 1 *00
376 1 *00 NOTES
377 1 *05 NO EXTERNAL REFERENCES
378 1 *05
379 1 *00
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 *00      FORTRAN CALLING PROCEDURE
390 1 *00
391 1 *00  CALL XRMV (LENGTH, SOURCE, OBJECT)
392 1 *00
393 1 *********
394 1 *01    MOVE 'LENGTH' WORDS FROM 'SOURCE' TO 'OBJECT'
395 1 *01
396 1 *01
397 1 *********
398 1 *02    INPUT
399 1 *02    LENGTH - POSITIVE INTEGER INDICATING NUMBER OF WORDS TO MOV.
400 1 *02    SOURCE - ARRAY OF WORDS TO BE MOVED
401 1 *02
402 1 *********
403 1 *03    OUTPUT
404 1 *03    OBJECT - ARRAY RECEIVING MOVED WORDS
405 1 *03
406 1 *********
407 1 *04    NOTES
408 1 *04    USES .ENTR
409 1 *05
410 1 *********
411 1 *
412 1 *
413 1 *
414 1 *
415 1 BEGIN XRMV
416 1 TRNSFER CALLING SEQUENCE
417 2 INITIALIZE MOVE
418 2 MOVE LENGTH WORDS FROM SOURCE TO OBJECT
419 1 END XRMV
FORTRAN CALLING PROCEDURE

CALL XMSG (NUMBER, LOCATE, LENGTH, SOURCE)

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

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

TRUNCATE TO EIGHTY CHARACTERS AND OUTPUT TO USER TERMINAL.

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

A - AREA INDICATOR AS FOLLOWS

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

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

WORDS OF 'SOURCE' TO BE OUTPUT

LOCATE - INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEDE

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

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

MESSAGE. ZERO INDICATES NO INSERTION

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

IF 'LENGTH' IS ZERO)

COMMON

LU - USERS LOGICAL UNIT NUMBER

C SPECIAL

OUTPUT

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

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

NOTES

USES FDS SYSTEM MESSAGE FILE XMSG

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

C
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 THEN
6    SET AREA CODE IN PREFIX
7    IF MESSAGE NUMBER > 0 THEN
8      IF VALID MESSAGE NUMBER THEN
9        COMPUTE MESSAGE RECORD NUMBER
10       READ RECORD
11       CALL XRMV TO MOVE LOCATE WORDS FROM RECORD INTO BUFFER
12     ELSE
13       EXIT TO :ERROR:
14     ENDIF
15     CALL XRMV TO MOVE LENGTH WORDS FROM SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
16     IF MESSAGE NUMBER > 0 THEN
17       CALL XRMV TO MOVE REMAINING RECORD INTO BUFFER (MAX OF 40 TOTAL WORDS)
18     ELSE
19       ENDIF
20     SET AREA IN PREFIX
21     :ERROR: CALL XRMV TO MOVE 'XRMSE ERROR' INTO BUFFER
22     CALL XRMV TO MOVE LENGTH WORDS OF SOURCE INTO BUFFER (MAX OF 40 TOTAL WORDS)
23     ENDIF
24     OUTPUT BUFFER TO USER'S TERMINAL
25     IF DEBUG IS REQUESTED THEN
26       CALL XDEBUG
27     ENDIF
28 1 END XMSG

S-236
INTEGER FUNCTION

XRNFB(BIT, BITNUM, STRING)

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

**INPUT**

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

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

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

**OUTPUT**

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

**NOTES**

USES .ENTR

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

A MAXIMUM BIT STRING LENGTH OF 65535 BITS (4096 WORDS) CAN BE MEANINGFULLY ACCOMMODATED.

ORIGINAL PAGE OF THE
1 BEGIN XRNKB
2  COMPUTE STARTING WORD NUMBER
3  COMPUTE STARTING BIT NUMBER
4  LOAD STARTING WORD
5  SHIF T WORD UNTIL STARTING BIT NUMBER IS IN SIGN BIT
6  INITIALIZE BIT COUNT
7  DO WHILE SIGN BIT IS NOT EQUAL TO BIT
8  IF WORD NOT FINISHED
9  THEN
10     SHIF T WORD LEFT
11     INCREMENT BIT COUNT
12  ELSE
13     DO UNTIL WORD WITH SOME 'BIT' VALUE FOUND
14     LOAD NEXT WORD
15     ENDDO
16     CLEAR BIT COUNT
17     ENDDO
18  ENDIF
19  ENDDO
20  RETURN VALUE OF MATCHING BIT NUMBER
21 END XRNKB
FORTRAN CALLING PROCEDURE

*00 CALL XRPACK (LENGTH, UNPKED, PACKED)

*00 ********

*01 CONVERT 'LENGTH' CHARACTERS OF 'UNPKED' FROM R1 FORMAT TO A2

*01 ********

*02 INPUT

*02 LENGTH - POSITIVE INTEGER NUMBER OF CHARACTERS IN UNPKED

*02 UNPKED - ARRAY OF CHARACTERS IN R1 FORMAT

*02 ********

*03 OUTPUT

*03 PACKED - ARRAY OF (LENGTH+1)/2 WORDS IN A2 FORMAT. IF LENGTH IS

*03 ODD, THE LAST WORD WILL BE BLANK FILLED.

*03 ********

*05 NOTES

*05 USES .ENTR

1  *05

1 *03

1 ********

BEGIN XRPACK

TRANSFER CALLING SEQUENCE

SET BYTE FLAG FOR HIGH BYTE

INITIALIZE PACKED POINTER

DO FOR EACH CHARACTER IN UNPKED

IF BYTE FLAG SET HIGH

THEN

LOAD A WITH NEXT WORD OF UNPKED

SHIFT CHARACTER INTO HIGH BYTE

ELSE

INCLUSIVE OR NEXT WORD OF UNPKED INTO A

STORE A IN PACKED

INCREMENT POINTER

ENDIF

FLIP BYTE FLAG

ENDIF

IF BYTE FLAG SET LOW

THEN

INCLUSIVE OR BLANK INTO LOW BYTE

ENDIF

STORE A IN PACKED

END XRPACK
FORTRAN CALLING PROCEDURE

CALL XRQFN (PREFIX, NAME4, NAME6)

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

XRQFN 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
END

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

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

INPUT

PREFIX - FILE TYPE PREFIX STORED IN R1 FORMAT

NAME4 - ONE TO FOUR CHARACTER PACKED NAME TO BE QUALIFIED

COMMON XE - QUAL.

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

OUTPUT

NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME

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

NOTES

USES .ENTR

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

* 

BEGIN XRQFN

STORE PREFIX IN FIRST POSITION OF INTERNAL CHARACTER STRING

MOVE NAME4 INTO NEXT FOUR POSITIONS

STORE BLANK IN SIXTH POSITION

LOCATE FIRST BLANK CHARACTER

REPLACE BLANK WITH USER ID CHARACTER (QUAL)

MOVE QUALIFIED NAME TO NAME6

END XRQFN
1  *00  FORTRAN CALLING PROCEDURE
696 1  *00
697 1  *00  CALL XRSET (START, LENGTH, SOURCE, OBJECT)
698 1  *00
699 1  *00
700 1  *********
701 1  *01
702 1  *01  REPLACE "LENGTH" BITS OF "OBJECT", BEGINNING WITH BIT "START",
703 1  *01  WITH THE RIGHT MOST "LENGTH" BITS OF "SOURCE"
704 1  *01
705 1  *********
706 1  *02  INPUT
707 1  *02  START - INTEGER INDICATING LEFT MOST BIT OF FIELD TO BE REPLACED
708 1  *02  (SIGN BIT = 0)
709 1  *02  LENGTH - POSITIVE INTEGER SIZE OF FIELD BEING REPLACED
710 1  *02  SOURCE - WORD CONTAINING REPLACEMENT FIELD RIGHT ADJUSTED
711 1  *02
712 1  *********
713 1  *03  OUTPUT
714 1  *03  OBJECT - WORD INTO WHICH FIELD IS TO BE INSERTED
715 1  *03
716 1  *********
717 1  *05  NOTES
718 1  *05  USES .ENTR
719 1  *05
720 1  *********
721 1  *
722 1  *
723 1  *
724 1  *
725 1  BEGIN XRSET
726 2  TRANSFER CALLING SEQUENCE
727 2  CONSTRUCT SHIFT INSTRUCTIONS
728 2  SHIFT LENGTH BITS OF SOURCE INTO HIGH END OF CLEARED REGISTER
729 2  SHIFT REGISTER RIGHT START BITS TO PROPERLY POSITION FIELD
730 2  CONSTRUCT MASK AND CLEAR FIELD OF OBJECT
731 2  INCLUSIVE OR POSITIONED SOURCE FIELD INTO OBJECT
732 1  END XRSET
INTEGER FUNCTIONS

1  *B0 XRSFL(COUNT, SOURCE)
2  *B0 XRSFR(COUNT, SOURCE)
1  *B0 ********
1  *D1  SHIFT 'SOURCE' LEFT/RIGHT LOGICALLY 'COUNT' BITS
2  *D1  ********
1  *D2  INPUT
2  *D2  COUNT - POSITIVE INTEGER SPECIFYING NUMBER OF BITS TO SHIFT
1  *D2  SOURCE - WORD TO BE SHIFTED
2  *D2  ********
1  *D5  NOTES
2  *D5  USES .ENTR
1  *D5  ********
1  *  1  *
1  *  1  *
1  *  1  *
1  *  BEGIN XRSFL
1  BEGIN XRSFR
2  TRANSFER CALLING SEQUENCE
2  CONSTRUCT SHIFT INSTRUCTION
2  LOAD A WITH SOURCE
2  CLEAR B
2  SHIFT BA AS SPECIFIED
1  END SHIFT
**FORTRAN CALLING PROCEDURE**

1 *DO
2 *DO
3 *DO
4 CALL XRUN6 (NAME6, NAME4)
5 *DO
6 ***********
7 XRUN6 REMOVES THE PREFIX AND SUFFIX QUALIFYING CHARACTERS FROM
8 A SIX CHARACTER FILE NAME
9 ***********
10 ***********
11 INPUT
12 *DO NAME6 - THREE TO SIX CHARACTER PACKED QUALIFIED NAME
13 *DO
14 ***********
15 OUTPUT
16 *DO NAME4 - ONE TO FOUR CHARACTER PACKED NAME WITH PREFIX AND SUFFIX
17 REMOVED
18 *DO
19 ***********
20 NOTES
21 *DO USES .ENTR
22 *DO
23 ***********
24 *
25 *
26 *
27 *
28 *
29 *
30 BEGIN XRUN6
31 MOVE CHARACTERS 2-5 OF NAME6 INTO NAME4
32 IF SIXTH CHARACTER IS BLANK, THEN
33 LOCATE LAST NON-BLANK CHARACTER OF NAME4
34 BLANK THAT CHARACTER
35 ENDIF
36 END XRUN6
FORTRAN CALLING PROCEDURE

CALL XRUPK (LENGTH, PACKED, UNPKED, COUNT)

**********

CONVERT 'LENGTH' WORDS OF 'PACKED' FROM A2 FORMAT TO R1 FORMAT,
REMOVING BLANKS NOT DELIMITED BY QUOTE MARKS, AND RETURN IN
'UNPKED' WITH THE NON-BLANK CHARACTER COUNT IS 'COUNT'.

INPUT
LENGTH - POSITIVE INTEGER NUMBER OF WORDS IN PACKED
PACKED - ARRAY OF CHARACTER DATA IN A2 FORMAT

**********

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

**********

NOTES
USES .ENTR
IF PACKED AND UNPKED ARE THE SAME ADDRESS SPACE UNPKED MAY OVERLAY
PACKED

** CAUTION: XRUPK CANNOT HANDLE QUOTE MARKS WITHIN CHARACTER
STRINGS.

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

100 CALL XRSIP(CHSTR, LENTH)

105 **********

106 This routine assumes

107 1. The input character string has a positive length > 0

108 2. Special handling of strings within quotation marks does not

109 begin until the first occurrence of quotation marks on each

110 entry

111 3. The first/last character in a character string has the

112 high order bit of the quote character '0' to indicate

113 begin/end of a character string. This bit is turned 'OFF'

114 by this program prior to exit. This change made to

115 incorporate quote marks w/i character strings.

116 Other processors changed for this modification were:

117 A. XILAN, XPAR

118 B. XILSS/XILSC

119 **********
FORTRAN CALLING PROCEDURE

CALL XNLBS(XSEGE)

XSEGE IS THE MAIN ROUTINE OF THE SEQUENCE TABLE EDITOR

INPUT

COMMON XE - LU

COMMON XB - DEBUG, IRETC, NEWTAB, NUMENT, PRMLEN

PROMPT, PRMND, WKBUF

OUTPUT

COMMON XE - REBUF, REAPTR

COMMON XB - NUMENT, PRMLEN, PRMND, PROMPT, WKBUF

NOTES

USES Routines
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)
  ENDIF
  BUILD PROMPT OF THE FORM ' MNNNNN-PROC,INT'
  SET PROMPT SEQUENCE NUMBER (SEENO) TO SEQUENCE NO. OF ENTRY
  ELSE
  SET PROMPT NO.6 TO CREATE
  ENDIF
  ENDIF
  IF PROMPT MODE IS CREATE, THEN
  SET TABLE ENTRY INDEX (TABndx) TO NEXT ENTRY (NUMENT * 7 + 1)
  IF NUMBER OF TABLE ENTRIES (NUMENT) > 0, THEN
  IF SEQUENCE NO. OF LAST ENTRY > 32695, THEN
  CALL XPAR = 'UNABLE TO BUILD SEQUENCE NO. > 32700'
  SET PROMPT MODE TO UPDATE
  ELSE
  IF PROMPT SEQUENCE NUMBER (SEENO) TO NEXT MULTIPLE OF 100
  BEYOND SEQUENCE NUMBER OF LAST TABLE ENTRY
  ENDIF
  ELSE
  SET PROMPT SER. NO. (SEENO) TO BE 100
  ENDIF
  IF PROMPT MODE IS NOT UPDATE, THEN
  BUILD PROMPT OF THE FORM ' MNNNNN
  ENDIF
  IF PROMPT MODE IS UPDATE, THEN
  BUILD PROMPT OF THE FORM
  SET PROMPT LENGTH TO 0 CAUSING #: PROMPT TO BE ISSUED
  ENDIF
END XSPRM
1 CD0  FORTRAN CALLING PROCEDURE
1 CD0
1 CD0  CALL XSNPT
1 CD0
1 CD0  ********
1 CD1
1 CD1  XSNPT PROCESSES THE INPUT RESPONSES OF THE SEQUENCE
1 CD1
1 CD1  TABLE EDITOR
1 CD1
1 CD1  ********
1 CD2
1 CD2  ********
1 CD2  INPUT
1 CD2
1 CD2  COMMON XE = COMBUF, COMPTR, LU, TOKENS
1 CD2
1 CD2  COMMON XB = DEBUG, DIRECT, NUMDIR, NUMENT, PMTMD,
1 CD2  SEQNO, TABNOX, WKBUF
1 CD2
1 CD2
1 CD2
1 CD2  ********
1 CD3
1 CD3  OUTPUT
1 CD3
1 CD3  COMMON XE = COMPTR
1 CD3
1 CD3  COMMON XB = INSERT, IRET, NUMENT, PMTMD, SEQNO,
1 CD3  TABNOX, TABSZ, WKBUF
1 CD3
1 CD3
1 CD3
1 CD3  ********
1 CD5
1 CD5  NOTES
1 CD5
1 CD5  USES ROUTINES
1 CD5
1 CD5  XMSG
1 CD5
1 CD5  XDEL
1 CD5
1 CD5  XSNPT
1 CD5
1 CD5  XSLIS
1 CD5
1 CD5  XSNUM
1 CD5
1 CD5  XSMPT
1 CD5
1 CD5  XUDSG
1 CD5
1 CD5
1 CD5
1 CD5  ********
200 1 BEGIN XSNPT
201 2  IF PROMPT MODE IS UPDATE, THEN
202 3  IF TOKEN INPUT IS AN INTEGER, THEN
203 4  ERREXIT IF INTEGER < 1 :ERROR:
204 5  RETAIN INTEGER AS SEQUENCE NO. (SENNO)
205 6  INCREMENT TO NEXT TOKEN
206 7  ERREXIT IF TOKEN IS NOT "=" :ERROR:
207 8  INCREMENT TO NEXT TOKEN
208 9  START SEARCH UNTIL NUMBER OF TABLE ENTRIES (NUMENT) SEARCHED
210 10 EXIT IF SEQUENCE NO. OF ENTRY = SEQUENCE NO. INPUT (SENNO.)
211 11 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
212 12 SET INSERT FLAG TO ZERO IMPLICATING REPLACEMENT OF ENTRY
213 13 SET TABLE ENTRY INDEX (TABNOX) TO THIS ENTRY
214 14 SET INSERT FLAG TO 1 INDICATING INSERT NEEDED
215 15 GORELSE
216 16 5 INCREMENT TO NEXT TABLE ENTRY
217 17 ENDOLOOP
218 18 SET TABLE ENTRY INDEX (TABNOX) TO NEXT ENTRY OF TABLE
219 19 SET INSERT FLAG TO 2 INDICATING EXTENSION TO END OF TABLE
220 20 ENDSERCH
221 21 CALL XSNT TO REPLACE/INSERT/ADD TABLE ENTRY BASED ON INSERT FLAG
222 22 ELSE
223 23 3 ERREXIT IF TOKEN IS NOT A NAME :ERROR:
224 24 START SEARCH UNTIL LIST OF SEQ. EDIT. DIRECTIVES SEARCHED
225 25 EXIT IF NAME INPUT IS DIRECTIVE
226 26 SET INDEX TO DIRECTIVE LIST ENTRY
227 27 GORELSE
228 28 5 INCREMENT TO NEXT DIRECTIVE
229 29 ENDOLOOP
230 30 ERREXIT :ERROR:
231 31 ENDSERCH
232 32 CASE (XSLIS, XSD, XSNPMT, XSNUM, XSNMGR), INDEX
233 33 ENDFI
234 34 ELSE, PROMPT MODE IS NOT UPDATE
235 35 IF TOKEN IS #, THEN
236 36 SET PROMPT MODE TO UPDATE
237 37 F AN & IS INPUT, THEN
238 38 ERREXIT IF PROMPT MODE IS NOT UPDATE :ERROR:
239 39 MARK THIS TABLE ENTRY AS DELETED
240 40 DO FROM END OF TABLE UNTIL A NONDELETED ENTRY IS FOUND
241 41 IF TABLE ENTRY IS MARKED FOR DELETION, THEN
242 42 DECREMENT NUMBER OF TABLE ENTRIES BY ONE
243 43 ENDFI
244 44 ENDDO
245 45 ELSE
246 46 IF PROMPT MODE IS ALL, THEN
247 47 SET INSERT FLAG TO TWO TO INDICATE EXTENSION OF TABLE
248 48 ELSE
249 49 SET INSERT FLAG TO ZERO TO INDICATE REPLACEMENT OF TABLE ENTRY
250 50 ENDFI
251 51 CALL XSNT TO BUILD ENTRY BASE ON INSERT FLAG
252 52 ENDFI
253 53 ENDFI
254 54 ENDFI
255 55 EXIT XSNPT
256 56 :ERROR: CALL XRMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
263 CDO FORTRAN CALLING PROCEDURE
264 CDO CALL XSENT
265 CDO
266 CDO
267 CDO
268 CDO
269 CDO
270 CDO
271 CDO
272 CDO XSENT CONSTRUCTS SEQUENCE TABLE ENTRIES
273 CDO COMMON XE - COMBUF, COMPTR, TOKENS, LU
274 CDO COMMON XB - DEBUG, LIBSIZ, NUMENT, PRMTMD,
275 CDO SEGNO, TABMDX, WBUF
276 CDO
277 CDO
278 CDO
279 CDO
280 CDO
281 CDO
282 CDO
283 CDO COMMON XE - COMPTR
284 CDO COMMON XB - INTNAM, NUMENT, PRCNAM, PRMTMD,
285 CDO SEGNO, TABMDX, WBUF
286 CDO
287 CDO
288 CDO
289 CDO
290 CDO
291 CDO
292 CDO
293 CDO NOTES
294 CDO
295 CDO
296 CDO
297 CDO
298 CDO
299 CDO
300 CDO
301 CDO
302 CDO
303 CDO BEGIN
304 CDO IF TOKEN INPUT IS NOT A NAME :ERRO1:
305 CDO RETAIN THIS NAME AS PROC. NAME
306 CDO INCREMENT TO NEXT TOKEN
307 CDO START SEARCH UNTIL ALL ENTRIES OF XLIBD SEARCHED
308 CDO EXIT IF XLIBD ENTRY = PROC. NAME
309 CDO ORELSE
310 CDO INCREMENT TO NEXT ENTRY
311 CDO ENDDO
312 CDO ERREXIT :ERRO3:
313 CDO ENDS
314 CDO IF COMMA IS NEXT TOKEN, THEN
315 CDO ERREXIT IF INT. TABLE NOT REQUIRED FOR THIS PROCESSOR :ERRO4:
316 CDO INCREMENT TO NEXT TOKEN
317 CDO ERREXIT IF NEXT TOKEN IS NOT A NAME :ERRO1:
318 CDO RETAIN THIS NAME AS INTERFACE TABLE NAME
319 CDO INCREMENT TO NEXT TOKEN
320 CDO ELSE
321 CDO SET INTERFACE TABLE NAME TO ZERO
ENDIF

ENDIF IF NEXT TOKEN IS NOT EQS :ERROR:

IF INSERT FLAG DOES NOT INDICATE REPLACE, THEN (I.E. INSERT OR ADD)

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

CALL XSPC2 TO PACK TABLE BUFFER (REMOVE DELETED ENTRIES)

IF NUMBER OF TABLE ENTRIES STILL = 0, THEN

SET PROMPT MODE TO UPDATE

ERROR :ERROR5:

ENDIF

ENDIF IF INSERT FLAG INDICATES INSERT (+1), THEN

IF ENTRY ABOVE INDEXED ENTRY (TABNDX) IS MARKED DELETED, THEN

SET TABLE ENTRY INDEX (TABNDX) TO BE THIS DELETED ENTRY

SET INSERT FLAG TO 0 INDICATING ENTRY REPLACEMENT

ELSE

SET MOVLEN = MIN (5, 150-NUMENT) + 7

DO FOR ALL TABLE ENTRIES FROM BOTTOM OF TABLE TO TABNDX

MOVE THE ENTRY DOWN MOVLEN WORDS

ENDD

IF MOVLEN > 7 (I.E. MORE THAN 1 ENTRY), THEN

MARK FOLLOWING ENTRIES AS DELETED

ENDIF

ENDIF

SET SEQUENCE NO. FIELD OF ENTRY TO SEQUENCE NO. (SERO) INPUT/PROMPTED

ENDIF

MOVE PROC. NAME AND INT. NAME INTO TABLE ENTRY AT TABNDX

IF INSERT FLAG DOES NOT INDICATE ENTRY REPLACEMENT, THEN

INCREMENT NUMBER OF ENTRIES (NUMENT) BY 1

ENDIF

EXIT XSENT

:ERROR1: CALL XRMEXIT 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'

:ERROR2: CALL XRMEXIT '..... IS NOT A VALID PROCESSOR NAME'

:ERROR3: CALL XRMEXIT '..... DOES NOT USE AN INTERFACE TABLE'

:ERROR4: CALL XRMEXIT 'MAX. SIZE OF 150 SEQUENCE ENTRIES ALREADY REACHED'

END XSENT
BEGIN XSPCK

IF THE TABLE IS NOT EMPTY, THEN

DO UNTIL NUMBER OF ENTRIES (NUMENT) PROCESSED

IF THIS ENTRY IS MARKED DELETED, THEN

SET MOVE LENGTH (MOVLEN) TO 7

DO UNTIL A NON-DELETED ENTRY IS FOUND

INCREMENT MOVLEN BY 7

ENDIF

ENDO

MOVE MOVLEN WORDS BEGINNING WITH THE NON-DELETED ENTRY TO

THE DELETED ENTRY

DECREMENT NUMENT BY MOVLEN / 7

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

DECREMENT TABLE INDEX (TABNOX) BY MOVLEN

ENDIF

END

END XSPCK
414 2  CD0  FORTRAN CALLING PROCEDURE
415 2  CD0
416 2  CD0  CALL XSLST
417 2  CD0
418 2  CD0
419 2  ********
420 2  CD0
421 2  CD0  XSLST WILL LIST TO A SPECIFIED DEVICE THE SEQUENCE TABLE
422 2  CD0  CONTAINED IN THE WORKING BUFFER
423 2  CD0
424 2  ********
425 2  CD0
426 2  CD0  INPUT
427 2  CD0
428 2  CD0  COMMON XE - LU, RERBUF, SUBSTA
429 2  CD0
430 2  CD0  COMMON XB - BENO, DEBUG, ENDNO, LISTLU,
431 2  CD0
432 2  CD0
433 2  CD0  ********
434 2  CD0
435 2  CD0  NOTES
436 2  CD0
437 2  CD0  USES ROUTINES
438 2  CD0
439 2  CD0  XRIG
440 2  CD0  XRMOV
441 2  CD0  XGSET
442 2  CD0  XDBG
443 2  CD0
444 2  CD0  ********
445 3  BEGIN XSLST
446 4  IF SUBSTATE FLAG INDICATES THAT SEQ. EDITOR NOT MAKING THIS CALL, THEN
447 4  DETERMINE SIZE OF SEQ. TAB FROM AWA REQUEST BUFFER ENTRY
448 4  SET LIMITS (BENO AND ENDNO) OF SEQ. ENTRIES LISTED
449 4  SET TABLE NAME (NEWTAB) FROM AWA REQUEST BUFFER ENTRY
450 3  ENDIF
451 3  WRITE HEADER LINE - 'SEQUENCE TABLE XXXXX'
452 3  IF SEQ. TABLE ENTRY IS NOT MARKED AS DELETED, THEN
453 4  DO FROM BENO TO ENDNO
454 5  MOVE PROC. NAME AND INT. NAME FROM ENTRY TO PRINT BUFFER
455 5  IF INT. TABLE NAME = 0, THEN
456 6  SET LENGTH OF PRINT TO BE 7 WORDS (14 CHARs.)
457 6  ELSE
458 6  SET LENGTH OF PRINT LINE TO BE 10 WORDS (20 CHARs.)
459 6  ENDIF
460 5  CALL XRIG TO CONVERT SEQ. NO. OF TABLE ENTRY AND PLACE IN BUFFER
461 5  WRITE PRINT BUFFER
462 4  ENDIF
463 3  ENDDO
464 2  END XSLST
BEGIN XSCAN
SET LIST LIMITS (BEGIN AND ENDOC) TO ZERO
IF NEXT TOKEN IS A COMMA, THEN
  INCREMENT TO NEXT TOKEN
IF NEXT TOKEN IS AN INTEGER, THEN
  ERROR 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
ERROR IF VALUE IS < 1 :ERROR6:
SET END LIMIT (ENDOC) TO THIS VALUE
INCREMENT TO NEXT TOKEN
ENDIF
ENDIF
ENDIF
ERENCE 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
ENDIF
END LOOP
ELSE
  ERROR :ERROR6:
ENDIF
ENDIF
ELSE
  START SEARCH FROM BEGIN LIMIT (BEGIN) TO LAST TABLE ENTRY
  EXIT IF SEQ. NO. OF THIS ENTRY = END LIMIT (ENDOC)
  SET END LIMIT (ENDOC) TO INDEX OF THIS ENTRY
  ORELSE
  INCREMENT INDEX TO NEXT TABLE ENTRY
ENDIF
END LOOP
ELSE
  ERROR :ERROR6:
ENDIF
ENDIF
ELSE
  ERROR :ERROR1: CALL 0RMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
ENDIF
ELSE
  ERROR :ERROR6: CALL 0RMSG - 'INVALID SEQUENCE NUMBER'
ENDIF
ELSE
  ERROR :ERROR6: CALL 0RMSG - 'INVALID SEQUENCE NUMBER RANGE'
ENDIF
EXIT XSLIB
  547  3  :ERROR1: CALL 0RMSG - 'SYNTAX ERROR - MISSING OR EXTRANEOUS FIELD'
  548  3  :ERROR6: CALL 0RMSG - 'INVALID SEQUENCE NUMBER'
  549  3  :ERROR6: CALL 0RMSG - 'INVALID SEQUENCE NUMBER RANGE'
  550  2  END XSCAN
FORTRAN CALLING PROEDURE

CALL XSPMT

**********

XSPMT PROCESSES THE SEQUENCE TABLE EDITOR PROMPT DIRECTIVE

**********

INPUT

COMMON XE -- COMBUF, COMPS, LU, TOKENS

COMMON XB -- DEBUG

**********

OUTPUT

COMMON XB -- PRMTMD, TABNOX

**********

NOTES

USES ROUTINES

XRMG

XUDSG

**********

BEGIN XSPMT

ERREXIT IF TOKEN IS NOT COMMA :ERROR1:

ERREXIT IF TOKEN IS NOT A NAME :ERROR1:

ERREXIT IF TOKEN IS NOT EOS :ERROR1:

IF NAME IS 'M', THEN

SET PROMPT MODE TO CREATE

ELSE

ERREXIT IF NAME IS NOT 'A' :ERROR9:

SET PROMPT MODE TO ALL

SET TABLE ENTRY INDEX (TABNOX) TO 0

ENDIF

ERR9: CALL XRMG -- 'SYNTAX ERROR -- MISSING OR EXTRANEOUS FIELD'

ERR1: CALL XRMG -- 'SYNTAX ERROR -- INVALID QUALIFIER'

END XSPMT
FORTRAN CALLING PROCEDURE

CALL JSDEL

XDEL IS THE SEQUENCE TABLE EDITOR DELETE DIRECTIVE PROCESSOR

INPUT

COMMON XE - LU

COMMON XB - BEGMO, DEBUG, ENDMO, IRETC, NUMENT

OUTPUT

COMMON XB - NUMENT, WKBUF

COMMON XE - NUMENT

NOTES

USES ROUTINES

XSCAN

XUDG

BEGIN XDEL

CALL XSCAN TO SCAN AND INTERPRET SEQ LIMITS ON THE DIRECTIVE

IF NO ERROR INDICATED, THEN

DO FROM THE BEGIN LIMIT (BEGMO) TO THE END LIMIT (ENDMO)

MARK THIS SEQ. TABLE ENTRY AS DELETED

ENDDO

DO FROM LAST TABLE ENTRY TO FIRST ENTRY, OR

UNTIL A NON-DELETED ENTRY FOUND

IF ENTRY IS MARKED DELETED, THEN

DECREMENT NUMBER OF TABLE ENTRIES (NUMENT) BY ONE

ENDIF

ENDDO

ENDIF

END XDEL
FORTRAN CALLING PROCEDURE

CALL XSLTS

CODE

XSLTS IS THE SEQUENCE TABLE EDITOR LIST DIRECTIVE PROCESSOR

INPUT

COMMON XE - LU

COMMON XB - DEBUG, ETC

CODE

CODE

NOTES

USES ROUTINES

XICAN

XSLST

XUDBG

FORTRAN CALLING PROCEDURE

CALL X$NUM

CODE

X$NUM IS THE SEQUENCE TABLE EDITOR NUMBER DIRECTIVE PROCESSOR

INPUT

COMMON XE - COMBUF, COMPTA, LU, TOKENS

COMMON XB - Debug, NUMDEF

OUTPUT

COMMON XB - WKBUF

CODE

NOTES

USES ROUTINES

XMSG

XUDBG
61 1 C05  EQUVALENCE (XS(1), INBUF), (XS(81), MOIN),
62 1 C05  + (XS(82), OUTBUF), (XS(122), SSFLAG),
63 1 C05  + (XS(123), INLEN)
64 1 C05  INBUF - 80 WORD ARRAY THAT CONTAINS THE USER'S RESPONSE
65 1 C05  IN LEN - LENGTH OF ORIGINAL PROMPT PASSED TO XTPRM
66 1 C05  MOIN - NUMBER OF WORDS OF INBUF ACTUALLY USED FOR THE
67 1 C05  USER'S RESPONSE
68 1 C05  OUTBUF - 40 WORD AREA WHERE PROMPT IS CONSTRUCTED AND
69 1 C05  WRITTEN FROM.
70 1 C05  SSFLAG - SYMBOLIC STRING INDICATOR FLAG
71 1 C05  0 = NOT WITHIN A SYMBOLIC STRING
72 1 C05  NONZERO = VALUE REPRESENTING THE INDEX INTO COMBUF
73 1 C05  WHERE LENGTH OF SS IS TO BE STORED
74 1 C05  SUBROUTINES AND FUNCTIONS CALLED:
75 1 C05  EXEC, KCVT, XRMOV, XRMSG, XNUPK, XCTAN, XTPRM
76 1 C05  PDL ROUTINES INCLUDED:
77 1 C05  XCTCOM, READSEG
BEGIN XTCOM

IF PROMPT IS NOT TOO LONG (76 CHAR) THEN
    MOVE PREFIX CHARACTER FOR EXEC LEVEL INTO OUTPUT AREA
    MOVE PROMPT INTO OUTPUT AREA
    MOVE SUFFIX CHARACTER INTO OUTPUT AREA
ENDIF

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

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

IF LA RETURN CODE SAYS OVERFLOW/UNDERFLOW THEN
    CALL KCVT TO CONVERT OCTAL TO ASCII
    CALL XRMIG TO WRITE ERROR MESSAGE
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF

IF LA RETURN CODE SAYS EXTENDED PROMPTING WAS REQUESTED THEN
    CALL XTPRM FOR EXTENDED PROMPT
    IF EXTENDED PROMPT (EP) RETURN CODE SAYS INVALID REQUEST THEN
        CALL XRMIG TO WRITE ERROR MESSAGE
    ENDIF
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF

IF LA RETURN CODE SAYS COMNT IS FULL THEN
    CALL XRMIG TO WRITE ERROR MESSAGE
    GO TO :LOOP: TO DISPLAY ORIGINAL PROMPT
ENDIF

SET XTCON RETURN CODE = LA RETURN CODE
ELSE
    SET XTCON RETURN CODE = PROMPT IS TOO LONG
ENDIF

END XTCON
FORTRAN CALLING PROCEDURE FOR LEXICAL ANALYSIS:

CALL XLAM (RETC)

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

OUTPUTS IN CALLING SEQUENCE:

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

0 - NORMAL RETURN. BUFFER CONTAINS RESPONSE.
1 - USER RESPONDED YES. BUFFER CONTAINS RESPONSE UP TO AND INCLUDING X.
5 - USER REQUESTED A CONTINUATION.
6 - EXTENDED PROMPTING REQUEST WAS RECEIVED. BUFFER CONTAINS RESPONSE UP TO AND INCLUDING THE REQUEST.
7 - COMMUNICATIONS BUFFER IS FULL.
1XX - ERROR IN RESPONSE AT OR BEYOND CHARACTER XX.
2XX - OVERFLOW/UNDERFLOW DETECTED AT OR BEYOND CHARACTER XX.

INTERNAL VARIABLES

COMLEN - LENGTH IN WORDS OF COMBUF = 256
DDBINT - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER VALUE
DDBLW - DOUBLE PRECISION LOCATION TO ACCUMULATE AN INTEGER AND FRACTIONAL VALUE FOR DOUBLE PRECISION OR REM.
FLGCON - COMMA FLAG
FLGEND - END LOOP FLAG
FLGNEG - NEGATIVE EXPONENT FLAG
FLGPOS - EXPONENT WAS POSITIVE
FLGNEG - EXPONENT WAS NEGATIVE
FLGTYP - TYPE OF REAL VALUE
FLGPREC - SINGLE PRECISION
FLGDOUBLE - DOUBLE PRECISION
FLGPOW - EXPONENT PARTOF A REAL NUMBER
RELRET - SINGLE PRECISION LOCATION FOR REAL VALUE
SCCHAR - 25 SPECIAL CHARACTER ARRAY CONTAINING THE KI FORMAT REPRESENTATION FOR:
"!-<>/@#&=\(?)''XZ.<.:DEW
X IS A CLOSED BRACKET
Y IS AN OPEN BRACKET
Z IS A BACK SLASH
204 1 CD4
205 1 CD5**********
206 1 CD5**********
207 1 CD5**********
208 1 CD5**********
209 1 CD5**********
210 1 CD5**********
211 1 CD5**********
212 1 CD5**********
213 1 CD5**********
214 1 CD5**********
215 1 CD5**********
216 1 CD5**********
217 1 CD5**********
218 1 CD5**********
219 1 CD5**********
220 1 CD5**********
221 1 CD5**********
222 1 CD5**********
223 1 CD5**********
224 1 CD5**********
225 1 CD5**********
226 1 CD5**********
227 1 CD5**********
228 1 CD5**********

REPRODUCIBILITY OF THE
ORIGINAL YABLES PAGE
1 BEGIN XTLAN
2   SET END FLAG OFF
3   SET LAST CHARACTER WAS A COMMA ON
4   INITIALIZE INDEX INTO INPUT BUFFER
5   INITIALIZE RETURN CODE TO NORMAL RETURN
6   DO WHILE END FLAG IS OFF
7   IF INPUT BUFFER HAS BEEN COMPLETELY SCANNED THEN
8       IF LAST CHARACTER WAS A COMMA OR A SYMBOLIC STRING IS STILL OPEN THEN
9           SET RETURN CODE TO SAY CONTINUATION REQUESTED
10      ENDIF
11   S ET END FLAG ON
12 ELSE
13   ERREXIT IF COMBUF IS FULL PERFORM COMFUL
14     IF INPUT CHARACTER IS A COMMA THEN
15       PERFORM COMPA
16 ELSE
17     SET LAST CHARACTER WAS A COMMA OFF
18     IF INPUT CHARACTER IS AN ALPHABETIC CHARACTER THEN
19       PERFORM ALPHA
20 ELSE
21       IF INPUT CHARACTER IS A DIGIT THEN
22         PERFORM DIGIT
23 ELSE
24         PERFORM SCHARS
25 ENDIF
26 ENDIF
27 ENDIF
28 ENDDO
29 STORE END OF BUFFER TOKEN IN COMBUF
30 INCREMENT #TOKENS BY 1
31 END XTLAN
32
33 1 BEGIN COMMA
34   SET LAST CHARACTER WAS A COMMA ON
35 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
36 STORE COMMA TOKEN IN COMBUF
37 INCREMENT #WORDS IN COMBUF BY 1
38 INCREMENT #TOKENS BY 1
39 GET NEXT INPUT CHARACTER
40 END COMMA
1 BEGIN ALPHA
2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM COMFUL
3 CALL XMOV TO INITIALIZE TEMPORARY BUFFER WITH 6 BLANKS
4 DO WHILE (INPUT CHARACTER IS AN ALPHA CHARACTER OR
5 INPUT CHARACTER IS A NUMERIC OR
6 INPUT CHARACTER IS AN EXCLAMATION POINT) AND
7 INPUT BUFFER HAS NOT BEEN COMPLETELY SCANNED
8 MOVE CHARACTER INTO TEMPORARY BUFFER
9 SET NEXT INPUT CHARACTER
10 ENDDO
11 SET CHARACTER COUNT = 6
12 STORE CHARACTER NAME TOKEN IN COMBUF
13 CALL ZRPCK ROUTINE TO PACK CHARACTERS INTO COMBUF
14 INCREMENT WORDS IN COMBUF BY 4
15 INCREMENT #TOKENS BY 1
16 END ALPHA
17 1
18 1
19 1
20 BEGIN DIGIT
21 INITIALIZE POWER TO ZERO
22 PERFORM DCOL
23 IF INPUT BUFFER IS NOT EXHAUSTED THEN
24 IF INPUT CHARACTER IS A . THEN
25 PERFORM DECPY
26 ELSE
27 IF INPUT CHARACTER IS AN "E" OR A "D" THEN
28 CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
29 PERFORM EOMB
30 ELSE
31 IF INPUT CHARACTER IS AN "R" THEN
32 PERFORM REPET
33 ELSE
34 PERFORM INTEGR
35 EMDIF
36 ENDIF
37 EMDIF
38 ENDIF
39 ELSE
40 PERFORM INTEGR
41 EMDIF
42 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      IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
  8      INCREMENT COUNTER BY 1
  9      GET NEXT CHARACTER
 10   END DO
11 1 END DCOL
12 1 *
13 1 *
14 1 *
15 1 BEGIN DECF
16 2   CONVERT INTEGER VALUE TO DOUBLE PRECISION VALUE
17 3   GET NEXT INPUT CHARACTER
18 4   IF INPUT BUFFER IS NOT EXHAUSTED THEN
19 5      IF INTEGER IS A DIGIT THEN
20 6         PERFORM DCOL
21 7         ADD FRACTIONAL PART TO DOUBLE PRECISION VALUE
22 8         IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
23 9      ENDIF
24 3      IF CHARACTER IS AN "E" OR A "D" THEN
25 4         PERFORM EORD
26 3      ELSE
27 4         PERFORM REAL
28 3      ENDIF
29 3      ELSE
30 4         PERFORM REAL
31 3      ENDIF
32 1 END DECF
379 1 BEGIN INTEGER
380 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL
381 2 CONVERT NUMBER TO INTEGER
382 2 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
383 2 STORE INTEGER TOKEN IN COMBUF
384 2 INCREMENT WORDS IN COMBUF BY 2
385 2 INCREMENT TOKENS BY 1
386 1 END INTEGER
387 1 *
388 1 *
389 1 *
390 1 BEGIN REAL
391 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL
392 2 SET REAL = DOUBLE PRECISION * 10 ** POWER
393 2 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
394 2 STORE REAL TOKEN IN COMBUF
395 2 INCREMENT WORDS IN COMBUF BY 3
396 2 INCREMENT TOKENS BY 1
397 1 END REAL
398 1 *
399 1 *
400 1 *
401 1 BEGIN DBL
402 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL
403 2 SET DOUBLE = DOUBLE PRECISION * 10 ** POWER
404 2 ERREXIT IF OVERFLOW/UNDERFLOW IS DETECTED PERFORM OVFLOW
405 2 STORE DOUBLE TOKEN IN COMBUF
406 2 INCREMENT WORDS IN COMBUF BY 4
407 2 INCREMENT TOKENS BY 1
408 1 END DBL
409 1 *
410 1 *
411 1 *
412 1 BEGIN REPEP
413 2 ERREXIT IF THERE IS NO ROOM IN COMBUF FOR THIS TOKEN PERFORM CONFUL
414 2 ERREXIT IF NUMBER OF TIMES TO REPEAT IS ZERO PERFORM INVAL
415 2 STORE REPEAT TOKEN IN COMBUF
416 2 INCREMENT WORDS IN COMBUF BY 2
417 2 INCREMENT TOKENS BY 1
418 2 GET NEXT CHARACTER
419 1 END REPEP
BEGIN SCARS
    CHARACTERS TABLE:
        - X IS A CLOSED BRACKET
        - Y IS AN OPEN BRACKET
        - Z IS A BACK SLASH

    SET J=1
    START SEARCH WHILE J<=#CHARACTERS IN TABLE
    EXIT IF INPUT CHARACTER MATCHES CHARACTER (J) IN TABLE
    INCREMENT J BY 1
    END LOOP
    PERFORM INVAL - NO RETURN
    END SEARCH
    SET NEXT FLAG ON

CASE J:
    * = - # / < > $ @ = & ?
    ( )
    * ( )
    ( : F ; F ; F ; F ; F ; F ; F ; F ; F ; F ; B ; F ;
    A : ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;

: A:
    SET NEXT FLAG OFF
    PERFORM QUOTE

: B:
    SET RETURN CODE TO SAY EXTENDED PROMPT REQUESTED

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

: D:
    SET RETURN CODE TO SAY I ENTERED

: E:
    IF FOLLOWING CHARACTER IS A DIGIT THEN
        SET NEXT FLAG OFF
        SET INTEGER = 0
        SET POWER = 0
        PERFORM DECPT
    END IF

: F:
    END CASE
    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
    END IF
    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
ENDDO
ENDQUOTESTRING
END}
503 1 BEGIN INVAL
504 2  SET RETURN CODE TO SAY INVALID RESPONSE
505 2  SET END FLAG ON
506 1 EXIT XILAN
507 1 END INVAL
508 1 *
509 1 *
510 1 *
511 1 BEGIN COMFUL
512 2  SET RETURN CODE TO SAY COMBUF IS FULL
513 2  SET END FLAG ON
514 1 EXIT XILAN
515 1 END COMFUL
516 1 *
517 1 *
518 1 *
519 1 BEGIN OVFLOW
520 2  SET RETURN CODE TO SAY OVERFLOW/UNDERFLOW
521 2  SET END FLAG ON
522 1 EXIT XILAN
523 1 END OVFLOW
1 BEGIN XTPRM
2 PERFORM SETUP TO COMPLETE CONTROL TABLE AND INDEX TO APPROPRIATE ENTRY
3 DO UNTIL "NO CONTINUE" FOUND (O IN ENTRY CONTINUE FIELD)
4 IF OPEN SUCCESSFUL, THEN
5 POSITION TO INDICATED STARTING RECORD AND READ
6 EXIT TO :ERROR9: IF FAILURE
7 IF TABLE SIZE FIELD < 128 (NOT A LIST RECORD), THEN
8 IF SIZE > 0 (NO LAST CHARACTER MASKING & POSSIBLE SPANNING), THEN
9 IF RECORD SPANNED (N=SIZE > 128), THEN
10 READ SECOND RECORD AND APPEND TO FIRST RECORD DATA
11 EXIT TO :ERROR9: IF FAILURE
12 ENDIF
13 ELSE
14 SET SIZE POSITIVE
15 DO FOR EACH LIST ITEM (1=M)
16 BLANK LAST CHARACTER
17 ENDDO
18 ENDIF
19 IF LIST SEARCHING IS INDICATED (SEARCH FIELD = 1), THEN
20 START SEARCH WHILE ITEM ITEMS REMAIN TO BE EXAMINED
21 EXIT IF TOKEN LOCATED IN LIST
22 POSITION TO APPROPRIATE RECORD (I+L-1) AND READ
23 EXIT TO :ERROR9: IF FAILURE
24 SET SIZE TO 128 (ENTIRE RECORD TO BE DISPLAYED)
25 END LOOP
26 IF TABLE MESSAGE NUMBER FIELD > 0, THEN
27 CALL XRMNG TO DISPLAY "NOT VALID ..." MESSAGE
28 ENDIF
29 EXIT TO ENDDO
30 END SEARCH
31 ENSDF
32 PERFORM DISPLAY
33 ELSE OPEN ERROR
34 IF FILE NOT FOUND AND TABLE MESSAGE NUMBER FIELD > 0
35 CALL XRMNG TO DISPLAY "NOT VALID ..." MESSAGE
36 ELSE
37 :ERROR9: CALL XRMNG TO DISPLAY "FILE MANAGER ERROR " MESSAGE
38 CLOSE FILE
39 ENSDF
40 ENDDO
41 END XTPRM
BEGIN SETUP
1 EXIT TO ERROR: IF FIRST TOKEN NOT ? OR NAME FOLLOWED BY ?
2 CASE (:X:, :T:, */) SUBST
3 :X: IF FIRST TOKEN == ?, THEN
4 SET TABLE INDEX TO FIRST ENTRY
5 ELSE SET TABLE INDEX TO SECOND ENTRY
6 ENDIF
7 :S: IF FIRST TOKEN == ?, THEN
8 SET TABLE INDEX TO FOURTH ENTRY
9 ELSE SET TABLE INDEX TO SIXTH ENTRY
10 ENSIXTH ENTRY FILE NAME FROM > AND TOKEN
11 ENSIF
12 :S: IF PROMPT LENGTH == 0, THEN
13 ASSUME TABLE INDEX OF THIRD ENTRY
14 ELSE ASSUME TABLE INDEX OF FOURTH ENTRY
15 ENSIF
16 IF FIRST TOKEN IS A NAME, THEN
17 INCREMENT TABLE INDEX BY TWO ENTRIES (NOW FIFTH OR SIXTH)
18 ENSIXTH ENTRY FILE NAME FROM > AND TOKEN
19 ENSIF
20 :X: FROM TENTH ENTRY FILE NAME FROM > AND EDITOR CURRENT PROCESSOR NAME
21 ASSUME TABLE INDEX OF TENTH ENTRY
22 IF FIRST TOKEN == ?, THEN
23 IF PROMPT LENGTH == 0, THEN
24 CHANGE TABLE INDEX TO SEVENTH ENTRY
25 COPY FILE NAME FROM ENTRY TEN TO ENTRY EIGHT
26 ELSE CHANGE TOKEN TO KEYWORD CURRENTLY BEING PROCESSED BY EDITOR
27 ENSIF
28 ELSE
29 IF PROMPT LENGTH == 0, THEN
30 CHANGE TABLE INDEX TO NINTH ENTRY
31 ENSIF
32 ENSIF
33 ENSIF
34 ENSIF
35 ENSIF
36 ENSIF
37 ENSIF
38 ENSIF
39 ENSIF
40 ENSIF
41 ERROR: EXIT XTPRM WITH INVALID REQUEST FOR EXTENDED PROMPTING
42 ENSIF
43 ENSIF
44 ENSIF
45 ENSIF
46 ENSIF
47 ENSIF
48 ENSIF
49 ENSIF
50 ENSIF
51 ENSIF
52 ENSIF
53 ENSIF
54 ENSIF
55 ENSIF
56 ENSIF
57 ENSIF
58 ENSIF
59 ENSIF
60 ENSIF
61 ENSIF
62 ENSIF
63 ENSIF
64 ENSIF
65 ENSIF
66 ENSIF
67 ENSIF
68 ENSIF
69 ENSIF
70 ENSIF
71 ENSIF
72 ENSIF
73 ENSIF
74 ENSIF
75 ENSIF
76 ENSIF
77 ENSIF
78 ENSIF
79 ENSIF
80 ENSIF
81 ENSIF
683 1 BEGIN DISPLAY
684 2 IF SIZE < 128, THEN
685 3 DO UNTIL ALL LIST ITEMS DISPLAYED
686 4 BLANK LINE BUFFER
687 4 MOVE EIGHT (OR REMAINING) ITEMS INTO BUFFER
688 4 DISPLAY LINE
689 3 ENDDO
690 ELSE
691 DISPLAY EXTENDED PROMPT
692 ENDF
693 1 END DISPLAY
FORTRAN CALLING PROCEDURE

CALL XUDBG (I.U., ID)

**NOTE:**

XUDBG PROVIDES THE PROGRAMMER WITH A CALLABLE INTERACTIVE MEMORY

- DUMP AND/OR MODIFICATION FACILITY

**INPUT (CALLING SEQUENCE):**

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

**OPTION (S,M,E):**

- S - SNAP OUT (DUMP) MEMORY
- M - MODIFY MEMORY
- E - EXIT XUDBG

**START:**

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

**STOP:**

- OCTAL MEMORY ADDRESS OF LAST WORD TO BE SNAPPED

**OUTPUT UNIT:**

- LOGICAL UNIT NUMBER TO WHICH SNAP IS TO BE WRITTEN

**VALUES:**

- ARRAY OF OCTAL VALUES TO BE STORED IN MEMORY
- BEGINNING AT LOCATION "START". WHEN INPUT "VALUES" MUST NOT EXCEED 50 CHARACTERS. NULL FIELDS, I.E., SUCCESSIVE COMMAS, INDICATE WORDS OF ZERO TO BE STORED.

**Notes:**

- BASE - NUMBER BASE FOR PROMPT AND CONVERSION PROCEDURE
- CLASS - CLASS I/O NUMBER FOR TERMINAL INPUT
- LENGTH - LENGTH OF CHARACTER STRING BEING MANIPULATED
- LINE - EIGHT CCHAR BUFFER OF WORD TO BE SNAPPED
- LUI - TERMINAL LOGICAL UNIT WITH INPUT ECHO BIT SET
- LUO - LOGICAL UNIT FOR SNAP OUTPUT
- OPTN - ONE CHARACTER EXECUTION OPTION CODE
- ORIGIN - REFERENCE POINT FOR MEMORY ACCESS OFFSET COMPUTATION
- ADDRESS OF ORIGIN
- "ORIGIN" RELATIVE END OF DATA TO BE SNAPPED
- "ORIGIN" RELATIVE BEGINNING OF MEMORY TO BE ACCESSED
- "ORIGIN" CONVERSION
- RETURN POINT INDICATOR FOR INTERNAL PROCEDURE 'PROMPT AND
BEGIN XUG
WRITE SNAP HEADER TO TERMINAL
DO UNTIL OPTION IS EXIT (E)
   PROMPT TERMINAL FOR OPTION
   IF OPTION IS NOT EXIT (E)
      THEN
      PERFORM PROMPT AND CONVERSION FOR STARTING ADDRESS
      IF OPTION IS MODIFY (%)
         THEN
         PERFORM PROMPT AND CONVERSION FOR VALUES AND STORE IN MEMORY
      ELSE
         PERFORM PROMPT AND CONVERSION FOR ENDING ADDRESS
      ENDIF
      PERFORM PROMPT AND CONVERSION FOR OUTPUT UNIT
      WRITE SNAP HEADER TO OUTPUT UNIT
   END DO
   CALL XUDPL TO FORMAT LINE
   OUTPUT LINE
END IF
END DO
BEGIN PROMPT AND CONVERSION
ISSUE PROMPT TO TERMINAL, SOLICIT RESPONSE AND WAIT
RETRIEVE RESPONSE
CLEAR SUM
DO FOR EACH CHARACTER
SHIFT SUM AND ADD NEXT DIGIT
END DO
END PROMPT AND CONVERSION
GENERAL FILE DUMP PROGRAM FOR FILE MANAGER FILES

INPUT
NAME - NAME OF FN FILE TO BE DUMPED
IREC - LOGICAL RECORD NUMBER OF FIRST RECORD TO BE DUMPED
(First record is record number one)
NREC - NUMBER OF LOGICAL RECORDS TO DUMP
FMT - RUN TIME FORMAT FOR RECORDS (MAXIMUM OF 72 CHARACTERS) OR
BLANK INDICATING THE DEFAULT OF OCTAL AND ASCII DUMP TYPE
FOR PRINTING OR THE CHARACTERS OF INDICATING UNFORMATTED
OUTPUT
LU - LOGICAL UNIT NUMBER OF OUTPUT DEVICE

OUTPUT
FORMATTED DUMP OF THE INDICATE PORTION OF THE INDICATED FILE

NOTES
USES EXEC, MAXQ, OPEN, POINT, READF, XNPAR, XPRBS, XNNOV, XUDPF
ANY FILE WITH VARIABLE LENGTH RECORDS WILL BE DUMPED USING A
RECORD BUFFER OF 1024 WORDS THUS LIMITING THE MAXIMUM DUMPABLE
RECORD LENGTH.

BEGIN XUDPF
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
ENDIF
READ LU OF PRINT DEVICE
OPEN FILE
IF SUCCESSFUL THEN
DO FOR NUMBER OF RECORDS TO DUMP
READ RECORD
EXIT TO :ERROR: IF FAILED
FORMAT AND PRINT RECORD
ENDDO
ELSE
:ERROR: OUTPUT MESSAGE
ENDIF
ENDDO
END XUDPF
FORTRAN CALLING PROCEDURE

158 1 CD8  CALL XUDPL (ADDRESS, LINE, BUFFER)
159 1 CD8
160 1 CD8
161 1 CD8
162 1 CD8
163 1 CD8
164 1 CD8
165 1 CD8
166 1 CD8
167 1 CD8
168 1 CD8
169 1 CD8
170 1 CD8
171 1 CD8
172 1 CD8
173 1 CD8
174 1 CD8
175 1 CD8
176 1 CD8
177 1 CD8
178 1 CD8
179 1 CD8
180 1 CD8
181 1 CD8
182 1 CD8
183 1 CD8
184 1 CD8
185 1 CD8
186 1 CD8
187 1 CD8
188 1 CD8
189 1 CD8
190 1
191 1
192 1
193 1
194 1
195 1
196 1
197 1
198 1
199 1
200 1
201 1
202 1
203 1
204 1
205 1

INPUT

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

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

BUFFER - FIFTY-ONE WORD BUFFER TO HOLD FORMATTED PRINT LINE. MUST
BE BLANKED PRIOR TO FIRST CALL TO XUDPL AND NOT STORED
INTO BETWEEN CALLS TO XUDPL.

OUTPUT

BUFFER - FIFTY-ONE WORD BUFFER CONTAINING FORMATTED LINE

COLUMNS CONTENTS

FIRST ADDRESS

SECOND ADDRESS

OCTAL REPRESENTATION OF 'LINE'

ASCII REPRESENTATION OF 'LINE'

NOTES

USES XREX, XRGE, XRSET

BEGIN XUDPL

CALL XRGE TO CONVERT EACH WORD OF ADDRESS TO OCTAL

DO FOR EACH WORD OF LINE

CALL XRGE TO CONVERT WORD TO OCTAL

DO FOR EACH BYTE OF WORD

IF BYTE < 40 OR BYT > 1368

THEN

REPLACE BYTE WITH ASCII PERIOD

END IF

END DO

END XUDPL
CD1  FORTRAN MAIN PROGRAM XUFTM IS SCHEDULED BY XU ON M TO PRINT
CD1  A PARTITION DUMP WHICH HAS BEEN WRITTEN TO DISK
CD1
CD1  END PROGRAM XUFTM
CD1
CD1
CD1
CD1

CD2  INPUT
CD2  ICWT - 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
CD2  END INPUT

CD3  OUTPUT
CD3  FORMATTED DUMP TO LU 6

CD4  END OUTPUT

CD5  EXTERNAL REFERENCES

CD6  EXEC
CD7  RMPAR
CD8  XIRCPY
CD9  XRVNO
CD10  XUDP

CD11  END EXEC

CD12  BEGIN XUFTM
CD13  CALL RMPAR TO GET LU AND STARTING TRACK NOS.
CD14  READ 1ST TRACK -- 1ST 12 WORDS ARE HEADER
CD15  COUNT OF ID-SEGS TO BE DUMPED
CD16  UP TO 7 ID-SEGMENT ADDRESSES
CD17  LOW AND HIGH BASE PAGE ADDRESSES
CD18  LOW AND HIGH MAIN MEMORY ADDRESSES
CD19  POINT TO 2ND SECTOR OF DUMP DATA
CD20  DO UNTIL ALL ID-SEGS PRINTED
CD21  DO UNTIL A EIGHT-WORD LINES PRINTED
CD22  PRINT 1 LINE AND INCREMENT POINTER AND ADDRESSES TO NEXT
CD23  END DO
CD24  END DO
CD25  END DO
CD26  END DO
CD27  END XUFTM
REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR
01 TYPE 1A 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 XVSTB
05 CALL XVPAM(PARMS)
06 ASSEMBLY FORM
07 JSB XVPAM
08 DEF +2 RETURN ADDRESS
09 DEF PARMS A(PARMS)
10 WHERE PARMS ARE P1,P2,P3,P4,P5
11 P1 IS THE SERVICE REQUEST
12 0= NORMAL TERMINATION (P2-P5 NOT USED)
13 1= WORK AREA REQUEST (P2-P5 NOT USED)
14 2= EXECUTE A SEQUENCE TABLE
15 (P2-P4 HAS TABLE NAME)
16 (P5 INDICATES EXECUTION CONTROL IN CLASS I/O BUFFER)
17 3= RESET SEQUENCE POINTERS
18 (P2 HAS SEQUENCE NUMBER)(P3-P5 NOT USED)
19 8= TERMINATE SEQUENCE (P2-P5 NOT USED)
20 9= TERMINATE FDS FUNCTION (P2-P5 NOT USED)
21 -32767= ABNORMAL TERMINATION OF ASSOCIATED TASK
22 FROM AN FDS MANAGER
23 ASSEMBLY FORM
24 JSB XVPAM
25 DEF (RETURN POINT)
26 OCT 0
27 DEF PARMS A(FDS MANAGER RESPONSE)
28 OUTPUTS
29 TO AN FDS MANAGER
30 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
31 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
32 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
33 MANAGER IS ACTIVATED VIA SLIST
34 TO AN ASSOCIATED TASK
35 REQUEST PARMS (P1-P5) MOVED INTO ASSOCIATED TASK ID-SEGMENT
36 RETURN ADDRESS IS MOVED INTO ID-SEGMENT WORD 9(XSUSP)
37 CURRENT ID-SEGMENT IS MOVED INTO GENERAL WAIT VIA SLIST
38 ASSOCIATE TASK IS ACTIVATED VIA SLIST
122 1 BEGIN XVPMAN
123 2 * DETERMINE REQUESTORS MANAGER BY USING FATHER ID NUMBER
124 2 * FIELD IN ID SEGMENTS AS A BACKWARD CHAIN
125 2 SET TARGET-ID FROM CURRENT-ID-SEGMENT
126 2 DO WHILE FATHER-ID-NUMBER .NE. 0 OR FATHER IS WAITING
127 3 COMPUTE FATHER-ID-SEGMENT FROM FATHER-ID-NUMBER IN TARGET-ID
128 3 PERFORM MGRFND(FATHER-ID-SEGMENT,COUNT)
129 3 EXIT IF COUNT .GT. 0
130 3 SET TARGET-ID TO FATHER-ID-SEGMENT
131 3 ENDDO
132 2 IF FATHER-ID-NUMBER .EQ. 0, OR FATHER NOT WAITING THEN
133 2 CALL SYMNG (12,*XV03,SEGMENT-NAME) 'XV03,NAME' REQUESTING PROG
134 2 PERFORM PUMP
135 2 EXIT TO $LIX TO ENABLE AND REDISPATCH
136 2 ENDF
137 2 SET AWAKEN-ID FROM FSD-ENTRY SYMNG
138 2 GET REQUEST PAKS MOVE INTO ID-SEGMENT
139 2 SET STBAT FROM CURRENT-ID
140 1 FIND XVPMAN
141 1 *
142 1 *
143 1 *
144 1 *
145 1 BEGIN XVPMAN
146 2 * DETERMINE IF CALLER IS A VALID FDS MANAGER
147 2 *
148 2 **PERFORM MGRFND (CURRENT-ID,COUNT)
149 2 **COUNT WILL BE 0 FOR NO MATCH.
150 2 **COUNT NOT EQUAL ZERO IMPLIES A MATCH
151 2 **AND FSD-ENTRY HAS MATCHING FSDTAB ENTRY ADDRESS
152 2 **IF COUNT .LT. 0 THEN CALLER IS NOT A FDS MANAGER
153 2 **CALL SYMNG (12,*XV01,SEGMENT-NAME) 'XV01 PROGRAM' REQUESTING PROG.
154 2 PERFORM PUMP
155 2 EXIT TO $SKW THE DISPATCHER
156 2 ENDF
157 2 FDSTAB-ENTRY HAS ENTRY FOR RESPONDING MANAGER
158 2 * SET AWAKEN-ID-SEGMENT FROM CURRENT-ASSOCIATED-TASK
159 2 IF CURRENT-ID NOT WAIT THEN IT WAS NOT ON WAIT LIST
160 2 CALL SYMNG (12,*XV02,SEGMENT-NAME) 'XV02 PROGRAM' ASSOCIATED PROG.
161 2 PERFORM PUMP
162 2 EXIT TO $SKW THE DISPATCHER
163 2 ENDF
164 2 AWAKEN-ID-SEGMENT-STAT
165 2 IF MANAGER HAS REQUEST FOR ABORT, THEN
166 2 CALL SBART FOR CURRENT AF
167 2 ENDF
168 2 MOVE FDS MANAGERS INPUT PARTS TO ASSOCIATED TASK ID SEGMENT
169 2 MOVE PARTS TO ID-SEGMENT WORDS 2-6
170 1 END XVPMAN
BEGIN PDUMP
*
SET A CALL TO XVABM AT CALLER'S SUSPEND POINT
*
END PDUMP
*
BEGIN MGRFND INPUT IS TARGET-ID
*
GET XVSTB TABLE OF ACTIVE FDS MANAGERS
*
SET COUNT TO NUMBER IN TABLE NUMBER MANAGER
*
SET FSTAB-ENTRY TO FIRST-ENTRY(STBES)
*
DO WHILE COUNT.GT.0
*
EXIT IF TARGET-ID.EQ.FDS-MANAGER-ID-SEGMENT
*
SET FSTAB-ENTRY TO FSTAB-ENTRY + STBEL
*
SET COUNT = COUNT - 1
*
ENDO
*
IF FSTAB-ENTRY HAS FOUND MANAGER ENTRY IF COUNT.GT.0
*
COUNT = D MEANS NO MATCH
*
END MGRFND
*
BEGIN XVSTB
*
N EQU 3 EQUATE FOR NUMBER OF ENTRIES IN XVSTB
*
STBNM DEF N NUMBER-IN-TABLE
*
STBC DEF O NUMBER-ACTIVE
*
STBN DEF O STB RESOURCE NUMBER
*
STES EQU * ENTRY START
*
STBLU DEF O L(U(CSTACK))
*
STBLA DEF O L(U(CSTACK))
*
STBMG DEF O A(FDS-MANAGER-ID-SEGMENT)
*
STBEX DEF O A(FDS-EXECUTIVE-ID-SEC*1/2)
*
STBEC DEF O CLASS NUMBER FOR EXEC
*
STBAT DEF O A(CURRENT-ASSOCIATED-TASK)
*
STBPC DEF O CLASS-NUMBER FOR PROCESSOR
*
STBEC DEF O ENTRY SAVE AREA
*
STBEE EQU * ENTRY "END"
*
STBEL EQU STBEE-STBES ENTRY LENGTH
*
STBNE EQU N NUMBER-OF-ENTRIES
*
ORG STBES
*
REPN N DO FOR N ENTRIES
*
REPS STBEL DEFINE STORAGE WORDS FOR AN ENTRY
*
END XVSTB
*
FORTRAN CALLING PROCEDURE:

CALL XXAUT

XXAUT HANDLES AUTOMATIC EXECUTION WITHOUT TRACE

INPUTS IN COMMON:

XE(5) MASSTA, XE(10) SERSTR, XE(11) SEREND, XE(12) SERPTR,
XE(140)TABEND, XE(1) 'HOPROC, XE(2) LIBD,
XE(249) SENNO, XE(250) SENLEN, XE(251) SERTAB

OUTPUTS IN COMMON:

XE(5) MASSTA, XE(1) FLGTAB

COMMON USED:

(EQUIVALENCE (XE(5), MASSTA)
+ (XE(10), SERSTR), (XE(11), SEREND)
+ (XE(12), SERPTR), (XE(140), TABEND),
+ (XE(249), SENNO), (XE(250), SENLEN),
+ (XE(251), SERTAB),
+ (XE(253), FLGTAB)

FBS ROUTINES CALLED:

XECPR, XEXT, XRMAY, XRMSE,
XSEX, XISTO, XITMP

RTE ROUTINES CALLED:

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

CALL 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 (X5, XASTA)

FBS ROUTINES USED

XERTH, XEXIT, XVMSG, XIHAN

NOTE: CONTAINS DUMMY CALL TO XEXEC
132 1 CD************
133 1 CD
134 1 CD  FORTRAN CALLING SEQUENCE:
135 1 CD
136 1 CD  CALL XXDEC (RETC)
137 1 CD
138 1 CD************
139 1 CD
140 1 CD  XXDEC DECODES A RESPONSE OF PROCESSOR NAME (,INT TABLE NAME)
141 1 CD  INTO A SEQUENCE TABLE ENTRY.
142 1 CD
143 1 CD************
144 1 CD
145 1 CD  INPUTS IN COMMON:
146 1 CD
147 1 CD  XE(85) TOKENS, XE(145) COMBUF, XB(1) NOPROC, XB(2) LIBD
148 1 CD
149 1 CD************
150 1 CD
151 1 CD  OUTPUTS IN CALLING SEQUENCE:
152 1 CD
153 1 CD  RETC - RETURN CODE (O IS NORMAL RETURN)
154 1 CD
155 1 CD  OUTPUTS IN COMMON:
156 1 CD
157 1 CD  XE(16) PRCNAM, XB(251) SERTAB
158 1 CD
159 1 CD************
160 1 CD
161 1 CD  COMMON USED:
162 1 CD
163 1 CD  EQUVALENCE (XE(16), PRCNAM),
164 1 CD  + (XE(85), TOKEHS), (XE(144), TOKPTR),
165 1 CD  + (XE(145), COMBUF),
166 1 CD  + (XB(1), NOPROC), (XB(2), LIBD ),
167 1 CD  + (XB(251), SERTAB)
168 1 CD
169 1 CD  FDS ROUTINES USED:
170 1 CD
171 1 CD  XRCP#, XREXT, XRMOV, XRSIG
172 1 CD
173 1 CD  RTE ROUTINES USED:
174 1 CD
175 1 CD  IAND
176 1 CD
177 1 CD************
179 1 BEGIN XXDEC
180 2 INITIALIZE RETURN CODE TO ZERO
181 2 SET SEQUENCE ENTRY TO ZEROS
182 2 ERREXIT IF FIRST TOKEN IS NOT A PROCESSOR NAME TO :ERR1:
183 2 SEARCH LIBRARY DIRECTOR FOR PROCESSOR
184 2 ERREXIT IF NAME NOT FOUND TO :ERR1:
185 2 MOVE PROCESSOR NAME, IT BIT AND VERSION INTO SEQUENCE ENTRY
186 2 IF AN INTERFACE TABLE NAME WAS ENTERED THEN
187 3 MOVE INTERFACE TABLE NAME INTO SEQUENCE ENTRY
188 2 ENDF
189 2 ERREXIT IF LAST TOKEN IS NOT EOS TO :ERR1:
190 2 ERREXIT IF INTERFACE TABLE IS SPECIFIED WHEN NOT NEEDED TO :ERR1:
191 2 IF AN INTERFACE TABLE IS REQUIRED BUT NOT SPECIFIED THEN
192 3 SET INTERFACE TABLE IN SEQUENCE ENTRY TO 'SINTAB'
193 2 ENDF
194 1 EXIT XXDEC
195 2 :ERR1:
196 2 CALL XRNSG TO DISPLAY ERROR
197 2 SET RETURN CODE TO SAY ERROR
198 1 END XXDEC
001 CD***************
002 CD0 FORTRAN CALLING PROCEDURE:
003 CD0 CALL XXDEF
004 CD0 ***************
005 CD0 ***************
006 CD0 READS IN THE DEFAULT INTERFACE TABLE FOR A PROCESSOR
007 CD0 AND STORES IT IN THE ASA AS BITAB
008 CD0 ***************
009 CD0 ***************
010 CD0 INPUTS FROM COMMON:
011 CD0 ***************
012 CD0 ***************
013 CD0 ***************
014 CD0 ***************
015 CD0 ***************
016 CD0 ***************
017 CD0 ***************
018 CD0 ***************
019 CD0 ***************
020 CD0 ***************
021 CD0 ***************
022 CD0 ***************
023 CD0 ***************
024 CD0 ***************
025 CD0 ***************
026 CD0 ***************
027 CD0 ***************
028 CD0 ***************
029 CD0 ***************
030 CD0 ***************
031 CD0 ***************
032 CD0 ***************
033 CD0 ***************
034 CD0 ***************
035 CD0 ***************
036 CD0 ***************
037 CD0 ***************
038 CD0 ***************
039 CD0 ***************
040 CD0 ***************
041 CD0 ***************
042 CD0 ***************
043 CD0 ***************
044 CD0 ***************
045 CD0 ***************
046 CD0 ***************
047 CD0 ***************
BEGIN XXE

INITIALIZE RESET INDEX TO ZERO

IF TABLE FLAG SAYS SEFTAB IN XE THEN
  CALL XVPAW WITH SEFTAB IN XE
ELSE - AM BEING CALLED TO EXECUTE A TEMPORARY SEQUENCE TABLE
  CALL XVPAW WITH SEQUENCE POINTER AS START AND END
ENDIF

CALL RMPAR TO RETRIEVE RETURN PARAMETERS

IF THE MANAGER DETECTED A 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 (:INIT1:;:ERR11;:INTZ;:ERR2;:ERR3;:RESET;:ERR4;:ERR5;:ERR6:)

:INIT1: INTERFACE TABLE NOT SPECIFIED
  CALL XXXDEF TO READ UP DEFAULT TABLE
  CALL XXMP TO SET UP A TEMPORARY ENTRY
  PERFORM XXXE TO EXECUTE **NO RETURN**

:ERR1: SET MESSAGE TO INTERFACE TABLE NOT FOUND

:INTZ: INTERFACE TABLE NOT COMPLETE
  SET SUBSTATE TO INTERFACE TABLE EDITOR
  CALL XERTN TO RETURN TO EXEC **NO RETURN**

:ERR2: SET MESSAGE TO INT TABLE NOT FOR PROCESSOR

:ERR3: SET MESSAGE TO VERSIONS DO NOT MATCH
  SET: RESET REQUESTED ON TERMINATION
  CONVERT SEQUENCE # TO INDEX
  SAVE INDEX AND SEQUENCE NUMBER IN COMMON
  EXIT XXXE

:ERR4: SET MESSAGE TO RESET SEIQ NOT FOUND

:ERR5: SET MESSAGE TO PROCESSOR ABENDED

:ERR6: SET MESSAGE TO VAW OVERFLOW

ENDCASE

ENDCASE

RING MSG TO DISPLAY ERROR

FORMAT SEQUENCE ENTRY INTO ASCII
  CALL XRSNG TO DISPLAY SEQUENCE ENTRY IN ERROR
  CALL XRSNG TO DISPLAY SEQUENCE TABLE ENTRY IN ERROR
  IF MODE IS SEMI-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 XERTN TO ABORT SEQUENCE *** NO RETURN ***
ENDIF

ENDIF

RETURN

END XXE
1 BEGIN XXMAN
2 IF ENTRY IS FROM A DIRECTIVE THEN
3 SET HASITA TO INDICATE RE-ENTRY
4 DO UNTIL PERCENT IS ENTERED
5 :PROMPT: CALL XICOM TO PROMPT FOR PRNAME*,ITNAME
6 IF PERCENT IS NOT ENTERED THEN
7 ERREXIT IF CR ENTERED TO :PROMPT:
8 CALL XIDEC TO DECODE RESPONSE
9 ERREXIT IF INVALID RESPONSE TO :PROMPT:
10 SET SER #5 IN :BYTE 0 ZEROS
11 SET #ENTRIES IN SERTAB IN XE TO 1
12 CALL XXSTO TO STORE SEQUENCE TABLE
13 IF NAMES IN SERTAB IS #INTAB THEN
14 PROCESSOR REQUIRES AN II THEN
15 CALL XXDEF TO READ UP DEFAULT INTERFACE TABLE
16 ENDIF
17 CALL XXEXE TO EXECUTE SERTAB
18 ENDIF
19 ELSE
20 CALL XXMP TO SET UP TO EXECUTE A TEMPORARY TABLE
21 PERFORM XXMAN **NO RETURN**
22 ENDIF
23 RETURN
24 :PRMERR: CALL XRMSG TO DISPLAY ERROR
25 PERFORM XXMAN **NO RETURN**
26 END XXMAN
**FORTRAN CALLING SEQUENCE:**

CALL XXSEM

**INPUTS IN COMMON:**
- XE(5) HASSTA, XE(10) SERSTA, XE(11)adera, XE(12) SEQ'TA,
- XE(140) TABEND, XE(1) NOPROC, XE(2) LIBD,
- XE(250) SERNO, XE(250) SERLEN, XE(250) SERTAB

**OUTPUTS IN COMMON:**
- XE(5) MASSTA, XE(13) FLGTAB

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

**COMMON USED:**
- EQUIVALENCE (XE(5), MASSTA),
- + (XE(10), SERSTA), (XE(11), SEREND),
- + (XE(12), SERPROC), (XE(85), EOS ),
- + (XE(86), INTGR ), (XE(100), AMER ),
- + (XE(139), EXEND ), (XE(140), TABEND),
- + (XE(141), CURIND), (XE(145), COMBUS),
- + (XE(17), NOPROC), (XE(22), LIBD ),
- + (XE(235), RESIND),
- + (XE(236), ASCENT), (XE(46), OLDIND),
- + (XE(240), SERNO), (XE(50), SERLEN),
- + (XE(249), SERTAB),
- + (XE(133), FLGTAB)

**FDS ROUTINES USED:**
- XRPCR, XREXT, XRGB, XRNND, XRN56, XRSRT, XXDEF, XXEFS, XXSTO, XRSMP, XTCCM
BEGIN XXSEM:
  IF ENTRY IS FROM A DIRECTIVE THEN
  SET MASTER STATE TO SAT REENTRY
  DO FOR # ENTRY, IN SEQUENCE TABLE
    SEARCH LABEL FOR PROCESSOR NAME
    ERREXIT IF NOT FOUND TO :ERRT:
    MOVE IT BIT AND VERSION INTO SEQUENCE TABLE ENTRY
  ENDIF
  SAVE ORIGINAL ENDING SEQUENCE #
  SET CURRENT SEQUENCE # TO BEGINNING SEQUENCE #
  SET FLTAB TO SAT EXECUTE ENTER SRTAB
  CALL XISTO TO STORE RESET IN AW:
  ELSE (I AM BEING REENTERED FROM INTE)
    CALL XTRP TO SETUP TEMPORARY EXECUT IN
    CALL XHERE TO EXECUTE ONLY THE FIRST ENTRY OF SRTAB
    IF RESET WAS REQUESTED THEN
      SET CURRENT SEQUENCE # TO NEW SEQUENCE #
    ELSE
      CALL XEXIT IF TERMINAL ENTRY WAS JUST EXECUTED
      SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
    ENDIF
  ENDIF
DO UNTIL TERMINAL ENTRY IS EXECUTED AND IT IS NOT A REQUEST TO RESET
CALL XTCM TO PROMPT USER WITH CURRENT ENTR Y
EXIT XXSEM IF RESPONSE IS X
  IF RESPONSE IS CR THEN
    IF THIS IS AN OVERRIDE WITH DEFAULT INTERFACE TABLE THEN
      CALL XDEF TO READ UP DEFAULT TABLE
    ENDF
    IF BEGINNING SEQUENCE # TO CURRENT SEQUENCE #
    IF ENDING SEQUENCE NUMBER TO BEGINNING SEQUENCE #
    CALL XHERE TO EXEC
    IF CURRENT SEQUENCE # TO NEXT SEQUENCE #
    ELSE
      IF RESPONSE WAS A SEQUENCE # THEN
        ERREXIT IF NUMBER IS ZERO TO :ERRT:
        SEARCH SEQUENCE TABLE FOR SEQUENCE #
        ERREXIT IF NUMBER IS NOT FOUND TO :ERRT:
        SET CURRENT SEQUENCE # TO SEQUENCE # REQUESTED
      ELSE IF RESPONSE IS AN ANDewhat THEN
        SET CURRENT SEQUENCE # TO NEXT SEQUENCE #
        ELSE (RESPONSE MUST HAVE BEEN AN OVERRIDE)
        CALL XEDC TO DECODE PROCESSOR NAME, IT NAME RESPONSE
        IF RESPONSE IS VALID THEN
          CALL XISTO TO STORE OVERRIDING ENTRY
          SAVE CURRENT SEQUENCE # IN OLD SEQUENCE #
          SET CURRENT SEQUENCE # TO FIRST ENTRY
        ENDF
      ENDF
ENDIF
ENDIF
ENDDO
EXIT XXSEM
:ERRT:
CALL XXMSG TO DISPLAY ERROR MESSAGE
END XXSEM
CALLING PROCEDURE FOR XXSTO:

CALL XXSTO

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

INPUTS FROM COMMON:

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

OUTPUTS TO COMMON:

`XX(5) MASSTA`

COMMON USED:

`EQUIVALENCE (XX(5), MASSTA),`

`+ (XREP(19), REP(18), XX(20), REP(21), REBUF),`

`+ (XB(250), SEGLEN), (XB(251), SERTAB),`

`+ (XX(13), FLGTAB)`

FDS Routines Used:

`XXRQ, XRMOV, XRMIG, XRTMN`

RTE Routines Used:

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

CALL XITMP

**XITMP SETS UP A ONE ENTRY SEQUENCE TABLE USING EINTAB AND STORES IT IN THE AMA TO EXECUTE WHEN EXECUTION WITH A TEMPORARY ENTRY IS NEEDED.**

**INPUTS FROM COMMON:**

XE(12) SEPTR, XB(250) SEQLEN, XB(251) SECTAB

**OUTPUTS TO COMMON:**

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

**COMMON USED:**

**EQUIVALENCE (XE(6), SUBSTA), (XE(12), SEPTR),**

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

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

+ (XB(251), SECTAB), (XS(6), TMPTAB),

**FDS Routines Used:**

XRED, XMNOV, XSSDA

**RTE Routines Used:**

EXEC
ASSGN - DATA ASSIGNMENT PROCESSOR

ASSGN - SCHEDULED BY FD5

ASSGN ALLOWS THE FD5 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 REPEATED EVALUATIONS IN ORDER TO COMPUTE AND STORE MULTIPLE VALUES.

INPUTS FROM THE MANAGER:

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

INPUTS FROM THE INTERFACE TABLE:

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

OUTPUTS TO THE ANA:

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

INTERNAL VARIABLES:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIMENSION</th>
<th>START</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARM</td>
<td>5</td>
<td>1</td>
<td>PARM(1)= LU, PARM(2) = DEBUG FLAGS</td>
</tr>
<tr>
<td>TOKER</td>
<td>32</td>
<td>6</td>
<td>IDENTIFYING NUMBERS FOR TOKENS</td>
</tr>
<tr>
<td>STWDW</td>
<td>1</td>
<td>38</td>
<td>SYMBOL TABLE WIDTH</td>
</tr>
<tr>
<td>STLON</td>
<td>1</td>
<td>39</td>
<td>SYMBOL TABLE LENGTH</td>
</tr>
<tr>
<td>LASTY</td>
<td>1</td>
<td>40</td>
<td>LAST SYMBOL TABLE ENTRY DEFINED</td>
</tr>
<tr>
<td>SYMTAB</td>
<td>12,81</td>
<td>41</td>
<td>SYMBOL TABLE (WORDS 1-8 = TOC ENTRY OR APPLICABLE INFORMATION, WORDS 9-11 = VALUE, WORD 12 = 1 FOR INDEX, = 2 FOR SUBSCRIPTED DATA ELEMENT)</td>
</tr>
<tr>
<td>SSTNG</td>
<td>247</td>
<td>1013</td>
<td>RESULT STACK USED DURING POST-FIX STRING EVALUATION (EACH EN-</td>
</tr>
<tr>
<td>RESULT</td>
<td>4,35</td>
<td>1260</td>
<td>TYP: WORD 1-3 CONTAIN VALUE; WORD 4 = DATA TYPE). DATA TYPE=</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1-2,3 FIXED DATA</td>
</tr>
</tbody>
</table>

-1 SYMBOL TABLE INDEX
<table>
<thead>
<tr>
<th>No.</th>
<th>CD4</th>
<th>Syntax</th>
<th>Token</th>
<th>Input Priority</th>
<th>Output Priority</th>
<th>Token #OPRnds-1</th>
<th>Data Type</th>
<th>Precedent Type</th>
<th>Valid Precedents</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>CD4</td>
<td>7x40 Table</td>
<td>CONTAINING INFORMATION FOR PROCESSING EACH TOKEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>CD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **OP:** Operator
- **LB:** Left Bracket
- **RB:** Right Bracket
- **CM:** Constant
<table>
<thead>
<tr>
<th>NAME(WORDS 1-3)</th>
<th># OPERANDS-1</th>
<th>OUTPUT TYPE</th>
<th>FIRST OP TYPE</th>
<th>SECOND OP TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>ALG</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>ALOGT</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>AMOD</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>ATAN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>ATAN2</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>COS</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DBS</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATAN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DINT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DLOG</td>
<td>0</td>
<td>DOUBLE</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DMOD</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DLOGT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DSIGN</td>
<td>1</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DSIN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DSHORT</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DTAN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DTANN</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FABS</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IDINT</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IFIX</td>
<td>0</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>IISIGN</td>
<td>1</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MOD</td>
<td>1</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SIGN</td>
<td>1</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>SIN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SMGL</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SORT</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TAN</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>TANH</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>0</td>
<td>REAL</td>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DEXP</td>
<td>0</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
</tbody>
</table>
Routines Referenced - RMPAR, XPCST, XRMOV, XI?S1, XIPS2, XPXIT

BUSCH-NAUR Language Definition

<Assignment> ::= <Replacement> <Range> //
<Evaluation> ::= <Range>

<Replacement> ::= <Non-Numeric De> X <Non-Numeric De> //
<Non-Numeric De> = "Character String" //
<Variable> = Free Data Element

<Non-Numeric De> ::= Free Data Element //
<Free Data Element> = <Subscript List> //
<Character Data Element> //
<Free Data Element> = <Subscript List> = / 
<Expression>

<Variable> ::= Fixed Data Element //
<Fixed Data Element> = <Subscript List>

<Subscript List> ::= <Subscript List>, <Expression> //
<Expression> //
<Expression>

<Expression> ::= <Expression> <Additive Operator> <Term> //
<Unary Operator> <Term>
<Term> ::= <Term> <Multiplicative Operator> <Factor> //
<Factor>

Factor ::= <Power>**<Power> //
<Power>

<Power> ::= (<Expression>) //</Operand>
<Operand>
<Additive Operator> ::= + // -
<Multiplicative Operator> ::= * // /
<Operando> ::= Function Name <LB> <Function List> <RB> //</Variable>
<Constant>
<Left Bracket> <RB> //</Right Bracket>

<Function List> ::= <Function List>, <Expression> //</Expression>
<Expression>
DBDSP - DATA BOX DISPLAY PROCESSOR

- SCHEDULED BY FDS

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

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

- XDIM - READS AND EDITS INTERFACE TABLE
- XDPI - READS ORCHE A, VALIDATES NAMES IN INTERFACE TABLE AGAINST NAMES IN DATABASE SCAN. WITH SUBROUTINE XDIMK IT DEVELOPS CONSTRAINT MASKS FOR ARRAYS
- XDIP2 - PROMPTS USER IF REQUIRED AND OUTPUTS REQUESTED PAGE ARRAYS OF UP TO TWO VARIABLES TO THE SPECIFIED LU DEVICES WITH SUBROUTINE XDIPD

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

INPUTS TO DBDSP FROM INTERFACE TABLE

1. DATABASE - DATA BOX FILE NAME
2. NOVAR - DISPLAY VARIABLE NAME LIST SET UP BY USER
3. KEYP - DISPLAY VARIABLE SCALE LIST SET UP BY USER
4. VIDEF - CONSTRAINT VARIABLE DEFINITION LIST

INPUTS TO DBDSP FROM ORDE FILE

RECORD 1

1. NAME OF FDS PROCESSOR CREATING FILE
2. NAME OF DATABASE VARIABLE NAME FOR THIS FILE
3. NAME OF FDS PROCESSOR UPDATING FILE
4. (3 ASCII WORDS OF BLANKS)
5. INTERFACE TABLE VARIABLE NAME FOR THIS U-DATE
6. (3 ASCII WORDS OF BLANKS)

RECORD 2

1. (IN - NO OF ENTRIES IN SUMMARY TABLE)
2. (X SCAN VARIABLE (6 CHAR))
3. (F 5:2: SUBSCRIPT (INT OR ZERO))
4. (XSCAND SUBSCRIPT (INT OR ZERO))
5. (X UNITS (6 CHAR))
6. (X CENTROID (REAL))
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>338</td>
<td>(12) - X INCREMENT (REAL)</td>
</tr>
<tr>
<td>339</td>
<td>(14) - X NUMBER OF STEPS (INTEGER 1-5)</td>
</tr>
<tr>
<td>340</td>
<td>(15) - YSCAN VARIABLE (6 CHAR)</td>
</tr>
<tr>
<td>341</td>
<td>(18) - Y FIRST SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>342</td>
<td>(19) - Y SECOND SUBSCRIPT (INT OR ZERO)</td>
</tr>
<tr>
<td>343</td>
<td>(20) - Y UNITS (6 CHAR)</td>
</tr>
<tr>
<td>344</td>
<td>(23) - Y CENTROID (REAL)</td>
</tr>
<tr>
<td>345</td>
<td>(25) - YX INCREMENT (REAL)</td>
</tr>
<tr>
<td>346</td>
<td>(27) - Y NUMBER OF STEPS (INTEGER 1-5)</td>
</tr>
</tbody>
</table>

DEPANANT VARIABLE NAME AND UNITS IN RECORDS 32 48 AND 5

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>SUMMARY TABLE RECORDS</td>
</tr>
<tr>
<td>351</td>
<td>- EACH SUMMARY TABLE CONTAIN VALUE FOR EACH</td>
</tr>
<tr>
<td>352</td>
<td>DEPENDANT VARIABLE SCANNED (UP TO 32 VALUES)</td>
</tr>
<tr>
<td>353</td>
<td>INCLUDING ERROR FLAG WHICH IS FIRST VALUE</td>
</tr>
<tr>
<td>354</td>
<td>IN SUMMARY TABLE)</td>
</tr>
</tbody>
</table>

OUTPUT FROM DBDSP

1 CD 3 - DISPLAY IS OUTPUT TO LU IDENTIFIED IN THE INTERFACE TABLE

1 CD 4 - COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XPRM
1 CD 4 - DATABX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
1 CD 4 - DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
1 CD 4 - IDV1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 1ST VAR
1 CD 4 - IDV2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR
1 CD 4 - ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XMSK (MAX OF 8)
1 CD 4 - ITSCN1 - FIRST SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
1 CD 4 - ITSCN2 - SECOND SUBSCRIPT FOR NAME OF X SCAN VARIABLE (INT OR 0)
1 CD 4 - ITSCN3 - SECOND SUBSCRIPT FOR NAME OF Y SCAN VARIABLE (INT OR 0)
1 CD 4 - LEMBF - LENGTH OF COMBUF IN 16-BIT WORDS - XPRM
1 CD 4 - LEMST - LENGTH IN WORDS OF CHAR STRING USED FOR USER PROMPT -XPRM
1 CD 4 - MASK - ARRAY CONTAINING MASKS FOR UP TO 8 CONSTR (4 X 121 SIZE)
1 CD 4 - NAMEVR - NAME LIST FOR VARIABLES SCANNED BY SCAN-ENDSCN
1 CD 4 - NC - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
1 CD 4 - NCRLS - LIST OF CONSTRAINT RELATIONS INPUT BY USER
1 CD 4 - NCVARL - LIST OF CONSTRAINT VARIABLE NAMES INPUT BY USER (32 MAX)
1 CD 4 - NDISP - NUMBER OF DISP VARIABLE PAIRS FOR PAGED OUTPUT[1-16P]
1 CD 4 - NDVRCT - NUMBER OF DEP DISP VAR IN NDVARL LIST (INTEGER)
1 CD 4 - NDVRUL - LIST OF DEP DISP VARIABLE PAIR SCALE FACTORS FOR 0P
1 CD 4 - NPMAX - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT = 0
1 CD 4 - NSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1 CD 4 - NYSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1 CD 4 - RETC - RTN TONE IF PROMPT - COMMON PORT TO EXEC.PROMPT GBR3ERR
1 CD 4 - SCORD - CHAR STRING CONTAINING USER PROMPT MESSAGE
1 CD 4 - XORD - LIST OF X VAR VALUES FOR X COORDINATES (1 - 11 REAL)
1 CD 4 - XSCAN = NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
DEFIN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

DEFIN ALLOCATES DATA ELEMENTS IN THE AWA THAT WERE SPECIFIED
BY THE PARAMETER KEYWORD DEFINE. IF THE DATA ELEMENT ALREADY
EXISTS, IT IS DELETED AND REALLOCATED. DATA ELEMENT 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 AWA:
SET OF DATA ELEMENT(S) REQUESTED

INTERNAL VARIABLES:
INTBUF - INTERFACE TABLE HEADER
SLEN - LENGTH OF SYMBOLIC STRING
IZOKEN - POSITION WITHIN THE SYMBOLIC STRING
MANARR - BUFFER FOR VALID NAMES TO BE ALLOCATED
MBUFF - BUFFER AREA FOR XGET AND XPUT USE
NRET - NUMBER OF ENTRIES IN THIS AWA REQUEST
NAME - NUMBER OF NAMES IN SYMBOLIC STRING
NEXTIN - TOKEN FOR NEXT NAME
STRING - SYMBOLIC STRING INPUT TO DEFINE

EXTERNAL ROUTINES USED:
EXEC, XGET, XPSER, XPUT, XPLT,
XREP, XPUT, XRG, XIOF, XRMG
485 1 BEGIN DEFIN
486 2 CALL RMVAR TO GET LU AND DEBUG FLAGS
487 3 CALL APGET TO RETRIEVE SYMBOLIC STRING
488 4 GET STRING LENGTH FROM INTERFACE TABLE HEADER
489 5 STARTSEARCH WHILE TOKEN-POSITION.LT. STRING-LENGTH, OR
490 6 WHILE CURRENT-TOKEN .NE. END-OF-STRING
491 7 PERFORM ZIIDM TO FIND THE NEXT NAME IN THE SYMBOLIC STRING
492 8 IF THERE WAS AN ERROR RETURN FROM ZIIDM
493 9 EXIT IF THERE WAS AN ERROR RETURN FROM ZIIDM
494 10 CALL ZIMSP TO DISPLAT SYNTAX ERROR AND POSITION IN SYMBOLIC STRING
495 11 SET PROCESSOR RETURN CODE TO ABEND
496 12 INCREMENT TO NEXT ELEMENT IN THE SYMBOLIC STRING
497 13 ENDLOOP
498 14 SET PROCESSOR RETURN FOR NORMAL EXIT
499 15 ENDSEARCH
500 16 SET OPTION SO XPRG WILL DO A QUEUE REQUEST
501 17 DO FOR NUMBER OF REQUESTS IN REQUEST BUFFER (MAXARR)
502 18 IF THIS IS LAST REQUEST, THEN
503 19 SET OPTION TO CLOSE REQUEST BUFFER
504 20 ENDIF
505 21 COMPUTE INDEX TO THIS REQUEST
506 22 CALL XPRG TO QUEUE THIS REQUEST
507 23 ENDDO
508 24 CALL XPRIT TO RETURN TO THE MANAGER
509 25 END DEFIN
BEGIN XIDNN

1. BEGIN XIDNN
2. SET NEXT NAME TO ZERO
3. DO WHILE NEXT NAME IS ZERO
4. CALL XZDFT TO FIND THE NEXT LEFT PAREN
5. CALL XZDFT TO FIND THE NEXT COMMA
6. IF THERE ARE NO SUBSCRIBTS (COMMA PRECEDES LEFT PAREN) THEN
7. SET NEXT NAME TO COMMA POSITION + 1
8. ELSE
9. CALL XZDFT TO FIND THE NEXT RIGHT PAREN
10. SET STARTING POSITION TO RIGHT PAREN POSITION + 1
11. IF STARTING POSITION > END OF STRING THEN
12. SET STARTING POSITION = END OF STRING
13. ENDIF
14. ENDIF
15. ENDDO
16. END XIDNN
594  1 CD**********
595  1 CD       ENDS IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER
596  1 CD
597  1 CD       ENDS PROCESSOR IS USED IN CONJUNCTION WITH SCAN ONLY. IT
598  1 CD       MARKS THE END OF THE SERIES OF PROCESSORS TO BE SCANNED.
599  1 CD
600  1 CD       IT HAS NO INTERFACE TABLE.
601  1 CD
602  1 CD**********
603  1 CD
604  1 CD**********
605  1 CD       INPUTS FROM THE MANAGER:
606  1 CD
607  1 CD       LU - LOGICAL UNIT OF THE USER'S TERMINAL
608  1 CD
609  1 CD       USEID - USER ID CODE
610  1 CD
611  1 CD       FLAGS - FLAGS FOR DEBUG
612  1 CD
613  1 CD       INPUTS FROM AWA:
614  1 CD
615  1 CD       &SCAN8 - SCAN CONTROL TABLE CREATED BY SCAN
616  1 CD
617  1 CD**********
618  1 CD
619  1 CD       OUTPUT: TO MANAGER:
620  1 CD
621  1 CD       XZRET - RETURN CODE TO MANAGER
622  1 CD
623  1 CD       OUTPUTS TO AWA:
624  1 CD
625  1 CD       XSCAN - NEW VALUE FOR X SCAN VARIABLE
626  1 CD
627  1 CD       YSCAN - NEW VALUE FOR Y SCAN VARIABLE
628  1 CD
629  1 CD**********
630  1 CD       RTE ROUTINES USED:
631  1 CD
632  1 CD       CLOSE, EXEC, FLOAT, KCVT, POINT,
633  1 CD       READ, RMPAR, WRIT
634  1 CD
635  1 CD       FDS ROUTINES USED:
636  1 CD
637  1 CD       XPXIT, XREX, XRMOV, XUDBG, XVPAM,
638  1 CD
639  1 CD**********
641 1 BEGIN ENDC
642 2 CALL XMPAR TO GET INPUTS FROM MANAGER
643 3 CALL XVPAM TO REQUEST AWS MANAGEMENT
644 4 ERREXIT IF $SCNTB NOT FOUND TO :ERR4:
645 5 CALL EXEC TO READ IN $SCNTB
646 6 SAVE REQUEST TO RETRIEVE SUMTAB IN REGBUF
647 7 CALL XVPAM TO REQUEST AWS MANAGEMENT
648 8 ERREXIT IF NOT FOUND TO :ERR4:
649 9 CALL EXEC TO READ IN SUMTAB
650 10 IF SUMMARY TABLE IS LARGE THAN 32 ENTRIES THEN
651 11 SET SIZE OF SUMTAB TO 32 ENTRIES
652 12 ENDF
653 13 CALL WRITF TO WRITE SUMTAB TO DATBOX
654 14 ERREXIT IF WRITF ERROR TO :ERR4
655 15 IF THERE IS 1 VARIABLE AND XCUR IS CENTROID OR
656 16 THERE ARE 2 VARIABLES AND XCUR IS CENTROID AND YCUR IS CENTROID THEN
657 17 CALL READ TO READ HEADER RECORD
658 18 ERREXIT IF READF ERROR TO :ERR4
659 19 UPDATE NUMBER OF SUMMARY TABLE ENTRIES
660 20 CALL WRITF TO WRITE UPDATED HEADER
661 21 ERREXIT IF WRITF ERROR TO :ERR4:
662 22 CALL CLOSE TO CLOSE DATBOX
663 23 SAVE REQUEST TO REMOVE $SCNTB IN REGBUF
664 24 IF THERE ARE REMAINING SCANS IN $SCNTB THEN
665 25 CALL EXEC TO WRITE REMAINING $SCNTB
666 26 SAVE REQUEST TO ALLOC AND STORE VALUES FOR NEW $SCNTB
667 27 ENDF
668 28 SET RETURN PARAMETER TO NORMAL RETURN
669 29 ELSE
670 30 PERFORM SETXY
671 31 CALL XMPAR TO GET INPUTS FROM MANAGER
672 32 ENDIF
673 33 CALL XVPAM TO REQUEST AWS MANAGEMENT
674 34 CALL XPSIT TO TERMINATE WITH RETURN PARAMETERS
675 35 EXIT ENDC
676 36 :ERR4:
677 37 CALL XMSG TO DISPLAY ERROR
678 38 CALL XPSIT TO ABEND PROCESSOR
679 39 END ENDC
1 BEGIN SETXY
2 IF XCUR IS END STEP THEN
3 IF THERE IS 1 VARIABLE THEN
4 SET X TO CENTROID
5 CALL POSNT TO POSITION FILE TO CENTROID RECORD
6 ERREXIT IF POSNT ERROR TO :ERRA:
7 ELSE
8 IF YCUR IS END STEP THEN
9 SET XCUR TO ZERO
10 SET Y TO CENTROID
11 CALL POSNT TO POSITION TO CENTROID RECORD
12 ERREXIT IF POSNT ERROR TO :ERRA:
13 ELSE
14 SET XCUR TO (XSTEP)
15 IF XSTEP IS ZERO AND Y IS CENTROID THEN
16 INCREMENT YCUR BY 1
17 CALL WRITF TO WRITE DUMMY AS CENTROID RECORD
18 ERREXIT IF WRITF ERROR TO :ERRA:
19 ENDIF
20 COMPUTE X AS ((XCENT + XINCR + FLOAT (XCUR))
21 COMPUTE Y AS ((YCENT + YINCR + FLOAT (YCUR))
22 ENDIF
23 ELSE
24 INCREMENT XCUR BY 1
25 IF THERE IS 1 VARIABLE AND X IS THE CENTROID OR
26 THERE ARE 2 VARIABLES AND X IS THE CENTROID AND Y IS THE CENTROID THEN
27 INCREMENT XCUR BY 1
28 CALL WRITF TO WRITE DUMMY AS CENTROID RECORD
29 ERREXIT IF WRITF ERROR TO :ERRA:
30 ENDIF
31 COMPUTE X AS (XCENT + XINCR + FLOAT (XCUR))
32 IF THERE ARE 2 VARIABLES THEN
33 COMPUTE Y AS (YCENT + YINCR + FLOAT (YCUR))
34 ENDIF
35 ENDIF
36 ENDIF
37 "FOR # SCAN VARIABLES"
38 CALL EXEC TO WRITE VARIABLE
39 SAVE REQUEST TO STORE VALUES FOR VARIABLE IN REBUF
40 END
41 CALL EXEC TO WRITE $SCNTB
42 SAVE REQUEST TO STORE NEW $SCNTB
43 SET RETURN PARAMETERS TO RESET SEQUENCE NUMBER
44 END SETXY
SCAN IS AN FDS PROCESSOR SCHEDULED BY THE MANAGER

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

INPUTS FROM MANAGER:

- L -LOGICAL UNIT OF THE USER'S TERMINAL
- USEID -USER ID CODE
- FLG -FLAGS FOR DEBUG
- ENTSUB-DISPLACEMENT OF THIS SCAN ENTRY IN SEGBR

INPUTS FROM INTERFACE TABLE:

- PROCAM-CARTRIDGE # FOR DATA BOX FILE
- SUMTAB-SUMMARY TABLE
- *DATBOX-NAMES OF DATA BOX ENTERED
- NVAR -NUMBER OF SCAN VARIABLE
- *TSCAN-NAMES OF X SCAN VARIABLE
- XUNIT -UNIT OF X VARIABLE
- XCENTR-CENTROID OF X VARIABLE
- XINC -INCREMENT FOR X
- XSTEPS-NUMBER OF STEPS FOR X
- YCENTR-CENTROID OF Y VARIABLE
- YINC -INCREMENT FOR Y
- YSTEPS-NUMBER OF STEPS FOR Y

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

OUTPUTS TO MANAGER

- XSCRET -RETURN CODE TO MANAGER

OUTPUTS TO WORK AREA (CAMA):

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

1. SUMTAB CHAR NAME OF SUMMARY TABLE
2. DISPL DISPLACEMENT FOR SUMMARY TABLE
3. DATBOX CHAR QUALIFIED NAME OF DATA BOX FILE
4. RESET RESET SEQUENCE NUMBER
5. CENECENTROID RECORD NUMBER
6. XSCAN NAME OF X VARIABLE
7. XDISP DISPLACEMENT FOR X
8. XCENTR-CENTROID FOR X
9. XINC -INCREMENT FOR X
BEGIN SCAN
CALL RMPAR TO RECEIVE INPUTS FROM MANAGER (LU, FLAGS, ENTRY DISPLACEMENT)
CALL XPSET TO GET PROCON AND SCAN VARIABLES
ERREXIT IF # SCAN VARIABLES < 1 OR > 2 TO :ERR3:
GET SUMMARY TABLE NAME AND DISPLACEMENT FROM INTERFACE TABLE
ERREXIT IF SINTERB 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 XPSET TO GET UNITS, CENTROID, INC, # STEPS
ERREXIT IF STEPS < O OR > 5 TO :ERR3:
ENDDO
SAVE REQUEST TO RETRIEVE VALUES FOR $SETDB AND $SCNDB
CALL YPAN TO REQUEST ASA MANAGEMENT
CALL EXEC TO READ IN $SETDB
IF $SCNDB NOT FOUND THEN
SET # SCANS TO ZERO
ELSE
SET # SCANS TO (TOTAL SIZE OF $SCNDB / SIZE OF ONE SCAN ENTRY)
CALL EXEC TO READ IN $SCNDB
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 $SETDB TO :ERR3:
GET THE SEQUENCE NUMBER OF THIS SCAN FROM $SETDB
IF THE DISPLACEMENT OF THIS SCAN IS ZERO THEN
SEARCH $SETDB FOR THE SEQUENCE NUMBER
ERREXIT IF THIS SCAN IS THE LAST ENTRY IN $SETDB 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
873 2 ERREXIT IF NO AWA SPACE TO :ERR1:
874 2 CALL A?XIT 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 BSCNTB
879 3 CALL EXEC TO WRITE OUT ORIGINAL BSCNTB
880 3 SAVE REQUESTS TO ALLOC AND STORE VALUES FOR ORIGINAL BSCNTB
881 2 ENDF
882 2 :ERR2:
883 2 CALL CLOSE TO CLOSE DATBOX
884 2 CALL PURGE TO PURGE DATBOX
885 2 SET VALUE FOR XPPUT
886 2 SAVE REQUEST TO DELETE DATBOX FROM AWA
887 2 CALL XPPUT TO REQUEST AWA MANAGEMENT
888 2 :ERR3:
889 2 CALL XIMSG TO DISPLAY ERROR
890 2 CALL XPXIT TO ABEND SCAN
891 1 END SCAN
1 BEGIN XICHR
2 SET # WORDS TO STORE IN OBJECT = # WORDS/ELEMENT FOR OBJECT
3 CASE (RESULT DATA TYPE) :
   :CHSTR:, :CHRFRE:, :CHRFRE:, :FIXERR:
2 IF # WORDS TO STORE = 1 THEN
3 FIXERR:
4 CHSTR:
5 DETERMINE # WORDS IN CHARACTER STRING
6 MOVE CHARACTER STRING TO RESULT LOCATION
7 IF # WORDS TO STORE < # WORDS/ELEMENT FOR OBJECT, THEN
8 BLANK FILL AFTER CHARACTER STRING
9 ENDIF
10 CHRFRE:
11 IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
12 POP DISPLACEMENT FROM RESULT STACK
13 ELSE
14 SET DISPLACEMENT = 0
15 ENDIF
16 POP RESULT OPERAND FROM STACK
17 IF RESULT OPERAND IS FREE, THEN
18 SET # WORDS TO BE RETRIEVED TO 0 WORDS/ELEMENT FOR OBJECT
19 ELSE CHARACTER = CHARACTER
20 SET # WORDS TO BE RETRIEVED TO # WORDS/ELEMENT FOR OBJECT
21 ENDIF
22 CALL XPRET TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
23 IF # WORDS RETRIEVED IS < # WORDS TO STORE, THEN
24 BLANK FILL REMAINING DATA
25 ENDIF
26 END CASE
27 EXIT XICHR

2 BEGIN XICHR
2 SET MESSAGE TO BE OUTPUT TO "CHARACTER DATA ELEMENT CANNOT BE SET EQUAL TO NUMERICAL DATA"
2 CALL XMSGS TO OUTPUT MESSAGE TO USER
2 CALL XLSSS TO LIST SYMBOLIC STRING
2 CALL XEXIT TO EXIT PROCESSOR
2 EXIT XICHR
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>963</td>
<td>1 CD******</td>
</tr>
<tr>
<td>964</td>
<td>1 CD0</td>
</tr>
<tr>
<td>965</td>
<td>XDBNM - DBSP INPUT PROCESSOR</td>
</tr>
<tr>
<td>966</td>
<td>1 CD0</td>
</tr>
<tr>
<td>967</td>
<td>1 CD******</td>
</tr>
<tr>
<td>968</td>
<td>1 CD0</td>
</tr>
<tr>
<td>969</td>
<td>XDBNM IS CALLED TO INTERPRET THE VARIOUS INTERFACE TABLE INPUTS</td>
</tr>
<tr>
<td>970</td>
<td>(MOSTLY SYMBOLIC STRINGS) AND BUILD DATA ARRAYS FROM THEM.</td>
</tr>
<tr>
<td>971</td>
<td>1 CD0</td>
</tr>
<tr>
<td>972</td>
<td>1 CD******</td>
</tr>
<tr>
<td>973</td>
<td>1 CD2</td>
</tr>
<tr>
<td>974</td>
<td>INPUT</td>
</tr>
<tr>
<td>975</td>
<td>1 CD2</td>
</tr>
<tr>
<td>976</td>
<td>1 CD2</td>
</tr>
<tr>
<td>977</td>
<td>ALL INPUT COMES FROM THE 26 INTERFACE TABLE ARGUMENTS</td>
</tr>
<tr>
<td>978</td>
<td>1 CD******</td>
</tr>
<tr>
<td>979</td>
<td>1 CD3</td>
</tr>
<tr>
<td>980</td>
<td>1 CD3</td>
</tr>
<tr>
<td>981</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>982</td>
<td>1 CD3</td>
</tr>
<tr>
<td>983</td>
<td>COMMON</td>
</tr>
<tr>
<td>984</td>
<td>1 CD3</td>
</tr>
<tr>
<td>985</td>
<td>MTooltip, MTooltip, MCUVAL, MTooltip, MTooltip, MC</td>
</tr>
<tr>
<td>986</td>
<td>1 CD4</td>
</tr>
<tr>
<td>987</td>
<td>NOTES</td>
</tr>
<tr>
<td>988</td>
<td>1 CD4</td>
</tr>
<tr>
<td>989</td>
<td>USES ROUTINES</td>
</tr>
<tr>
<td>990</td>
<td>1 CD4</td>
</tr>
<tr>
<td>991</td>
<td>EXEC</td>
</tr>
<tr>
<td>992</td>
<td>1 CD4</td>
</tr>
<tr>
<td>993</td>
<td>XPGET</td>
</tr>
<tr>
<td>994</td>
<td>1 CD4</td>
</tr>
<tr>
<td>995</td>
<td>XMOV</td>
</tr>
<tr>
<td>996</td>
<td>1 CD4</td>
</tr>
<tr>
<td>997</td>
<td>XLSIS</td>
</tr>
</tbody>
</table>

**NOTE:** The quality of the OCR is 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
XZDP2 - DATA BOX DISPLAY OVERLAY - PROMPTS USER FOR DISPLAY DESIRED, THEN FORMATS AND DISPLAYS DATA ACCORDINGLY
- SCHEDULED BY DBDSP

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

INPUTS TO XZDP2 FROM XZDP1

MASK TABLES CONTAINING THE CONSTRAINT MASKS FOR EACH
CONSTRAINT WHICH WAS VIOLATED AND INDICATION OF
WHETHER 02 NOT ANY CONSTRAINT WAS VIOLATED FOR EACH
ARRAY COORDINATE

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

DATA BOX DISPLAY ARRAY VALUES FOR ALL VARIABLES

OUTPUT FROM XZDP2

DISPLAY FORMAT SHOWN IN DOCUMENTATION IS SENT TO
NAME LU DEVLCE

COMBUF - BUFFER TO RECEIVE ENCODED USER RESPONSE - XROM
DATBOX - NAME OF DATA BOX TO BE DISPLAYED BY DBDSP
DATBUF - COMMON BUFFER FOR SCAN SUMMARY DESCRIPTORS
IDV1 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR FIRST VAR
IDV2 - POINTER TO DEP. DISPLAY VARIABLE NAME LIST FOR 2ND VAR.
ISAVE - LIST OF VIOLATED CONSTRAINTS BUILT BY XIMSK (MAX OF 8)
IXSCAN1 - FIRST SUBSCRIPT FOR NAME OF Z SCAN VARIABLE (INT OR O)
IXSCN2 - SECOND SUBSCRIPT FOR NAME OF Z SCAN VARIABLE (INT OR O)
IXBUF - NAME OF COMMON AREA USED FOR INTERFACE TABLE
IXSVN1 - FIRST SUBSCRIPT FOR NAME OF A SCAN VARIABLE (INT OR O)
IXSVN2 - SECOND SUBSCRIPT FOR NAME OF A SCAN VARIABLE (INT OR O)
LENBF - LENGTH OF COMBUF IN 16-BIT WORDS - XROM
LENST - LENGTH IN WDS OF CHAR STRING USED FOR USER PROMPT - XROM
MASK - ARRAY CONTAINING MASK FOR UP TO A COMBTY 4 X 121 SIZE
NAME - NAME LIST FOR VARIABLES SCANNED BY SCAN/ENDSCN
MNUM - NUMBER OF CONSTRAINTS INPUT BY USER (INTEGER)
MCOREL - List of constraint relations input by user
MCVAR - List of constraint variable names input by user (32 MAX)
1461 1 CD 4 NVARL - LIST OF DEP LISP VARIABLE PAIRS FOR PAGED OUTPUT (1-16 PR)
1462 1 CD 4 NVRCT - NUMBER OF DEP LISP VAR IN NVARL LIST (INTEGER)
1463 1 CD 4 NVRUL - LIST OF DEP LISP VARIABLE PAIR SCALE FACTORS FOR O/P
1464 1 CD 4 NSKERR - SET OF INDICATORS FOR CONSTRAINTS VIOLATED=0/NOT 0
1465 1 CD 4 NSSTEP - NUMBER OF STEPS ON EITHER SIDE OF X CENTROID (0 TO 5)
1466 1 CD 4 NSTEP - NUMBER OF STEPS ON EITHER SIDE OF Y CENTROID (0 TO 5)
1467 1 CD 4 RETC - RTN CODE FM XPRDM; 0=NRM, 1=RTN TO EXEC, 2=NULL BFR, 3=ERR
1468 1 CD 4 STRING - CHAR STRING CONTAINING USER PROMPT MESSAGE
1469 1 CD 4 TORDER - LIST OF X VAR VALUES FOR X COORDINATES (1-11 REAL)
1470 1 CD 4 XSCNM - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1471 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1472 1 CD 4 TORDER - LIST OF X VAR VALUES FOR X COORDINATES (1-1 REAL)
1473 1 CD 4 XSCNM - NAME OF X VAR SCANNED TO BE PLACED ON DISPLAY (6 CHAR)
1474 1 CD 4 XUNITS - NAME OF X VAR UNITS TO BE PLACED ON DISPLAY (6 CHAR)
1475 1 CD 4 ZTABLE - TABLE IN COMMON FOR NUMTAB VARIABLE NAMES AND UNITS
1476 1 CD 4 NUMVAL - UNITS LIST FOR VARIABLES SCANNED BY SCAN/ENDSCAN
1477 1 CD 4 SUMTAB - VALUES FOR SCAN VARIABLE(S) - 1 TO 32 VALUES/RECORD
1478 1 CD 4 CRMS - COMMUNICATION BUFFER FOR RPRR - LU, USER ID, FLAGS
1479 1 CD 4 LU - LOGICAL UNIT # FOR XPRDM CALLING SEQUENCE - USER LOCAT
1480 1 CD 4 ^DSP - DBDSP WILL OUTPUT DISPLAY TO THIS USER SUPPLIED L
1481 1 CD 4 FRMPT - TABLE IN COMMON TO COMMUNICATE WITH XPRDM
1482 1 CD 4 SELECT - SELECT =0 PROMPT; SELECT NOT 0 RUN ALL DISPLAYS TO O/P
1483 1 CD 4 WITHOUT PROMPT
1484 1 CD 4 CARTAG - CARTRIDGE USED TO LOCATE DATA BAK
1485 1 CD 4 ******** USES ROUTINES
1486 1 CD 4 XPRDM, EXEC, XIDOT
XZDOT - DISPLAY OUTPUT ROUTINE

XZDOT IS CALLED ONCE BY XZDP2 TO OUTPUT THE DISPLAY PAGE

INPUTS

COMMON - NCVARL, NDVARL, NCRELL, VALUE, K1, K2, ITABLE
ATABLE, HDVURL, HPAGE, DTHX, IXSCNM, YSCHNM
XUMIT, YUMIT, IXSCH1, IXSCH2, IYSCH1, IYSCH2

OUTPUTS

NONE

NOTES

USES ROUTINES

XRPOV
EXEC
XITSP
XRPCK
XRUPK
XR6
XRE16
IAB8

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR.
<table>
<thead>
<tr>
<th>Call: X12P (STRING, LEN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL X12P (STRING, LEN)</td>
</tr>
</tbody>
</table>

**Calling Sequence**

<table>
<thead>
<tr>
<th>Input: STRING</th>
<th>Character string with all areas of consecutive blanks filled</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEN</td>
<td>- Number of words in string</td>
</tr>
</tbody>
</table>

**Output: STRING**

| Character string with trailing blanks removed |

**Notes**

1. CALL X12P to remove trailing blanks from STRING
2. DC WHILE THERE ARE TRAILING WORDS IN STRING
3. END X12P
FORTRAN CALLING SEQUENCE:

CALL XIFCL (LU)

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

INPUTS FROM CALLING SEQUENCE:

LU - LOGICAL UNIT OF TERMINAL BEING USED.

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

BEGIN XIFCL

CALL _ENTR TO RESOLVE PARAMETER ADDRESS
START SEARCH FOR ALL XVSTB ENTRIES
EXIT IF THIS ENTRY'S LU IS MINE
END SEARCH
GET CLASS NUMBER FROM XVSTB
SAVE CLASS NUMBER IN XPCLS
END XIFCL
1755 1 BEGIN XZFRE
1756 2 * CASE (RESULT DATA TYPE) X:PRESTR:, X:FPRE:;, X:Z:PREF:;, X:PREX:;
1757 3 *                   X:PRESTR:, X:FPRE:;, X:Z:PREF:;, X:PREX:;
1758 4                      X:PRESTR:, X:FPRE:;, X:Z:PREF:;, X:PREX:;
1759 5                      X:PRESTR:, X:FPRE:;, X:Z:PREF:;, X:PREX:;
1760 3                      X:PRESTR:
1761 3 SET # WORDS TO BE STORED = LENGTH OF CHARACTER STRING IN WORDS
1762 3 MOVE CHARACTER STRING TO RESULT LOCATION
1763 3 X:PRESTR:
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 ENDF
1769 3 X:PRESTR:
1770 3 POP RESULT OPERAND FROM RESULT STACK (SYMBO TABLE INDEX)
1771 3 GET DATA TYPE FOR RESULT OPERAND FROM SYMBOL TABLE
1772 4 CALL XPQ:Q 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 XPQ:Q 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 ENDF
1779 3 X:PRESTR:
1780 3 CALL XZP:Q TO POP RESULT OPERAND, CONVERT IF NECESSARY, AND SET UP FOR STORE
1781 3 SET # WORDS TO BE STORED = RESULT DATA TYPE
1782 2 END CASE
1783 1 END XZFRE
FORTRAN CALLING PROCEDURE:

CALL XIFXD

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

INPUTS FROM ASSGCOM

LU, SYMTAB, SSTRING, RESULT, RSLTPT, DATYPS, CLSTAN

OUTPUTS TO ASSGCOM

RSLTPT, REQTST, NUMWDS, OPRNOS

FDS - XPREQ, XPXIT, XRM0V, XLSS, XMSG, XIPCS

RTE - IAND
1 BEGIN XIXFD
2  SET # WORDS TO BE STORED IN OBJECT = OBJECT DATA TYPE
3  * CASE (RESULT DATA TYPE ) :NOCHAR,:FIXRE:, :FIXRE:, :FIXF:, :FIXF:, :FIXF: 
4  3  
5  :FIXF: 
6  3  ERREXIT TO :NOCHAR: IF TYPE IN SYMBOL TABLE FOR RESULT OPERAND IS NOT FREE
7  3  IF TOP ENTRY ON RESULT STACK IS A DISPLACEMENT (TYPE = -2), THEN
8  4  POP DISPLACEMENT FROM RESULT STACK
9  3  ELSE FREE ELEMENT HAS NOT BEEN SUBSCRIBED
10  4  SET DISPLACEMENT = 0
11  ENDIF
12  3  POP RESULT OPERAND FROM RESULT STACK (SYMBOL TABLE INDEX)
13  3  CALL XPREQ TO RETRIEVE DATA FROM RESULT OPERAND AT DISPLACEMENT DETERMINED
14  3  (# WORDS RETRIEVED = OBJECT DATA TYPE)
15  3  :FIXF: 
16  3  SET TARGET TYPE TO OBJECT DATA TYPE
17  3  CALL XPCS TO POP RESULT VALUE, CONVERT IF NECESSARY, AND SET UP FOR STORE
18  3  ENDCASE
19  1  EXIT XIXFD
20  2  :NOCHAR:
21  2  SET MESSAGE TO BE OUTPUT TO "NUMERICAL DATA ELEMENT CANNOT BE SET EQUAL TO
22  2  CHARACTER DATA"
23  2  CALL XIMSG TO OUTPUT MESSAGE TO USER
24  2  CALL XILESS TO LIST SYMBOLIC STRING
25  2  CALL XPITX TO EXIT PROCESSOR
26  1  END XIXFD
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
OR FLAG DESCRIBING HOW TO COMPUTE
THEM
WORD 3 (TKSIZ) = NO. OF WORDS IN SYMBOLIC STRINGS

CD4 ENTRY TOKEN WORD 1 (SIZE) FIELD WORD 3 (TKSIZ)
CD4 1 INTEGER 3 -1=>CALL XR16 2
CD4 2 REAL 7 -2=>CALL XR14 3
CD4 3 DOUBLE 9 -3=>CALL XR13 4
CD4 4 --> 0 Q=>ERROR (INVALID) 0
CD4 5 NAME 3 -4=>USE 3 WORDS FOLLOWING TOKEN
CD4 6 ---- 0 Q=>ERROR (INVALID) 0
CD4 7 ---- 0 Q=>ERROR (INVALID) 0
CD4 8 CHAR. STR. -1=>USE VALUE -5=>USE SIZE WORDS -9=>USE SIZE+2 WORDS
CD4 10 WORD FOLLOWING BEGINNNING 2 PAST
CD4 11 TOKEN TOKEN
CD4 12 + 1 + 1
CD4 13 + 1 + 1
CD4 14 > 1 > 1
CD4 15 < 1 < 1
CD4 16 > 1 > 1
CD4 17 $ 1 $ 1
CD4 18 = 1 = 1
CD4 19 & 1 & 1
CD4 20 ? 1 ? 1
CD4 21 ( 1 ( 1
CD4 22 ) 1 ) 1
CD4 23 --- 0 Q=>ERROR (INVALID) 0
CD4 24 END SYM.STR. 0 -7=>END SYM. STR. 0
CD4 25 X 1 X 1
CD4 26 BACKSLASH 1 BACKSLASH 1
CD4 27 $ 1 $ 1
CD4 28 - 1 - 1
CD4 29 LEFT BRACKET 1 LEFT BRACKET 1
CD4 30 RIGHT BRACKET 1 RIGHT BRACKET 1
CD4 31 REPEAT 4 6=>CALL XR16 AND 2
CD4 32 , 1 APPEND "A" 1
**1924** 1 C******

**1925** 1 CDS

**1926** 1 CDS

**1927** 1 CDS

**1928** 1 CDS

**1929** 1 CDS

**1930** 1 CDS

**1931** 1 CDS

**1932** 1 CDS

**1933** 1 CDS

**1934** 1 CDS

**1935** 1 CDS

**1936** 1 CDS

**1937** 1 C******

**1938** 2 BEGIN XILSS

**1939** 2 MOVE A ' CHARACTER INTO PRINT BUFFER AND INCREMENT BUFFER POINTER

**1940** 2 DO UNTIL ALL TOKENS OF SYMBOLIC STRING HAVE BEEN PROCESSED

**1941** 2 EXIT TO ERROR 1 IF TOKEN VALUE IS < 1 OR > 32

**1942** 3 USE TOKEN VALUE TO RETRIEVE 3 CONTROL WORDS (SIZE, FIELD, TOKSIZ)

**1943** 2 EXIT TO ERROR 2 IF FIELD = 0

**1944** 3 IF SIZE < 0, THEN

**1945** 4 SET SIZE TO VALUE IN WORD FOLLOWING THIS TOKEN IN THE SYMB. STRING

**1946** 3 ENDIF

**1947** 3 IF TOKSIZ < 0, THEN

**1948** 4 SET TOKSIZ TO SIZE + 2

**1949** 3 ENDIF

**1950** 3 IF THERE IS NOT ROOM IN PRINT BUFFER FOR SIZE WORDS, THEN

**1951** 4 CALL EXEC TO WRITE PRINT BUFFER TO INDICATED DEVICE

**1952** 4 SET BUFFER POINTER TO 1ST POSITION FOR DATA

**1953** 4 IF INDICATED TOKEN PROCESSED, THEN

**1954** 5 OUTPUT LINE WITH INDICATOR

**1955** 4 ENDIF

**1956** 3 ENDIF

**1957** 3 IF FIELD > 0, THEN

**1958** 4 MOVE FIELD INTO CURRENT PRINT BUFFER POSITION

**1959** 3 ELSE

**1960** 4 CASE (ONE, TWO, THREE, FOUR, FIVE, SIX, EXIT), -FIELD

**1961** 5 SOME: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING

**1962** 5 AND PUT RESULTS INTO PRINT BUFFER

**1963** 5 TWO: CALL XRE14 WITH VALUE IN NEXT 2 WORDS OF SYMB. STRING

**1964** 5 AND PUT RESULTS INTO PRINT BUFFER

**1965** 5 THREE: CALL XRDI8 WITH VALUE IN NEXT 3 WORDS OF SYMB. STRING

**1966** 5 AND PUT RESULTS INTO PRINT BUFFER

**1967** 5 FOUR: MOVE THE NEXT 3 WORDS OF SYMB. STRING INTO PRINT BUFFER

**1968** 5 FIVE: MOVE SIZE WORDS FROM 2ND WORD PAST CURRENT TOKEN THE SYMB. STRING

**1969** 5 SIX: CALL XRI6 WITH VALUE IN NEXT WORD OF SYMB. STRING AND PUT RESULTS 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** 6 INDICATOR LINE NOT YET OUTPUT, THEN
FORTRAN CALLING PROCEDURE

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

Insert 'LENGTH' words of text from 'SOURCE' into message
'NUMBER' beginning after 'LOCATE' words, concatenate to prefix,
truncate to eighty characters and output to user terminal

INPUT-THREE CONTROL WORDS CONTAINING THE TERMINAL LU AND THE
DEBUG CONTROL FLAG. IF CONT(3) Bit 12 is ON, XUDP6 WILL
BE CALLED AFTER THE MESSAGE IS OUTPUT

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

NN-MESSAGE NUMBER OR ZERO WHICH INDICATES ONLY 'LENGTH'
WORDS OF 'SOURCE' TO BE OUTPUT

LOCATE-INTEGER NUMBER OF WORDS OF MESSAGE TO PRECEED
SOURCE-NOT USED IF 'NN' OF 'NUMBER' IS ZERO

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

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

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

NOTES
USES FDS SYSTEM MESSAGE FILE XJRMSS
USES CLOSE, EXEC, IAND, KCVT, OPEN, READF, XRM0V, XUDP6

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

CALL XIOPR(ENTRY)

XIOPR IS USED BY THE ASGAM ROUTINE XIPOS TO EVALUATE MATH OPERATIONS

INPUTS

ENTRY - OPERATOR TOKEN CURRENTLY BEING PROCESSED

FROM ASGCM - LU,SSTRNG,SYNTAB,DATYPES,RSLTPF,CLSTM,MAPWDS,RESULT

OUTPUTS TO ASGCM

RESULT,OPRND5,RE5ST,RSLTP5

INTERNAL VARIABLES

MAPOP - MAPS OPERATOR TOKENS FOR EXECUTION

EXTERNAL REFERENCES

FDS - XPREA,XPIT2,XREM0V,XI5SS,XIMSG

RTE - IAMP,OOF
1 BEGIN XOPR
3 :EXP:, :UNEQUAL:
4 :ADD:
5 PERFORM ADDITION
6 SET RESULT'S DATA TYPE AS APPROPRIATE
7 :SUBTR:
8 PERFORM SUBTRACTION
9 SET RESULT'S DATA TYPE AS APPROPRIATE
10 :MULT:
11 PERFORM MULTIPLICATION
12 SET RESULT'S DATA TYPE AS APPROPRIATE
13 :DIVIDE:
14 PERFORM DIVISION
15 SET RESULT'S DATA TYPE AS APPROPRIATE
16 :INDEX:
17 IF RESULT STACK IS NOT EMPTY, THEN
18 GET 1-DIM FOR TOP OPERAND IN RESULT STACK
19 SET # WORDS PER ELEMENT BASED ON TYPE OF TOP ENTRY OF RESULT STACK
20 ELSE IF OBJECT IS BEING EVALUATED
21 GET 1-DIM FOR FIRST ENTRY IN SYMBOL TABLE
22 SET # WORDS PER ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
23 ENDIF
24 CALCULATE RESULT (i.e., INDEX = IDIM*(SECOND OPERAND-1)+FIRST OPERAND)
25 :SUBTR:
26 IF RESULT STACK IS NOT EMPTY, THEN
27 SET # WORDS/ELEMENT BASED ON TOP ENTRY OF RESULT STACK ENTRY
28 ELSE IF OBJECT IS BEING SUBSCRIPTED
29 SET # WORDS/ELEMENT BASED ON TYPE OF FIRST ENTRY IN SYMBOL TABLE
30 ENDIF
31 CALCULATE RESULT (i.e., DISPLACEMENT = (OPERAND-1)*#WORDS PER ENTRY)
32 IF RESULT STACK IS NOT EMPTY AND TOP ENTRY IS NOT FREE OR CHARACTER, THEN
33 CALL XPAR TO RETRIEVE A LOGICAL VALUE FOR TOP OPERAND AT DISPLACEMENT
34 DECREMENT RESULT STACK POINTER TO DISCARD ARRAY NAME
35 ELSE IF OBJECT IS BEING SUBSCRIPTED
36 SET TYPE = -2
37 ENDIF
38 :EXP:
39 PERFORM EXPONENTIATION
40 SET RESULT'S DATA TYPE AS APPROPRIATE
41 :UNEQUAL:
42 CHANGE SIGN OF OPERAND FOR RESULT
43 ENDCASE
44 IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:
45 PUSH RESULT AND TYPE INTO RESULT STACK
46 EXIT XOPR
47 :OVER:
48 SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"
FORTRAN CALLING PROCEDURE:

CALL XPCS(Target, Opnum)

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

INPUTS

TARGET - DESIRED FDS FIXED DATA TYPE
OPNUM - OPERAND NUMBER FOR ENTRY CURRENTLY BEING SET UP
FROM ASGCOM - LU, SSTRNG, RESULT, RSLTPT, DATYPES

OUTPUTS TO ASGCOM
RSLTPT, OPNUMS

EXTERNAL REFERENCES
FDS - XEXIT, XHMOVE, XZLS, XMSG
RTE - COLE, FLOST, IFIX, OVF, SNGL
BEGIN XPCS

POP OPERAND AND DATA TYPE FROM TOP ENTRY OF RESULT STACK

IF TARGET TYPE AND DATA TYPE ARE NOT EQUAL, THEN

CASE (TARGET TYPE) :
  :INTS:
  :REAL:
  :DBLE:

CONVERT OPERAND TO INTEGER
CONVERT OPERAND TO SINGLE PRECISION REAL
CONVERT OPERAND TO DOUBLE PRECISION REAL

ENDCASE

ENDIF IF OVERFLOW OR UNDERFLOW IS INDICATED TO :OVER:

STORE OPERAND AND CURRENT DATA TYPE FOR USE BY FUNCTION OP ARITHMETIC OPER.

EXIT XPCS

OVER:

SET MESSAGE TO BE OUTPUT TO "OVERFLOW OR UNDERFLOW DETECTED"

CALL XMSG TO OUTPUT MESSAGE TO USER

CALL XLIST TO LIST SYMBOLIC STRING

CALL XPXIT TO EXIT PROCESSER

EXIT XPCS
FORTRAN CALLING PROCEDURE:
CALL XIPS1

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

INPUTS FROM ASGCOM
LU, TOKENS, STRING, EXPTR, SYNTAX, FNCTBL

OUTPUTS TO ASGCOM
EXPTR, RNGSTK, POLISH, SYNTAX

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

EXTERNAL REFERENCES
RTE - IAMD, MIND
FDS - XPIXIT, XILSS, XIMSG, XISYN

EXTERNAL REFERENCES
BEGIN XIPS1
DO FOR EACH TOKEN UNTIL ; OR ESS IS REACHED
PERFORM SETUP FOR SPECIAL HANDLING FOR NUMBERS, NAMES, +, *, (, AND 
PERFORM STRING FOR SYNTAX CHECKING AND POLISH STRING BUILD
END DO
PERFORM RANGE TO PROCESS RANGE DEFINITION, IF NECESSARY
END XIPS1
BEGIN SETUP
CASE Token
  WHEN CHARACTER
    CALL XSYM TO STORE CHARACTER
  WHEN SYMBOL
    CALL XSYM TO STORE SYMBOL
  WHEN NAME
    CALL XSYM TO STORE NAME
  WHEN NUMBER
    CALL XSYM TO STORE NUMBER
ENDCASE
ENDSETUP

IF Token IS A Name AND Next Token IS A Left Bracket, THEN
  STARTSEARCH UNTIL FUNCTION TABLE IS CHECKED
  EXITIF Token NAME MATCHES Function NAME
  CHANGE Token TO Function'S INDEX IN TABLE + 128
  PUSH COUNT OF BINARY/UNARY Flag AND LIMIT=0 onto Grouping Stack
ENDLOOP
ELSE Token IS An OPERAND
  IF Token IS A NAME AND Next Token IS (, THEN
    SET SUBSCRIPT Flag
  ENDIF
  CALL XSYM TO STORE SYMBOL AND SUBSCRIPT Flag
  Push Symbol Index onto Expression Stack
ENDIF

:CHAR:
  Push Negative Pointer To Character String onto Expression Stack

:+:
  IF Preceding Token Was Left Bracket Or ( Or , Or =, THEN
  INCREMENT To Next Token AND EXIT TO :ENDO:
ENDIF

:MINUS:
  IF Preceding Token Was Left Bracket Or ( Or , Or =, THEN
  CHANGE Token To UNARY MINUS
ENDIF

:*:
  IF Next Token Is *, THEN
  CHANGE Token TO EXPONENTIATION
ENDIF

:*:
  IF Preceding Token Was A VARIABLE (Data Element NAME), THEN
  PUSH Count AND LIMIT ALLOTTING DOUBLE SUBSCRIPTING onto Grouping Stack
  ELSE Expression Is Being Processed
  PUSH Count AND LIMIT ALLOWING NO SUBSCRIPTING onto Grouping Stack
ENDIF
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 XIMSG TO OUTPUT MESSAGE TO USER
2332 2 CALL XILOSS TO LIST SYMBOLIC STRING
2333 2 CALL XPIXIT 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
ERREXIT TO :SYNTAX2: 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 STACK
IF COMMA LIMIT SHOULDSUBSCRIBING IN FREAD 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 :BDELETE:
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 PARENTHESES HAVE BEEN MATCHED
IF TOP ENTRY OF OPERATOR STACK IS ( THEN
ERREXIT IF CURRENT TOKEN IS NOT , THEN
IF TOP ENTRY OF GROUPING STACK INDICATES SUBSCRIBING(LIMIT>0), THEN
PUSH SUBSCRIBING 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 :FLFTER: IF FUNCTION LIST IS NOT COMPLETE (TOP OF GROUPING STACK 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:
:SYNTAX2:
:COMERR:
:COMERR:
SET MESSAGE TO BE OUTPUT TO "INVALID SEQUENCE OF CHARACTERS"
SET MESSAGE TO BE OUTPUT TO "PARENTHESES OR BRACKETS DO NOT MATCH PROPERLY"
SET MESSAGE TO BE OUTPUT TO "INVALID COMMA OR TOO MANY COMMAS IN LIST"
2391 2  :PLSTER:
2392 2  SET MESSAGE TO BE OUTPUT TO "INCOMPLETE FUNCTION LIST"
2393 2  :BADEBL:
2394 2  SET MESSAGE TO BE OUTPUT TO "INVALID SYNTAX TO LEFT OF = "
2395 2  CALL XING TO OUTPUT MESSAGE TO USER
2396 2  CALL XLISL TO LIST SYMBOLIC STRING
2397 2  CALL XPRIT TO EXIT PROCESSOR
2398 1  END STRING
1 BEGIN RANGE
2 INITIALIZE RANGE TABLE FOR 4 RANGES STARTING AT 1, ENDING AT 1, INCREMENT = 1
3 IF LAST TOKEN INPUT WAS ; THEN
4 DO UNTIL ESS IS REACHED OR FOUR RANGES ARE PROCESSED
5 ERREXIT IF NEXT TOKEN IS NOT NAME TO :RNGSYM:
6 CALL XSYM TO INSERT NAME IN SYMOL TABLE, IF NECESSARY, FLAG ENTRY AS 
7 AN INDEX, AND RETURN SYMBOL TABLE INDEX
8 ERREXIT IF NEXT TOKENS ARE NOT "INTEGER,INTEGER" TO :RNGSYM:
9 IF SECOND INTEGER VALUE < FIRST INTEGER VALUE, THEN
10 SET INCREMENT IC -1
11 ENDF1
12 PUSH START, END, AND INCREMENT VALUES AND SYMBOL TABLE INDEX ONTO STACK
13 ENDF1
14 ERREXIT TO :RNGSYM: IF ESS HAS NOT BEEN REACHED
15 EXIT RANGE
16 :RNGSYM:
17 SET MESSAGE TO BE OUTPUT TO "RANGE SPECIFICATION DOES NOT FOLLOW: ";NAME=
18 INTEGER,INTEGER"
19 CALL XMSG TO OUTPUT MESSAGE TO USER
20 CALL XLIST TO LIST SYMBOLIC STRING
21 CALL XPRINT TO EXIT PROCESSOR
22 END RANGE
FORTRAN CALLING PROCEDURE:

CALL XIPS2

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

INPUTS FROM ASGCOM

LU, TOKENS, LASTY, SYMTAB, SSTRING, RESULT, OPNDS

OPINFO, NUMWDS, DATTPS, RSLTPT, EXPR, CLSREQ, CLSTRM, DECLAS,

RNGSTR, POLISH, SYNTAX, FMCTBL

OUTPUTS TO ASGCOM

SYMTAB, RESULT, RERST, RSLTPY, OPINFO

INTERNAL VARIABLES

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

EXTERNAL REFERENCES

FDS - XPAR, XPART, XRMV, XZCMR, XZFMN, XZFR, XZFD, XLSS, XIMSG,

XIOP, XPAR, XZRET

RTE - IABS, IAND, MAXD
1 BEGIN XIPS2
2 PERFORM TOC TO RETRIEVE TOC ENTRIES FOR ALL DATA ELEMENTS IN SYMBOL TABLE
3 PERFORM DATA1 TO RETRIEVE DATA FOR NON-SUBSCRIPED FIXED-TYPE DATA ELEMENTS
4 INITIALIZE RANGE VALUES FOR ITERATION
5 DO UNTIL ALL RANGES ARE FINISHED
6 DO UNTIL POLISH STRING IS EVALUATED (STARTING WITH SECOND ENTRY OF STRING)
7 POP ENTRY FROM POLISH STRING
8 IF ENTRY IS AN OPERAND, THEN
9 PERFORM RPUSH TO RETRIEVE DATA IF AVAILABLE AND PUSH INTO RESULT STACK
10 ELSE ENTRY IS AN OPERATOR
11 IF OPERATOR IS NOT =, THEN
12 PERFORM EXEVAL TO EVALUATE POLISH STRING
13 ELSE OPERATOR IS =
14 PERFORM REPLAC TO STORE VALUE INTO OBJECT DATA ELEMENT
15 ENDEF
16 ENDIF
17 END DO
18 END DO
19 END XIPS2
BEGIN TOC
DO UNTIL ALL ENTRIES IN SYMBOL TABLE ARE PROCESSED
IF SYMBOL NOT = 0 (SYMBOL IS DE OR INDEX), THEN
IF ENTRY IS A DATA ELEMENT (SYMBOL'S FLAG WORD NOT= 1), THEN
CALL XPREQ TO RETRIEVE TOC ENTRY
ELSE SYMBOL IS A RANGE INDEX
SET DATA TYPE TO INTEGER
ENDIF
ENDIF
END DO
CALL XPREQ WITH A CLOSE BUFFER REQUEST
END TOC
2508 1 BEGIN RPUSH
2509 2 IF OPERAND IS A CHARACTER STRING POINTER (OPERAND < 0), THEN
2510 3 PUSH ABSOLUTE VALUE OF OPERAND AND -5 DATA TYPE ONTO 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 ONTO RESULT STACK
2514 7 ENDRP
2515 8 END RPUSH
1 BEGIN EVAL
2 IF OPERATOR IS A FUNCTION, THEN
3 DETERMINE # OPERANDS FROM FUNCTION TABLE
4 ELSE
5 DETERMINE # OPERANDS FROM SYNTAX TABLE
6 ENDIF
7 DO FOR EACH OPERAND
8 IF RESULT STACK CONTAINS A SYMBOL INDEX, THEN
9 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
10 ENDIF
11 CALL XREP TO RETRIEVE DATA AND TYPE FOR FIRST ELEMENT OF ARRAY
12 STORE DATA AND TYPE IN RESULT STACK AT THIS ENTRY
13 ELSE
14 ERREXIT IF DATA TYPE IS FREE OR CHARACTER TO :MIXERR:
15 ENDIF
16 END
17 IF THE OPERATOR IS A FUNCTION, THEN
18 DO FOR EACH FUNCTION OPERAND
19 SET TARGET TYPE FROM FUNCTION TABLE
20 CALL XTPCS TO POP OPERAND, CONVERT IF NECESSARY, AND SET UP
21 END DO
22 CALL XIFMC TO PERFORM FUNCTION FOR RESULTS
23 SET DATA TYPE FOR RESULTS FROM FUNCTION TABLE
24 ELSE, THIS IS AN OPERATOR OTHER THAN A FUNCTION OR "="
25 DETERMINE TYPE REQUIREMENTS FOR THIS OPERATOR FROM SYNTAX TABLE
26 CASE (TYPE REQUIREMENTS+1) :
27 1 SAME:
28 3 IF # OPERANDS > 1, THEN
29 4 SET TARGET TYPE TO MAX OF TWO DATA TYPES
30 5 ELSE # OPERANDS = 1
31 6 SET TARGET TYPE TO OPERAND'S DATA TYPE
32 7 ENDIF
33 2 :INTGR:
34 3 SET TARGET TYPE TO INTEGER
35 4 END CASE
36 DO FOR EACH OPERAND
37 CALL XTPCS TO POP OPERAND, CONVERT IT TO TARGET TYPE, AND SET UP
38 END DO
39 CALL XOPR TO PERFORM ARITHMETIC OPERATION AND PUSH RESULT AND TYPE INTO STACK
40 ENDIF
41 0 EXIT EVAL
42 :MIXERR:
43 SET MESSAGE TO BE OUTPUT TO "FREE OR CHARACTER DATA ELEMENT FOUND IN AN
44 EXPRESSION"
45 CALL XTPSG TO OUTPUT MESSAGE TO USER
46 CALL XLSIS TO LIST SYMBOLIC STRING
47 CALL XPRIT TO EXIT PROCESSOR
48 1 END EVAL
BEGIN REPLAC
IF THE TOP ENTRY OF THE RESULT STACK IS A SYMBOL TABLE INDEX, THEN
IF THE DATA TYPE IN THE SYMBOL TABLE IS FIXED, THEN
CALL XPREQ TO RETRIEVE FIRST ELEMENT OF THE ARRAY
STORE DATA AND TYPE IN TOP RESULT STACK ENTRY
ENDIF
CASE (OBJECT DATA TYPE )
:FREE: ; :FIXED: ; :FIXED: ; :FIXED: ; :CHAR: ; :CHAR:
:CHAR: ; :CHAR: ; :CHAR:
:FREE: CALL XFRE TO RETRIEVE DATA AND SET UP FOR STORE
:FIXED: CALL FXDOBJ TO RETRIEVE DATA, CONVERT IF NECESSARY, AND SET UP FOR STORE
:CHAR: CALL CHROBJ TO RETRIEVE DATA AND SET UP FOR STORE
ENDIF
IF TOP ENTRY OF RESULT STACK IS A DISPLACEMENT, THEN
POP OBJECT'S DISPLACEMENT FROM RESULT STACK
ELSE OBJECT HAS NOT BEEN SUBSCRIPTED
SET OBJECT'S DISPLACEMENT TO 0
ENDIF
BUILD XPREQ REQUEST TO STORE # WORDS CALCULATED INTO OBJECT AT OBJECT'S DISPLACEMENT
CALL XPREQ TO STORE DATA IN OBJECT
END REPLAC
2593 1 BEGIN RNGSET
2594 2 DO FOR EACH RANGE UNTIL AN INDEX IS SUCCESSFULLY INCREMENTED OR ALL DEFINED
2595 3 # RANGES ARE PROCESSED
2596 4 IF THE CURRENT VALUE FOR RANGE INDEX IS NOT = TO END LIMIT, THEN
2597 5 INCREMENT RANGE VALUE
2598 6 ELSE
2599 7 SET RANGE INDEX VALUE TO BEGIN VALUE
2600 8 END IF
2601 2 END DO
2602 1 END RNGSET
FORTRAN CALLING PROCEDURE:

CALL XIRET(ENTRY)

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

INPUTS

ENTRY - SYMBOL TABLE INDEX CURRENTLY BEING PROCESSED

FROM ASGCOM - SYMTAB,RSLTPT

OUTPUTS TO ASGCOM

RESULT,RSLTPT

EXTERNAL REFERENCES

FDS - XRMOV

RTE - IAND
BEGIN XIRET
IF SYMBOL ENTRY IS FOR NON-SUBSCRIPTED FIXED-TYPE DATA (DATA ELEMENT OR RANG INDEX), THEN
SET RETURN VALUE = VALUE IN SYMBOL TABLE FOR THIS ENTRY
SET RETURN DATA TYPE = DATA TYPE IN SYMBOL TABLE FOR THIS ENTRY
ELSE SYMBOL IS CHARACTER OR FREE DATA OR SUBSCRIPTED FIXED DATA
SET RETURN VALUE = SYMBOL TABLE INDEX
SET RETURN DATA TYPE = -1
ENDIF
END XIRET
FORTRAN CALLING PROCEDURE

2650 1 CD0 CALL XSYM (TKNTPR, FLAG, SYMIND)
2651 1 CD0
2652 1 CD0
2653 1 CD0
2654 1 CD0**
2655 1 CD1 XSYM PROVIDES SYMBOL TABLE BUILDING AND ACCESS FOR ASSIGN. 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 TKNTPR - INDEX INTO INPUT SYMBOLIC STRING (SSTRNG) 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 ASGCOM COMMON - LU, SSTRNG, 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 ASGCOM COMMON - SYMTAB
2675 1 CD3
2676 1 CD0**
2677 1 CD4 INTERNAL
2678 1 CD4 SYMTPR - 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 XPIT, XILSS, XMSG, XSYM
2684 1 CD5
2685 1 CD0**
FORTRAN CALLING PROCEDURE

CALL XISYT (TKNPTR, SYMIND)

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

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

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

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

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

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.

ELOOP - INDEX OF LAST ALLOCATED WORD IN SYMTAB. VALUE IS EQUAL
TO LASTSY*STWIDE

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

EXTERNAL REFERENCES
XRPR, XRNW

SPECIAL REMARKS
THE REQUIRED FORMAT OF EACH SYMBOL TABLE ENTRY IS

I = TOKEN CODE I (3 WDS) I RESERVED FOR TOC ENTRY I (5 WDS) I (IN WDS) I

CD********
BEGIN XISTT
CLEAR BUFFER TO BE USED IN MOVING TOKEN
IF TOKEN IS A NAME, THEN
SET COMPARISON DISPLACEMENT IN TABLE TO 1 (NAME FIELD)
ELSE
SET COMPARISON DISPLACEMENT IN TABLE TO 8 (VALUE FIELD)
ENDIF
MOVE TOKEN INTO BUFFER
START SEARCH UNTIL ALL ALLOCATED SYMBOL TABLE ENTRIES EXAMINED
EXIT IF ENTRY MATCHES BUFFER CONTENTS AND TYPE FIELD MATCHES TOKEN CODE
END LOOP
STORE TOKEN CODE IN TYPE FIELD OF NEXT ENTRY
STORE BUFFER CONTENTS INTO APPROPRIATE FIELD OF ENTRY (NAME OR VALUE)
INCREMENT NUMBER OF ALLOCATED ENTRIES
END SEARCH
SET STRING TO ENTRY NUMBER
END XISTT
# Symbol Definition Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>2114</td>
</tr>
<tr>
<td>ADD1</td>
<td>1106</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>ASGN</td>
<td>272</td>
</tr>
<tr>
<td>ASTEN</td>
<td>2314</td>
</tr>
<tr>
<td>BACK</td>
<td>2593</td>
</tr>
<tr>
<td>BADNUM</td>
<td>2528</td>
</tr>
<tr>
<td>BADDR</td>
<td>2701</td>
</tr>
<tr>
<td>CALL</td>
<td>1098</td>
</tr>
<tr>
<td>CALL2</td>
<td>1132</td>
</tr>
<tr>
<td>CHAR</td>
<td>2304</td>
</tr>
<tr>
<td>CHAR2</td>
<td>2549</td>
</tr>
<tr>
<td>CHRFRE</td>
<td>937</td>
</tr>
<tr>
<td>CHRFRE2</td>
<td>931</td>
</tr>
<tr>
<td>COMERR</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>2206</td>
</tr>
<tr>
<td>DEFIN</td>
<td>443</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>2125</td>
</tr>
<tr>
<td>ENDCAS</td>
<td>2325</td>
</tr>
<tr>
<td>ENDO</td>
<td>2382</td>
</tr>
<tr>
<td>ENDOC</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-ON
SENDSP - INDEX OF CURRENTLY EXECUTING SEQUENCE TABLE ENTRY

INPUT FROM INTERFACE TABLE
DOTTPE - 4CH CHARACTER STRING SPECIFYING LOOP CONDITION TYPE
UNTIL - LOOP UNTIL RELATION BETWEEN OPND1 AND OPND2 IS
TRUE. TEST IS MADE AT THE END OF EACH LOOP
WHILE - LOOP WHILE RELATION BETWEEN OPND1 AND OPND2 IS
TRUE. TEST IS MADE PRIOR TO STARTING EACH LOOP
EXECUTION.

OPND1 - FIRST REAL QUANTITY TO COMPARE (VALUE NOT RETRIEVED.
CHARACTERISTICS PASSED TO ENDDO)
OPND2 - SECOND REAL QUANTITY TO COMPARE (SEE OPND1)

INPUT FROM HVH ON REQUEST TO MANAGER
INTBUF - INTERFACE TABLE (LESS LITERAL AREA)
LITERAL - NINE WORD BUFFER FOR HOLDING ORIGINAL INTERFACE TABLE
LITERAL AREA
BSTK - SEE OUTPUT DEFINITION
SERTB - 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 - $FHWN$E RESET NUMBER OF ENDDO IF RETURN = 3

INPUT/OUTPUT FROM TO HVH
BSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
(27,4) 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 PDF 5.0,
VOLUME IV, FIGURE 1.2-17)

WORDS 8-14 - INTERFACE TABLE ENTRY FOR OPRMD1 (SAME AS

IN ORIGINAL TABLE EXCEPT FOR POSSIBLE NEW

VALUES POINTING INTO LITERAL AREA)

WORDS 15-21 - INTERFACE TABLE ENTRY FOR OPRMD2 (SEE

ABOVE)

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

OPRMD1 & 2

WORD 26 - RELATION ID CODE AS follows

0 - <
1 =>
2 =
3 =
4 <= OR =<
5 <

WORD 27 - SEQUENCE NUMBER OF TOP OF LOOP

CD3

*********

CD4

INTERNAL VARIABLES

CD4

CODE  - ARRAY OF EIGHT ACCEPTABLE RELATION MNEMONICS AND

CD4

CORRESPONDING INTERNAL CODES

CD4

DOENT  - INDEX INTO GOSSTK FOR NEW 27 WORD ENTRY

CD4

PRDUFF  - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT W0RD

CD4

ENTRIES ARE OF THE FORM

CD4

RENST  - AWA MANAGEMENT REQUEST CODE

CD4

CLAS=  - CLASS AND TYPE OF DATA

CD4

NAME  - SIX CHARACTER DATA NAME

CD4

SIZE  - SIZE OF DATA

CD4

DISP  - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN

CD4

CLASSH  - CLASS I/O NUMBER THROUGH WHICH DATA IS

CD4

TRANSMITTED

CD4

TOP  - INDEX INTO SERTB OF TOP OF LOOP

CD4

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

CD4

XPGET)

100 CD4

*********

101 CD5

REFERENCE ROUTINES

102 CD5

EXEC, IAND, MOD, XRPAR, XPGET, XPREG(XPGET), XPIXIT, XRCPR, XMOV,

103 CD5

XSET, XUDGS, XUPAM, XWRS

104 CD5

105 CD5

NOTES

106 CD5

DO AND ENDDO MUST BE USED IN PAIRS

107 CD5

GOSSTK IS ALLOCATED IN THE AWA FOR INTERPROCESSOR COMMUNICATIONS

108 CD5

ZESCH CLEANS UP ANY RESIDUAL GOSSTK AFTER EXECUTION CONTROLER

109 CD5

TERMINATION

110 CD5

THE MAXIMUM NUMBER OF NESTED LOOPS IS 4

111 CD5

ALL NESTING (STRUCTURE) ERRORS ARE LEFT FOR EXECUTION TIME

112 CD5

DETECTION
124 1 BEGIN DO
125 2 CALL XGET TO INITIALIZE ACCESS TO XPREU AND TO RETRIEVE DOTYPE AND RELATN
126 3 CALL XPAM TO RETRIEVE BOSTK INTO BUFFER
127 4 IF RETRIEVAL FAILED, THEN
128 5 INITIALIZE BUFFER FOR BUILDING FIRST BOSTK ENTRY
129 6 ENDIF
130 7 IF BOSTK IS NOT FULL (NOT MAXIMUM NUMBER OF NESTS), THEN
131 8 IF XPET INTERFACE TABLE BUFFER INDICATES LITERAL DATA EXIST, THEN
132 9 CALL XPREU 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 SINTAB AND NUMBER OF PARAMETER; 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 ENDF
143 20 ENDDO
144 21 CALL XPREU TO RETRIEVE $SERTB (EXECUTING SEQUENCE TABLE)
145 22 IF $SERTB DISPLACEMENT ($SRDSP) > 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 ENDF
155 32 ENDF
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 ENDF
167 44 ENDF
168 45 EXIT IF NEST COUNTER IS ZERO
169 46 SET RESER NUMBER TO CURRENT SEQUENCE NUMBER (ENDDO JUST FOUND)
170 47 END LOOP
171 48 EXIT TO :ERROR4: FOR NO MATCHING ENDDO
172 49 END SEARCH
173 50 UNTIL:
174 51 CLEAR RESER NUMBER (CONTINUE SEQUENTIAL EXECUTION)
175 52 OTHER:
176 53 TERMINATE WITH ERROR7 FOR UNRECOGNIZED DOTYPE
177 54 ENDF
178 55 CALL XPREU TO OUTPUT NEW EXPANDED BOSTK
179 56 ELSE INVALID RELATION
1 CD******
1 CD0  FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS
1 CD0
1 CD0
1 CD0
1 CD******
1 CD1
1 CD1  THE ELSE UTILITY PROCESSOR LOCATES THE ENDF CORRESPONDING TO
1 CD1  THIS ELSE AND SETS THE SEQUENCE RESET NUMBER (VIA XPXIT) SO
1 CD1  THE PROCESSOR FOLLOWING ENDF IS EXECUTED NEXT IN THE SEQUENCE.
1 CD1
1 CD1
200 1 CD******
201 1 CD2  INPUT
202 1 CD2  SCHEDULING PARAMETERS
203 1 CD2
204 1 CD2  LU - LOGICAL UNIT NO. OF THIS FDS USER
205 1 CD2  FLAGS - DEBUG FLAGS -- BIT 11 ON WILL CAUSE DEBUG PRINT
206 1 CD2
207 1 CD******
208 1 CD3
209 1 CD3  OUTPUT
210 1 CD3
211 1 CD3  RPARMS - RETURN PARAMETERS SENT TO FDS MANAGER VIA XPXIT
212 1 CD3  (1) = 3 => PROCESSOR EXECUTION SEQUENCE TO CONTINUE AT THE SEQUENCE
213 1 CD3  NO. GIVEN IN RPARMS(2)
214 1 CD3  (2) = 8 => ABNORMAL TERMINATION OF THE PROCESSOR EXECUTION SEQUENCE
215 1 CD3  (2) = SEQUENCE NO. TO BE EXECUTED NEXT WHEN RPARMS(1) = 3
216 1 CD3
217 1 CD******
218 1 CD5
219 1 CD5  REFERENCE ROUTINES
220 1 CD5
221 1 CD5  RMPAR
222 1 CD5  XPXIT
223 1 CD5  XREX
224 1 CD5  XUGNE
225 1 CD5  XIFCL
226 1 CD5  XISCN
227 1 CD5
228 1 CD******
REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR
ENDO
  ENDDO - LOOP TERMINATION PROCESSOR

  SCHEDULED BY FDS

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

  INPUT FROM MANAGER VIA SCHEDULING PARAMETERS

  LU - LOGICAL UNIT OF USER'S TERMINAL

  FLAGS - DEBUG FLAGS FROM USER SIGN-ON

  INPUT/OUTPUT FROM/TO AWA

  BDOSTK - CONTROL INFORMATION FOR DO/ENDDO PROCESSORS. DIMENSIONED
  (Z7,N) WHERE N IS THE CURRENT NUMBER OF ACTIVE LOOPS.

  SEE DO PROCESSOR FOR CONTENTS

  THE FDS SYSTEM STATUS TABLE, XYSTB, IS ACCESSED FOR THE MANAGER/
  PROCESSOR CLASS I/O COMMUNICATIONS NUMBER

  OUTPUT TO THE MANAGER VIA RETURN PARAMETERS

  RETURN - RETURN CODE TO MANAGER
  0 - NORMAL TERMINATION. CONTINUE SEQUENTIAL
  3 - NORMAL TERMINATION. SKIP TO SPECIFIED SEQUENCE
  5 - NUMBER FOR CONTINUED EXECUTION
  8 - ABNORMAL TERMINATION. ABORT SEQUENCE EXECUTION

  RESET - SEQUENCE RESET NUMBER OF TOP OF LOOP IF RETURN = 3

  INTERNAL VARIABLES

  DOENT - INDEX INTO BDOSTK FOR LAST 27 WORD ENTRY

  XRBUFF - 64 WORD MANAGER COMMUNICATIONS BUFFER. EIGHT WORD
  ENTRIES ARE OF THE FORM

  REGST - AWA MANAGEMENT REQUEST CODE
  CLASS - CLASS AND TYPE OF DATA
  NAME - SIX CHARACTER DATA NAME
  SIZE - SIZE OF DATA
  DISP - DISPLACEMENT INTO DATA OF TRANSACTION ORIGIN
  CLASNO - CLASS I/O NUMBER THROUGH WHICH DATA IS
  TRANSMITTED

  XPCLS - CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS

  REFERENCED ROUTINES

  EXEC, IAND, IHPAR, XPREQ(XPGET), XPXIT, XRLOC, XRMOV,
298 1 BEGIN ENDDO
299 2 FIND CLASS I/O NUMBER FOR MANAGER COMMUNICATIONS IN XVSTB
300 2 CALL XVPAW TO RETRIEVE &DOSTK
301 2 IF RETRIEVAL WAS SUCCESSFUL, THEN
302 3 WRITE &INTAB TO CLASS I/O FROM LAST &DOSTK ENTRY
303 3 WRITE NON-LITERAL PORTION OF &INTAB TO CLASS I/O (LEAVE FOR XPGET)
304 3 CALL XVPAW TO RESTORE &INTAB FROM FIRST CLASS BUFFER INTO AWA
305 3 IF RESTORE SUCCESSFUL, THEN
306 4 CALL XPGET TO RETRIEVE OPRMD
307 4 CALL XEVL TO EVALUATE RELATION
308 4 IF RELATION IS TRUE, THEN
309 5 CLEAR RESET NUMBER (CONTINUE SEQUENTIAL EXECUTION)
310 5 IF THIS IS LAST &DOSTK ENTRY, THEN
311 6 DELETE &DOSTK FROM AWA
312 6 ELSE
313 6 CALL XPREQ TO REPLACE &DOSTK LESS LAST ENTRY
314 5 ENDDIF
315 4 ELSE
316 5 SET RESET NUMBER TO TOP OF LOOP
317 4 ENDDIF
318 3 ELSE
319 4 TERMINATE WITH ERRO6 FOR AWA OVERFLOW
320 3 ENDDIF
321 2 ELSE
322 3 TERMINATE WITH ERRO3 FOR ENDDO WITH OUT DO
323 2 ENDDIF
324 1 END ENDDO
FORTRAN MAIN PROGRAM - STANDARD FDS SCHEDULING PARAMETERS

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

INPUTS

SCHEDULING PARAMETERS

LU = LOGICAL UNIT NO. OF FDS USER

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

OUTPUTS

RMPARMS - RETURN PARAMETERS FOR FDS MANAGER VIA XPIXIT

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

RUTINES USED

RMPAR

XREXT

XPIXIT

XUDBG
346 1 CD********
347 1 CD0
348 1 CD0
349 1 CD0
350 1 CD********
351 1 CD1
352 1 CD1
353 1 CD1
354 1 CD1
355 1 CD1
356 1 CD1
357 1 CD1
358 1 CD1
359 1 CD1
360 1 CD1
361 1 CD1
362 1 CD1
363 1 CD1
364 1 CD1
365 1 CD1
366 1 CD1
367 1 CD1
368 1 CD1
369 1 CD1
370 1 CD1
371 1 CD1
372 1 CD1
373 1 CD1
374 1 CD1
375 1 CD1
376 1 CD1
377 1 CD1
378 1 CD1
379 1 CD1
380 1 CD1
381 1 CD1
382 1 CD1
383 1 CD1
384 1 CD1
385 1 CD1
386 1 CD1
387 1 CD1
388 1 CD1
389 1 CD1
390 1 CD********
391 1 CD3
392 1 CD3
393 1 CD3
394 1 CD3
395 1 CD3
396 1 CD3
397 1 CD3
398 1 CD3
399 1 CD3
400 1 CD3
401 1 CD********
402 1 CD5
403 1 CD5
404 1 CD5
405 1 CD5
406 1 CD5
407 1 CD5
408 1 CD5
409 1 CD5
410 1 CD5
411 1 CD5
412 1 CD5
413 1 CD********
1 BEGIN IF
2   CALL IMPAR TO GET INPUT (SHEPULING) PARAMETERS
3   CALL XPGET TO RETRIEVE VALUES FOR INTERFACE TABLE INPUTS
4   VERIFY RELATIONAL OPERATOR INPUT AS VALID AND TRANSLATE IT TO A CODE
5   IF RELATIONAL OPERATOR INVALID: ERR03:
6     CALL XIEVL TO EVALUATE THE RELATIONAL EXPRESSION
7     IF THE EXPRESSION IS FALSE, THEN
8       CALL XISCH TO LOCATE THE ELSE OR ENL' CORRESPONDING TO THIS IF
9       AND SET SEQUENCE RESET NUMBER
10      ELSE
11       CLEAR SEQUENCE RESET NUMBER FOR NORMAL CONTINUATION OF THE SEQUENCE
12     ENDIF
13     EXIT IF
14   :ERR03: CALL XIMSG 'INVALID RELATIONAL OPERATOR - MUST BE #,>=,>,=,<,<=,OR =>
15 1 END IF
FORTAN CALLING PROCEDURE

LVALUE = XIEVL (OPRD1, RELATH, OPRD2)

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

INPUT

OPRD1 - FIRST REAL VALUE TO COMPARE
OPRH2 - RELATION CODE AS FOLLOWS
   0 - NOT EQUAL
   1 - GREATER THAN
   2 - GREATER THAN OR EQUAL
   3 - EQUAL
   4 - LESS THAN OR EQUAL
   5 - LESS THAN
OPRD2 - SECOND REAL VALUE TO COMPARE

OUTPUT

XIEVL - FUNCTION VALUE OF LOGICAL TRUE OR FALSE

INTERNAL VARIABLES

TABLE - BIT MASK REPRESENTING TRUTH TABLE VALUES DERIVED AS FOLLOWS

<table>
<thead>
<tr>
<th>OPRD1 - OPRD2</th>
<th>OPRD1 - OPRD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE RELATION</td>
<td>RELATION + 0</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0 &gt;</td>
<td>T F F</td>
</tr>
<tr>
<td>1 &gt;</td>
<td>T F F</td>
</tr>
<tr>
<td>2 &gt;=</td>
<td>T F F</td>
</tr>
<tr>
<td>3 =</td>
<td>T F F</td>
</tr>
<tr>
<td>4 &lt;=</td>
<td>T F F</td>
</tr>
<tr>
<td>5 &lt;</td>
<td>T F F</td>
</tr>
</tbody>
</table>

(By Complementing Code > 2)

THEREFORE FOR OPRD1 - OPRD2 POSITIVE (BITS 0-2) 1, =, 1
FOR OPRD1 - OPRD2 ZERO (BITS 3-5) 0, 0, 1
AND FOR OPRD1 - OPRD2 NEGATIVE (BITS 6-8) 1, 0, 0
(BITS 9-15 ARE NOT USED) IF RELATN IS 0, 1, 0, 2 OR THE
COMPLEMENT IF RELATN IS 3, 4, OR 5

REFERENCED ROUTINES

XEXIT

NO CHECKS FOR VALID RELATION CODES OR OVERFLOW/UNDERFLOW ARE MADE
BEGIN XIEVL
CASE */@, *=2: DIFFERENCE OF OPRAD1 AND OPRAD2

1: SET FIELD OFFSET TO ZERO (BITS 0-3 OF TTABLE)
2: SET FIELD OFFSET TO THREE (BITS 3-5 OF TTABLE)
3: SET FIELD OFFSET TO SIX (BITS 6-8 OF TTABLE)
4: END CASE
5: IF RELATH > 2 (BOTTOM OF TRUTH TABLE), THEN
6: DECUMENT TTABLE
7: DECREMENT RELATH BY 3
8: ENDIF
9: ADD RELATH TO FIELD OFFSET (INDEXES TO CORRECT 4TH FOR RELATH AND DIFFERENCE)
10: SET FUNCTION VALUE TO INDEXED BIT OF TTABLE
11: END XIEVL
```fortran
C**       **
C** FORTRAN CALLING PROCEDURE **
C**       **
C CALL ZISCN (SRCN, PPARM)
C**       **
C ZISCN IS CALLED TO LOCATE THE CORRESPONDING ELSE/ENDIF IN THE CURRENT
C SEQUENCE TABLE. THE RETURN PARAMETERS FOR THE FBS MANAGER ARE OUTPUT
C**       **
C COMMON C**
C SRCN - FLAG INDICATING ORIGINATION OF THIS CALL
C 0 => CALLED BY IF TO FIND MATCHING ELSE OR ENDIF
C 1 => CALLED BY ELSE TO FIND MATCHING ENDIF
C COMMON C**
C DEBUG - FLAG INDICATING WHETHER ONLINE DEBUG TO BE OUTPUT
C 0 => NO DEBUG
C 1 => DEBUG
C COMMON C**
C PPARM - SCHEDULED PARAMETERS FROM THE FBS MANAGER
C (1) - LOGICAL UNIT NO. OF THE FBS USER
C (5) - SEQSP, INDEX INTO SEQUENCE TABLE (&SEQTAB) OF THE CURRENT
C COMMON C**
C**       **
C** OUTPUTS **
C**       **
C COMMON C**
C PPARM - PARAMETERS TO BE RETURNED TO FBS MANAGER VIA XPIT.
C (1) - 0 => CONTINUE NORMAL PROCESSOR EXECUTION SEQUENCE
C 3 => EXECUTE SEQUENCE NO. GIVEN BY PPARM(2) NEXT
C 8 => ABNORMALLY TERMINATE PROCESSOR EXECUTION SEQUENCE
C (2) - SEQUENCE NO. TO BE EXECUTED NEXT IF PPARM(1) = 3
C**       **
C**** Routines Used **
C****       **
C EXEC
C RMPAR
C XDPAN
C XUDG
C XRCPR
C XIMSG
C**       **
```
1 BEGIN XISCH
2 RETRIEVE $SEQTAB FROM THE AVA USING XVPAM
3 STARTSEARCH UNTIL ALL COMMANDS IN $SEQTAB
4 EXIT IF CURRENT COMMAND IS FOUND
5 SET NUMBER OF IF NESTS TO 1
6 STARTSEARCH FROM NEXT COMMAND IN $SEQTAB UNTIL ALL FOLLOWING COMMANDS
7 IF COMMAND IS FOR ENDIF PROCESSOR, THEN
8 DECREMENT NUMBER OF IF NESTS BY 1
9 ELSE
10 IF COMMAND IS FOR IF PROCESSOR, THEN
11 INCREMENT NUMBER OF IF NESTS BY 1
12 ELSE
13 IF CALLED BY IF PROCESSOR, AND
14 COMMAND IS FOR ELSE PROCESSOR, THEN
15 ERREXIT IF THIS IS THE END OF $SEQTAB $ERROR1:
16 IF NUMBER OF IF NESTS IS 1, THEN
17 DECREMENT NUMBER OF IF NESTS TO 0
18 ENDIF
19 ENDIF
20 EXIT IF NUMBER OF IF NESTS IS 0
21 SET SEQUENCE RESET NUMBER (RPARMS(2)) TO BE SEQUENCE NUMBER OF THE
22 NEXT COMMAND IN THE TABLE
23 ENDLOOP
24 ERREXIT :ERROR1:
25 ENDSEARCH
26 ENDLOCP
27 ERREXIT :ERROR4:
28 ENDSEARCH
29 1 EXIT XISCH
30
31 :ERROR1: CALL XIMSG - 'IF CANNOT "E EXECUTED WITHOUT MATCHING ENDIF''
32 :ERROR4: CALL XMSG - 'SYSTEM ERROR - NO $SEQTAB'
33 1 END XISCH
6.0 DETAILED LOGIC FLOW LISTING - PROGRAM EXECUTION

The initial pages and tailsheet of the program execution that produced this volume is presented.
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>DATE</th>
<th>TIME</th>
<th>SEQ #</th>
<th>SIZE-PRE-TEXT</th>
<th>CYCLE</th>
<th>WORD</th>
<th>PSRMODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT2</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:55:40</td>
<td>1</td>
<td>55</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1791</td>
</tr>
<tr>
<td>FSTURD</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:56:07</td>
<td>2</td>
<td>30</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1845</td>
</tr>
<tr>
<td>XUPD</td>
<td>ELT SYMB</td>
<td>15 APR 77</td>
<td>12:59:32</td>
<td>3</td>
<td>2</td>
<td>55</td>
<td>0</td>
<td>1</td>
<td>1875</td>
</tr>
<tr>
<td>FAPLIST</td>
<td>ELT SYMB</td>
<td>22 MAR 77</td>
<td>03:17:22</td>
<td>4</td>
<td>84</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1932</td>
</tr>
<tr>
<td>POLIST</td>
<td>ELT SYMB</td>
<td>08 AUG 77</td>
<td>09:10:28</td>
<td>5</td>
<td>87</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2011</td>
</tr>
<tr>
<td>POLIST</td>
<td>POLIST</td>
<td>09 AUG 77</td>
<td>09:10:34</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2012</td>
</tr>
<tr>
<td>POLIST</td>
<td>ABSOLUTE</td>
<td>08 AUG 77</td>
<td>09:10:43</td>
<td>7</td>
<td>280</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>2015</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>08 NOV 77</td>
<td>05:47:05</td>
<td>8</td>
<td>199</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>2385</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>10 NOV 77</td>
<td>06:19:22</td>
<td>9</td>
<td>394</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2494</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>11 JAN 78</td>
<td>23:22:28</td>
<td>10</td>
<td>55</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>2888</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 JAN 78</td>
<td>00:40:28</td>
<td>11</td>
<td>373</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>2923</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>13 FEB 78</td>
<td>19:17:22</td>
<td>12</td>
<td>824</td>
<td>5</td>
<td>9</td>
<td>5</td>
<td>3296</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>12:48:36</td>
<td>13</td>
<td>627</td>
<td>5</td>
<td>14</td>
<td>5</td>
<td>4120</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 FEB 78</td>
<td>23:50:36</td>
<td>14</td>
<td>320</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>4747</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>03 MAR 78</td>
<td>22:20:04</td>
<td>15</td>
<td>200</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>4857</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>14 MAR 78</td>
<td>03:16:51</td>
<td>16</td>
<td>19</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5037</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:29</td>
<td>17</td>
<td>101</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5068</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>17 APR 78</td>
<td>10:54:52</td>
<td>18</td>
<td>534</td>
<td>5</td>
<td>22</td>
<td>5</td>
<td>5094</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>27 APR 78</td>
<td>11:45:48</td>
<td>19</td>
<td>384</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5193</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>18 MAY 78</td>
<td>09:12:58</td>
<td>20</td>
<td>1548</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5729</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:31</td>
<td>21</td>
<td>403</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>6113</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:39</td>
<td>22</td>
<td>508</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>7661</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>28 AUG 78</td>
<td>12:52:42</td>
<td>23</td>
<td>151</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8064</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:46:47</td>
<td>24</td>
<td>118</td>
<td>5</td>
<td>13</td>
<td>5</td>
<td>8572</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:13</td>
<td>25</td>
<td>1201</td>
<td>5</td>
<td>21</td>
<td>5</td>
<td>8841</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:39</td>
<td>26</td>
<td>600</td>
<td>5</td>
<td>29</td>
<td>5</td>
<td>10042</td>
</tr>
<tr>
<td>X</td>
<td>ELT SYMB</td>
<td>25 SEP 78</td>
<td>09:47:57</td>
<td>27</td>
<td>326</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>10968</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**