PART 4

PROGRAMMER'S GUIDE

4.1 Introduction

Setting up the EVQs system consists of compiling five programs, storing four procedures and initializing five files. Using the system requires knowing how to use all of these to supply the keypuncher, verifier and ultimately the terminal user.

This is probably the worst manual in this thesis. This is because procedures cannot be correctly set-up until the system is finished. Using this manual requires an expertise in JCL (Job Control Language), or access to an expert.

This, then, is more an outline than a manual, and its title Programmer's Guide is warning enough. It is arranged around the self-documented JCL computer listing and eleven jobs which set-up and use the programs, procedures and files previously mentioned. The enclosed listings also contain various sample jobs illutrating handling the data.

4.2 Main Computer Runs

Set-up

EVQSBGN -- set-up initial files and procedures

EVQSCOMP -- compile the programs

Maintenance

EVQSLIST -- list status of various programs & compress PDS's

EVQSRENW -- reset the verification data sets

EVQSDOC -- enter members to documentation file

Verification

PREPNH -- prepunching sequence numbers on cards

EVQSCOR1 -- produce obvious correction listings

EVQSCOR2 -- up-date raw files and produce full verification listings

EVQSNUM -- replacing sequence numbers with access numbers

EVQSFINE -- final sort/merge

On-line System

EVQSJCL -- Alpha JCL for EVQS terminal system

Table 4.1 Main Computer Runs

4.3 Programs

Name	Language	Description	Size in K bytes
PREPNH	Assembler G/H	prepunches cards	1
OBCORR	Assembler G/H	verify listing to catch obvious errors	2
VRFY	PL/I F level 5	full verify listing	15
RENUM	Pl/I F level 5	final processing/ access number generation	15
EVQS	Assembler H	on-line analysis system	24

Table 4.2 Computer Programs

4.3.1 Compiling/Assembling The job control language required to compile the programs is basically standard. The following listing shows it. The only trick is assembling the EVQS terminal system. First, the main routine requires FSAM, the file scanning module, to run. FSAM needs the special macro SCANDCB to be present, it is now in ECOL.MACLIB, and the FSAMLIB DD card requests this. Second, the system programmer's module EVQERR must be assembled separately from the rest, because of naming duplications, and Assembler G's BATCH option handles this.

4.3.2 PREPNH The cards are punched with a five digit sequence number in columns 3 - 7; two numbers are supplied to PREPNH and it punches cards starting with the first number and incrementing by 1 up to and including the second number.

The X's in the examples are replaced by the starting and ending numbers.

There are two ways of punching cards:

1) From batch: Use job EVQS1 in JCL listing.

NOTE: A number the number of cards to be punched must be specified as the JOB card maximum cards parameter.

2) From terminal on Alpha:

EXECUTE PREPNH,TYPE=FOR,ACCT=(11Ø1,,1,45ØØ),LPARM='/XXXXX,XXXXX' [,MODE=REMOTE]

NOTE: Slash must precede the starting number.

This is necessary because Alpha invokes PREPNH using the loader.

4.3.3 Invoking Terminal System To run analysis system:

SUB EVQSJCL,ACCT=(11Ø1,123,3),MODE=CONV

4.4 Procedures

Name	Description
EVQSCOR1	runs obvious correction verification listings and enters raw data
EVQSCOR2	runs up-dates and full listing (if requested)
EVQSNUM	does final processing on corrected data
EVQSFINE	does sort/merge into master data file
EVQSJCL	runs on-line system

Table 4.3 Computer Procedures

These are standard JCL procedures, and they are listed subsequently along with the program required to add them to the procedure library, termed GENERAL.PROCLIB. The terminal system's JCL has two versions; one for batch and one for terminal. Lists for both are provided, even though the terminal version must be entered via the terminal. JCL for ZAP as described in the System Programmer's Guide, is also included.

4.4.1 Verification The verification programs are complex enough to require procedures to run them. The processing of data is basically:

1) Enter data using EVQSCOR1, and get OBCORR listing.

2) Enter corrections using EVQSCOR2, and also, if desired, get a full listing.

3) Generate access numbers using EVQSNUM, and check out numbered* cards and processing listing.

4) Sort/merge processed cards into EVQSFILE using EVQSFINE.

An example of this whole process is shown on the JCL listing. (Section 4.4.2)

---

*Access Number Interpreting the numbered listings requires knowledge of the make-up of the access number. See Section 3.3.7.1.

4.4.2 Terminal Job Control Language

```
100  //EVQSGO      EXEC  PGM=FORTGLOD,REGION=(,64K),PARM='TERM,EP=EVQS'
                                                    (change to 'EP=EVQS')*
110  //*   RUN COMMAND : SUBMIT   EVQSJCL,ACCT=(11Ø1,123,2),MODE=CONV
120  //STEPLIB    DD   DSN=CL.LLIB.SRES14,DISP=SHR
200  //           DD DSNAME=SYS1.FORTLIB,DISP=SHR
400  //SYSLIB     DD DSNAME=CL.LLIB.SRES14,DISP=SHR
410  //           DD DSNAME=SYS1.FORTLIB,DISP=SHR
500  //SYSLOUT    DD SYSOUT=9      (1)*
600  //SYSTERM    DD UNIT=TERMOUT   (remove line)*
700  //SYSLIN   DD       DSN=CL.LLIB.SRES14(EVQS),DISP=SHR
800  //*                 OUTPUT FILES
900  //SYSPRINT DD       UNIT=TERMOUT
1000  //MASPRINT DD       SYSOUT=1
1100  //FTØ3FØØ1 DD       UNIT=TERMOUT,DCB=(BLKSIZE=133,BUFNO=1,RECFM=F)
1105  //FTØ6FØ1 DD       UNIT=TERMOUT,DCB=(BLKSIZE=133,BUFNO=1,RECRM=F)
1110  //SYSUDUMP DD       SYSOUT=9      (1)*
1200  //*                 INPUT FILES
1300  //TERMINAL  DD  UNIT=TERMIN,DCB=(BUFNO=1,BLKSIZE=8Ø)
1310  //ERRIN  DD  UNIT=TERMIN,DCB=(BUFNO=1,BLKSIZE=8Ø)
1500  //*                 WORK FILES
1600  //EVQSWKØ1  DD      DSN=ROTH,EVQSFILE,DISP=SHR
1700  //EVQSWKØ2  DD  DSN=&EWKØ2,DISP=(,PASS),SPACE=(TRK,(1Ø,1Ø)),UNIT=DISK
1800  //EVQSWKØ3  DD  DSN=&EWKØ3,DISP=(,PASS),SPACE=(TRK,(1Ø,1Ø)),UNIT=DISK
1900  //EVQSWKØ4  DD  DSN=&EWKØ4,DISP=(,PASS),SPACE=(TRK,(1Ø,1Ø)),UNIT=DISK
2100  //EVQSWKØ5  DD      DSN=ROTH.WKØ5,DISP=OLD
2200  //*                 MISC FILES
2300  //DD1       DD      VOL=SER=2311122,DISP=SHR,UNIT=2311
2400  //DD2       DD      VOL=SER=2314Ø1,DISP=SHR,UNIT=2314
2500  //
#


50  //            EXEC    PGM=SUPERZAP,REGION=3ØK
100  //SYSIN     DD  UNIT=TERMIN,DCB=(BUFNO=1,BLKSIZE=8Ø)
150  //SYSPRINT  DD          UNIT=TERMOUT
200  //SYSLIB    DD          DSN=CL.LLIB.SRES14,DISP=SHR
300  //SYSIN     DD          UNIT=TERMIN
400  //
#
```

*Corrections, in parenthesis, should be made after debugging; uncorrected form is for debugging.

4.5 Data Sets

	Type
EVQSDATA -- verification and up-dating file	PDS
EVQSFILE -- master data file	sequential
EVQSDOC -- terminal system documentation file	PDS
EVQSLLIB -- library of load modules of programs--an Alpha load library	PDS
WKØ5 -- permanent work file for terminal system	sequential

Table 4.4 Computer Data Sets

These five data sets essentially contain the whole system. Three are parititoned data sets (PDS) and one of these, EVQSLLIB, is the load library for the Alpha account the system is used on. This was data set CL.LLIB.SRES14; and resided on the Alpha 2314 pak, Vol. serial no. 2314Ø2. The rest were all on the EVQS 2311 pak, Vol. serial no. 231122.

The EVQS terminal system requires four files; normally three are temporary and the fourth permanent, but with similar specifications. WKØ5 is the fourth, the temporary ones are created and deleted in the EVQS operating JCL.

JCL Reference Name	Name in PROC's	
EVQSDATA	ROTH.EVQSDATA	
EVQSFILE	ROTH.EVQSFILE	ON 231122
EVQSDOC	ROTH.DOCMN	
EVQSLLIB	CL.LLIB.SRES14	Alpha load library

Table 4.5 Present Data Set Names

4.5.1 Description of Data Files

1) EVQSDATA which was catalogued under the name {ROTH.EVQSDATA}. This contains the necessary control cards and the intermediate data during verification. It is a partitioned data set (PDS) and has members of three name types: (where X's represent numbers and letters)

RAWXXXX -- a sequentially numbered set of data cards, being verified and up-dated,

NUMDXXXX -- a verified set of data cards which have gone through final processing and contain access NUM's, and

aaaaDATA -- control cards for a program represented by aaaa=MERG,SORT,VRFY.

2) EVQSFILE -- (Named ROTH.EVQSFILE) This data set is the final data file of verified card images, sorted by access numbers. It is only allowed a single extent and is unblocked, as required by the present version of FSAM.

4.5.2 Terminal System Documentation File

DDNAME name -- DOCUMN

File name -- ROTH.DOCMN

DCB=(BLKSIZE =36ØØ,LRECL=8Ø,RECRM=FB,DSORG=PO)

SPACE =(TRK,(2Ø,2Ø,1Ø))

Member Names: PAGEXXXX -- describe code book form

ITEMXXXX -- describe item and where it occurs item number

SPXXXXXX -- other information

Numbers are left justified with no preceding zeroes. i.e.,

ITEM=32 ITEM32ØØ Use IEBUPDTE ./ ADD NAME=ITEM32 .

PART 5

PROGRAM LOGIC MANUALS

5.1 PREPNH PLM

(Entry name: PREPNH : LANG: ASMF: SIZE: 2ØØ)

PREPNH is the simplest of all the EVQS programs. It merely steps between the parameters given, punching cards numbered correspondingly as it goes.

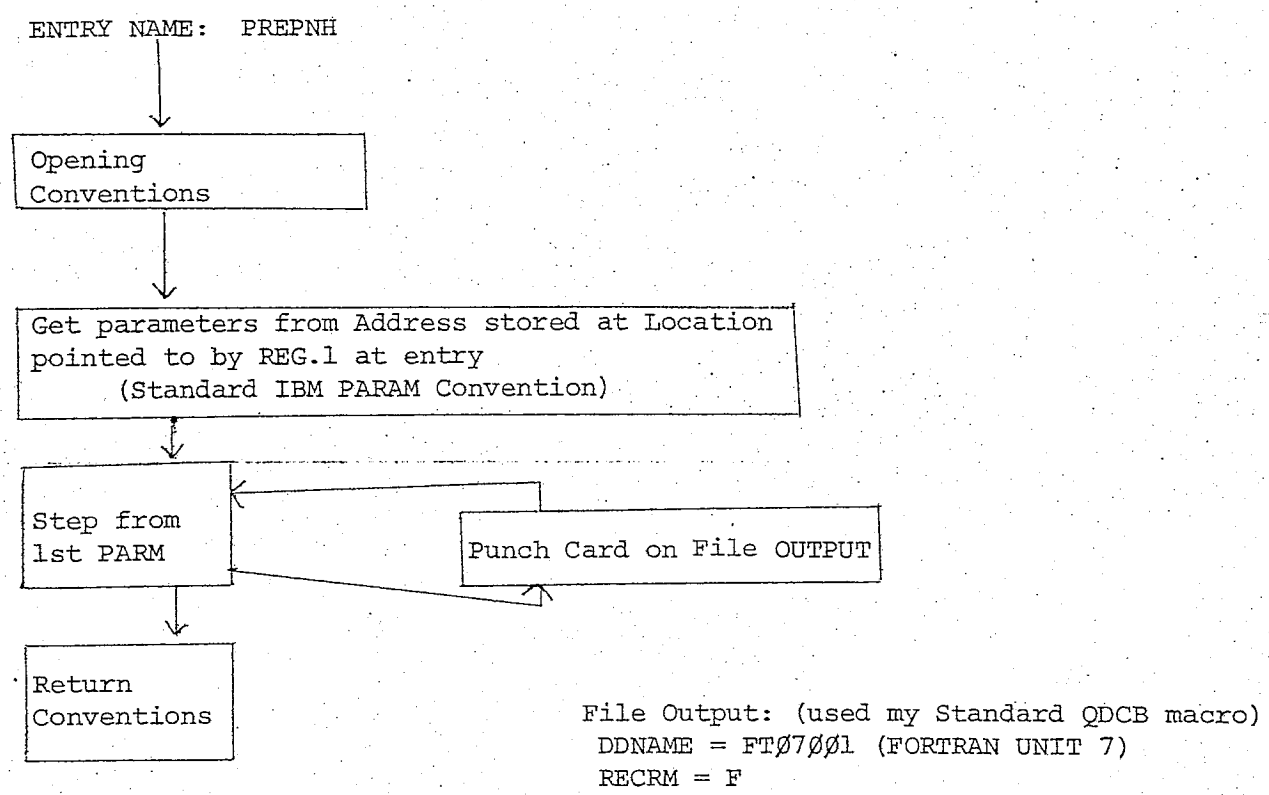
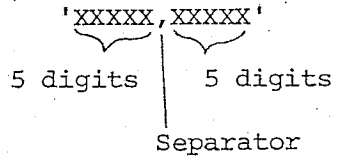


Figure 5.1 PREPNH Flow Chart

5.1.1 Parameter List

Parameter list expected is:



No checking is done on ~~v~~alidity of PARM list.

--PARM list is passed by initializing Program form JCL EXEC card.

--The program may be executed directly (PGM=PREPNH). In this case PARM='XXXXX,XXXXX' is used.

--If the loader is used, especially on ALPHA, the parameter list must be PARM='/XXXXX,XXXXX'. The slash is to cause the loader to properly pass the PARM LIST to PREPNH.

5.2 OBCORR

OBCORR has four functions:

1) Indicate the presence of an EOH on a card,

2) Indicate the absence of an EOL on a card,

3) Process all EOP's (single '@') to only '#',

4) Write a processed copy of inputed data to EVQS system file.

Files: SYSIN -(CARDS)-(INPUT)

(My QDCB MACRO used.) SYSPRINT-(PRINTER)-(OUTPUT of obvious correction listing)

EVQSOUT-(EVQS SYSTEM FILE-OUTPUT)-(If a data listing is not needed, set DD card to DUMMY)

All files: DDNAME=DCB NAME

BLKSIZE=LRECL=80,RECFM=F,MACRF=(P) or G

DSORG=PS

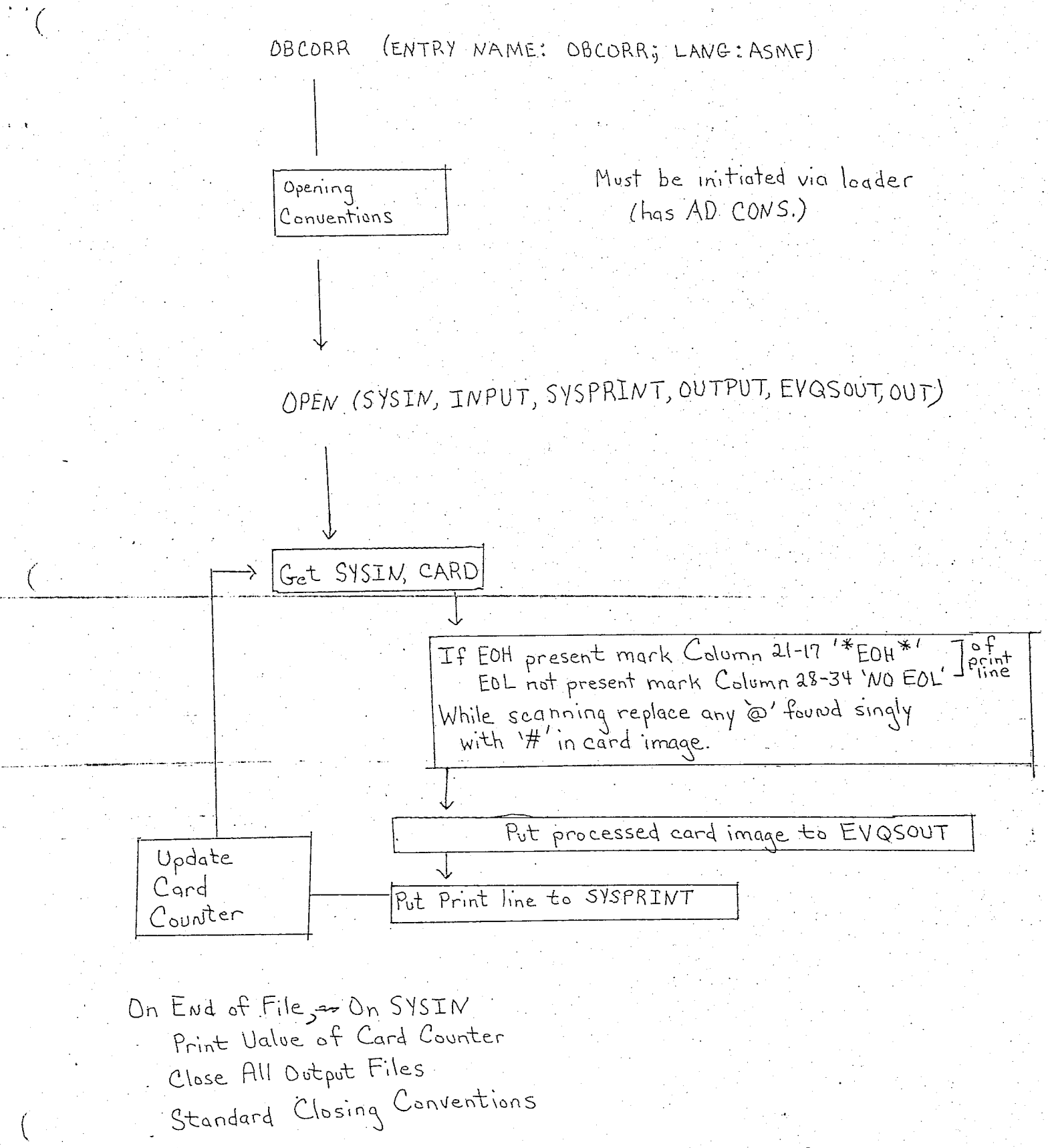


Figure 5.2 OBCORR Flow Chart

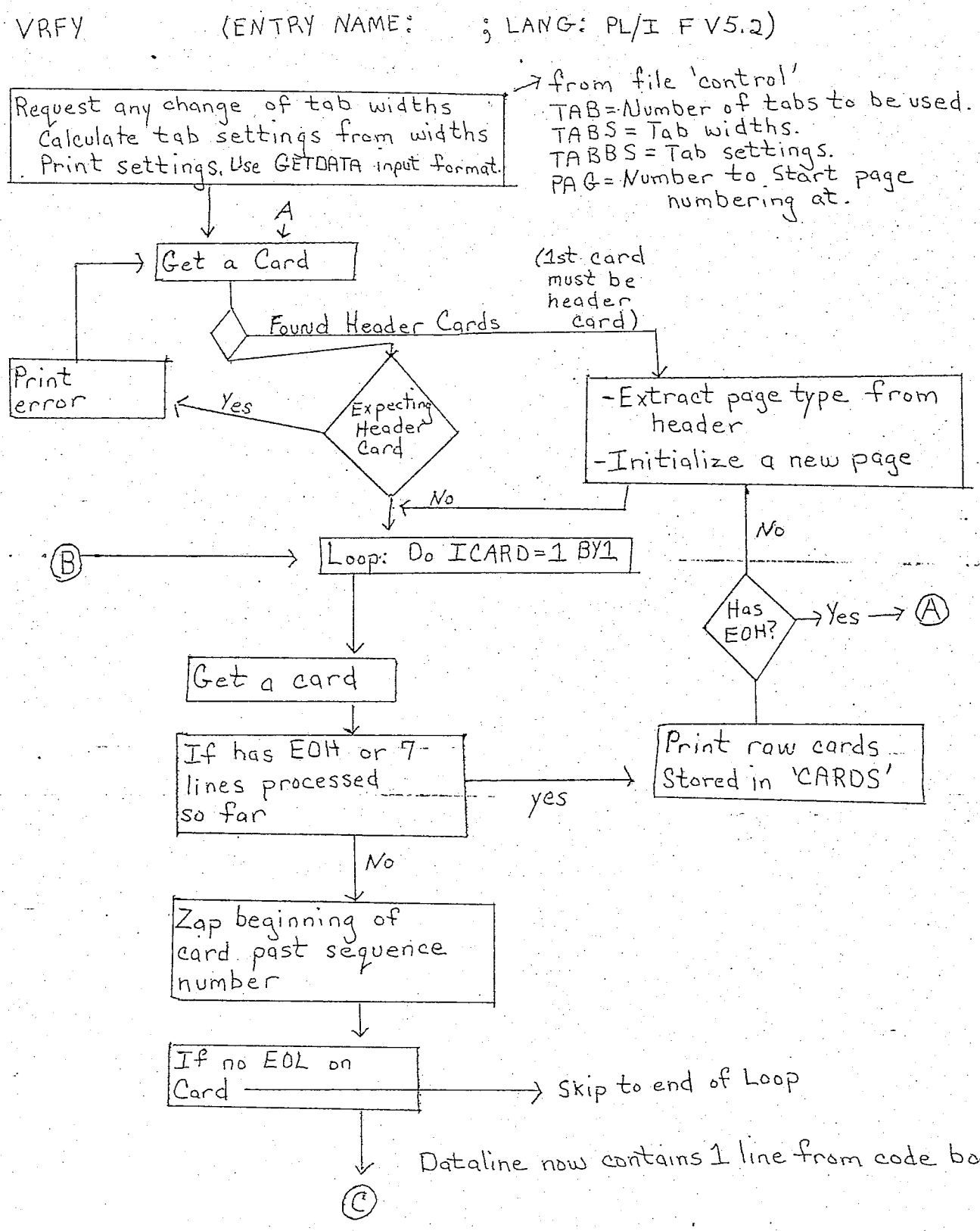


Figure 5.3 VRFY Flow Chart

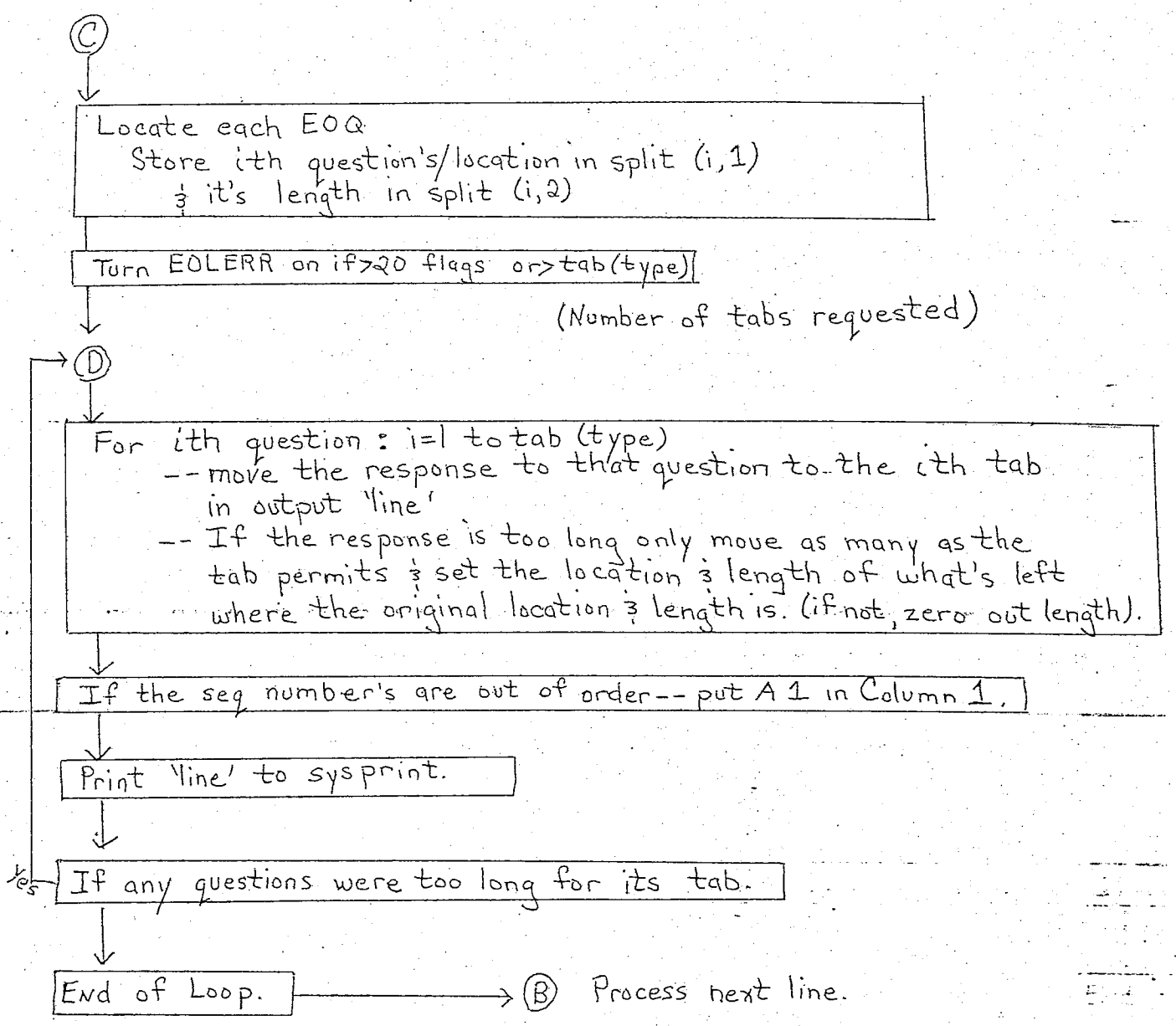


Figure 5.3 VRFY Flow Chart (cont'd)

5.3 VRFY

5.3.1 Special Actions On end of file on SYSIN:

Print raw cards stored in 'CARDS'.

Print counter of cards read and stop.

STOP.

On anything raising of the error condition:

PUT DATA useful information

If an error occurs on the put data--do a PL/I dump.

Files: (Standard PL/I files)

SYSIN-INPUT-CARDS

SYSPRINT-OUTPUT-PRINTER

PL1DUMP-ERROR OUTPUT-PRINTER (OPTIONAL)

CONTROL-TAB MODIFYING CONTROL CARDS

Funtion:

To produce a formated listing of raw EVQS data cards for verification purposes.

The format to consist of each question (indicated by EOQ) to be aligned to tabs (Default or as modified by control input). (Wherever a question is too long to fit in the tabs width given it, it will be folded over to the next line and still be within its tab settings.) If too many EOQ's exist for specified number of tabs, print remainder of line, raw, as an error.

5.4 RENUM PLM

(Entry name: RENUM ,LANG: PL/I)

Function:

To generate an access number for each EVQS data card by taking information from:

1) preceding header card,

2) first card for each line, and

3) counter indicating what card this is in relation to the first card for the line. A maximum of 1Ø cards per line.

A documentation listing is produced indicating the sequence number of all header cards.

An error in sequence number order terminates processing.

Compress lines extended to second card by length of respondent number.

Do some editing for special forms and header card uses.

Files: (All are Standard Pl/I files.)

SYSIN: INPUT CARDS RAW (BUT VERIFIED) EVQS DATA CARDS.

SYSPRINT: OUTPUT/PRINTER--DOCUMENTATION AND ERROR LISTINGS.

SYSPUNCH: OUTPUT/DATA SET--FINAL PROCESSED EVQS CARDS.

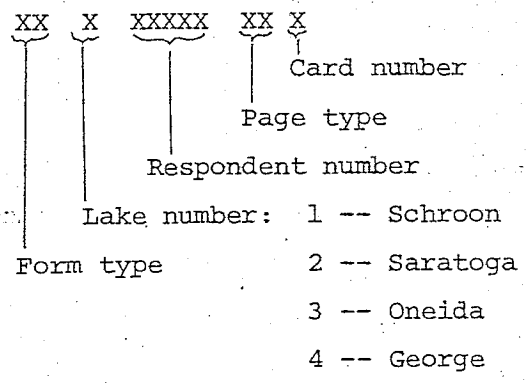


Figure 5.4 Access Number Format (11 Characters)

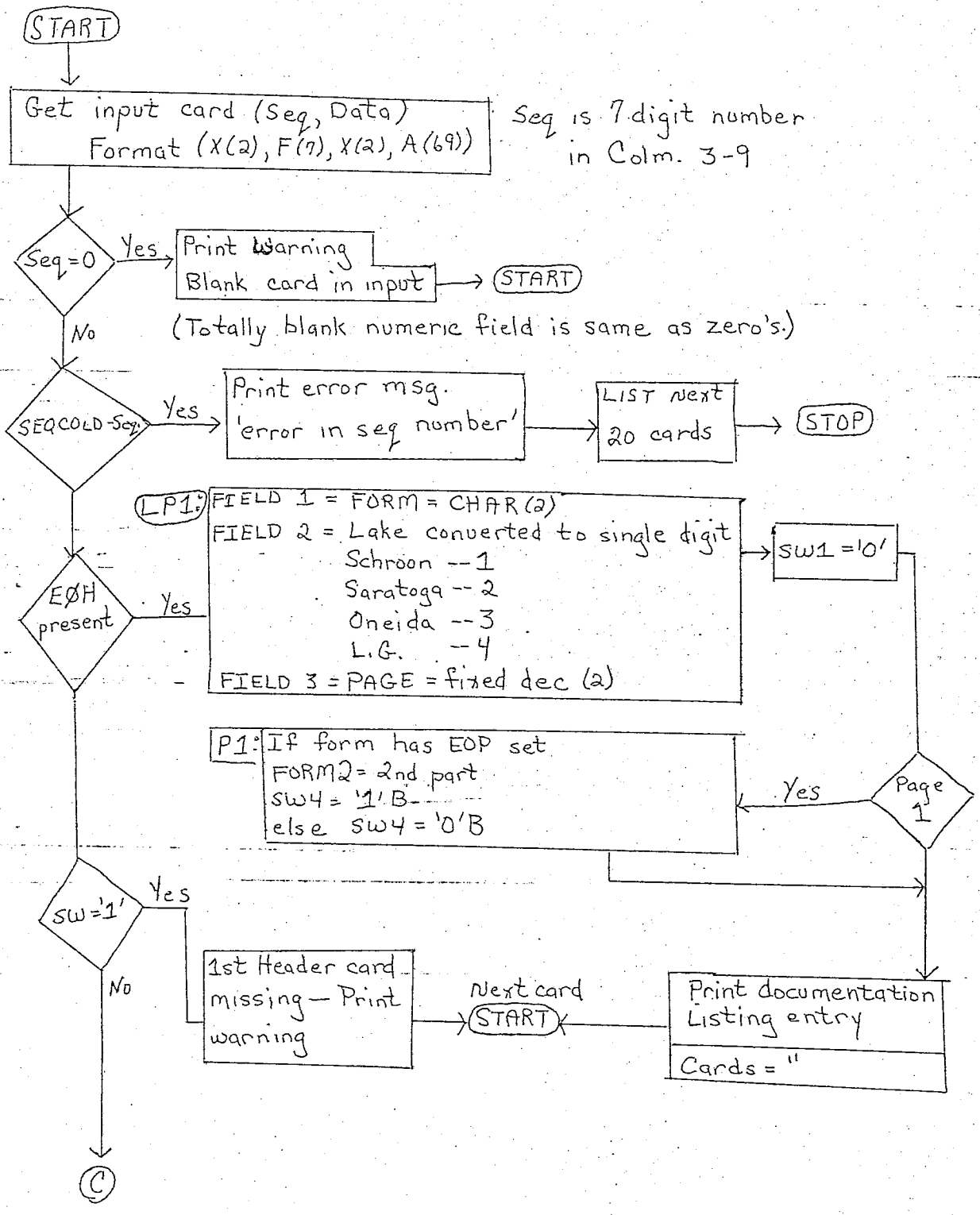


Figure 5.5. RENUM Flow Chart

NOTE: For a switch--i.e. Variable named SW# on is '1'B; off is '0'B

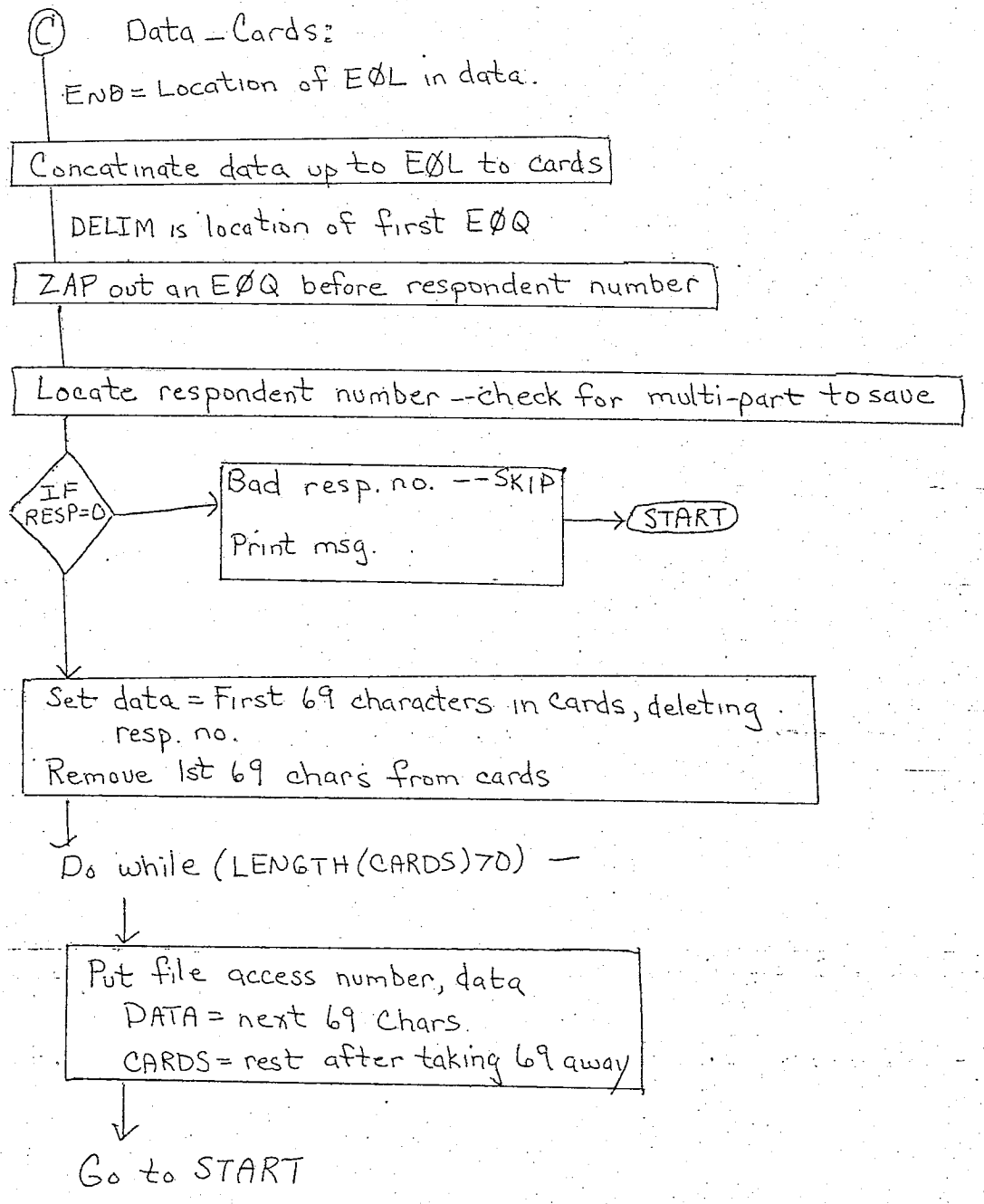


Figure 5.5 RENUM Flow Chart (cont'd)

5.5 EVQS System Programmer's Guide & Analysis Program Logic Manual

System Description--An Introduction

On-line Analysis

MACRO's and DSECT's System

Command Handler : EVQS

Routine Identifier

Syntax Parser : COMS

Grammer

Syntax Parser Set-up

Initializing

Input/Output Operations

DCB Locations : EVQSFILES

Assign Command Logic : EVQASIGN

File Scanning

Scan Command Logic : EVQSCAN

The Analysis Routines

Analysis Outputs

Flags

Lists

Conditional input handler : EVQIF

Tally Input Handler : EVQTALLY

Evaluative Routines

Overhead Manager : EVQEOQR

List Action Routine : QLLIST

Binary Tally Action Routine : QLTALLY

Binary List Action Routine : QLBINARY

Conditional Evaluation Routine : QLCOND

Translation Routines

Character String Translator : EVQCST

Tally Output Routine : EVQTYOUT

Documentation Function

Query Function Logic : EVQUERY

System Programmer's Functions

Begin Command

End/Error Recovery Commands

Special Logic Notes

Begin Command Logic : EVQBGN

End Command Logic : EVQEND

Adding Functions

Adding Options

Conventions, MACRO's & DSECT's

MACRO & DSECT Listings

5.5.1 Introduction to System Description The analysis system was written in IBM Assembler, Level F and has separate control sections (CSECT's) for each major function. The main control routine also contains a two level systax parser, which :

a) recognizes and activates the requested functions,

b) recognizes parameters if re-entered from the function routine.

The function requested is identified from the command input by the main routine and the appropriate control section is entered. The function CSECT does initialization and then usually calls the parameter parser, which is in the main CSECT. The parser passes the input parameters to the specific routine required, within the function CSECT, based

on the keyletter and form of the parameter. After the parameters are processed, an action routine is called, within the function CSECT; then control is returned to the main CSECT to process the next command. (Figure 5.6 shows the control sections involved.)

In addition to the main CSECT, and function CSECT's, there are a number of control sections invisible to the user that perform necessary utility processes and reduce the complexity of the system considerably. Input/output file management is handled by one of these utility CSECT's, as is system initialization. I/O processing is centralized by placing all the general-purpose file's data control blocks (DCB's) in a special CSECT and having a table of file assignments containing the DCB addresses. This arrangement does not allow use of standard 360/Assembler I/O Macro's, and so special ones have been provided where they are used often (read,write); other functions normally filled by MACRO's are written in-line (open,close). A number of other special macro's are also used; to convert a character representation of a number to internal form, CSECT beginning conventions, and save area conventions.

Dummy control sections (DSECT's) and a common storage area (Getmain'd at initialization time) also reduce the complexity and allow the system to be easily made re-enterant if required in the future. An error recovery routine is included as a system programmer's command to allow on-line debugging; it allows storage or registers to be dumped on request, and if

so requested, will switch to the dumping routine after a program interrupt, rather then abending. The system is invoked using the loader to resolve address constants. Fortran modules used by the error recovery routine are also included at this time from the Fortran Load library.

When debugging, 'MSGCLASS=9' added to the submit command will cause full JCL and system messages to be listed, otherwise they are lost.

All CSECT's are followed by LTORG command. This causes all literals to be assembled following their respective CSECT's, and the linkage editor may then be used to replace corrected CSECT's without disturbing any others.

5.5.2 The On-Line Analysis System This computer program consists of the two main divisions:

1) The command handler: This part breaks down the commands input from the terminal and passes the parameters given, to the appropriate analysis routine.

2) The analysis routines: These do the actual work; included are routines to:

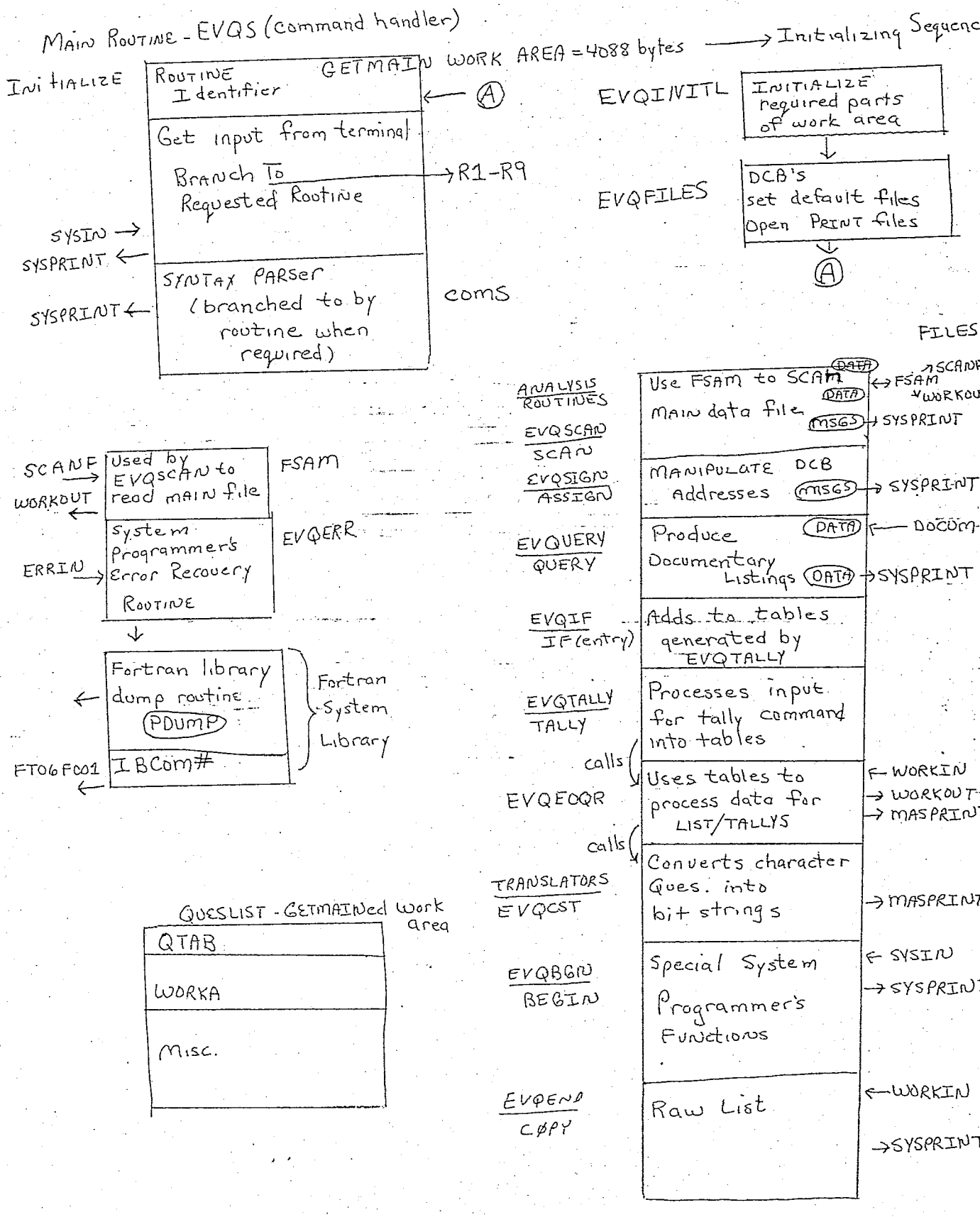
SCAN--create a work file subset of the master data file for analysis,

LIST--generate a raw or formated list of a work file,

TALLY--generate various full or conditional tallies of data,

and To manipulate the various work files.

Figure 5.6 EVQS System Control Sections



ROUTINES

CSECT	LEVEL	CALLED	FUNCTION	SAVE	BASE
EVQS--Main CSECT	Ø	OS	Main Routine Command Processer Routine Identifier	YES	BASE (12)
COLON* thru FINE* COMS*	2	Function Syntax Parser Routine	[1]	NO[2]	
EVQINITAL	1	EVQS	System Initialization		
EVQEOQR	2	EVQTALLY	Analysis Overhead Manager	SAVEQ	BASE
EVQTYOUT	2	EVQTALLY	Tally Output		
EVQUERY	1	EVQS	QUERY	YES	BR1
EVQFILES	1	EVQS	File Initialization	NO	BR1
EVQASIGN	1	EVQS	ASSIGN	YES	BR1
EVQSCAN	1	EVQS	SCAN	YES	BR1
EVQBGN	1	EVQS	BEGIN	YES/SAVEF	BR1
EVQEND	1	EVQS	END	YES	BR1
EVQERR	1/2	OS EVQBGN	Error Recovery Routine	YES	BASE
EVQIF	1	EVQS	IF	YES/SAVE	BR1
EVQTALLY[3]	2	EVQS	TALLY	YES/SAVE	BR1

*These are internal program labels .
[1]Address stored in COMSROUT of DSECT QUESLIST.
[2]Do not touch reg9 (CHAR). This is the location of the last character scanned.
[3]This is an entry in EVQIF; contains body of EVQIF.

Table 5.1 Terminal Routines

5.5.3 MACRO's and DSECT's These assembler functions were used to ease the lay-out of the program: a) Macro instructions, and b) Dummy control sections (DSECT's). In addition, a number of conventions for registers and save areas were developed. Full listings are computer lists at the end of this section; following are summaries of these: DSECT's.

Summary of Dummy Control Sections (DSECT's)

DSECT	Base Register	Description
ADRLIST	COMBASE (6)	Command Access List. These are tables for Systax Parser. The base register is filled by the preface part of the function routine requested.
QUESLIST	QUESBASE	Work area of storage--GETMAIN'ed during initialization used for data employed in many control sections.
DCBLIST	DCBASE	Default DCB addresses--used by file assignment routine to set DCB addresses in QUESLIST.
QTLIST	REG2	Form of question table; overlayed on the variable QTAB in QUESLIST--REG2 set to determine question viewed. Current value stored in REG2STRE.

Table 5.2 Summary of DSECT's

Register	Symbol	Purpose
2 - 5	REG2,REG3,REG4,REG5	Work registers.
9	CHAR	Address of input character being processed.
10	CNT	General purpose counter.
12	BASE	Main CSECT base.
11	BR1	Function CSECT base.

Table 5.3 Summary of Registers.

5.5.4 Command Handler The main control section, in addition to various overhead duties, recognizes the input commands. Two functions are performed: a) identify the routine requested, and b) locate the parameters. The first is done by the routine identifier; while the latter is done by the Syntax analyzer. The standard command format is shown in Table 5.5. (NOTE: Angular brackets, < > , are used to denote character strings in the command language. Unless otherwise noted, blanks are always ignored, and only first letters of keywords are checked.)

<keyword> :	Routine Keyword
<Keyword>	Parameter-Form 1
<Keyword> = <STRING>	Form 2
<KEY> = (<STR> , <STR>)	Form 3
<KEY> = (<STR> - <STR> , <STR> - <STR>)	Form 4

Table 5.4 EVQS Standard Command Format

NOTES: a) No blanks my appear between the first letter of the routine keyword and the following colon; after an equal sign, a left parenthesis, a dash or a comma within parenthesis.

b) Different forms, when used together, are separated by commas and blanks may surround that use of a comma.

c) Blanks placed between a delimiter and a string are ignored in counting string length, but because the location passed to the function routine is that of the preceeding delimiters, and not all routines check for intervening blanks, such blanks are best left out.

5.5.4.1 Routine Identifier This section of the code, identifies the analysis routine that is being requested and follows the necessary path to link to the routine.

The basic logic is to locate the initial character of the routine keyword by backspacing to a blank preceding a colon. Then looking up the letter after the blank in a table of 26 entries containing offsets in a list of routine names. The caharacter is located by simple calculation using the EBCDIC value of the initial character. EXAMPLE:

CHAR	Hexadecimal
A is	X'C1'
T is	X'E3'

T-A=X'E3'-X'C1'=X'22'=34

The 34th entry in the table is offset 24 which is the address of the routine to do the function 'TALLY'. (Because of the EBCDIC representation of letters are not contiguous, fillers exist between I & J: R & S in the table. Also, any character other than a letter located in this manner will abend the system by loading an incorrect branch address--no check is made.

5.5.4.2 Keyword/Parameter Syntax Parser

5.5.4.2.1 Logic When the analysis routine has been identified and linked to, it is passed a string of characters containing its parameters. To allow use of a relatively free form input; a syntax was developed and a finite state syntax recognizer was implemented. This allows each function control section to specify to the syntax parser the particular form of commands which the CSECT requires and have returned to it the

parameters in a form easily recognizable to the CSECT. The syntax parser does the "dirty work" of recognizing the input text. Figure 5.7 show possible forms and related parameters.

The translation grammar used here is described in Tables 5.5 and 5.6 using standard lexical notation. It was designed on the basis of theory developed by Lewis (1). The states are indicated by brackets along the left margin and are distinguished by a hexidecimal flag 'LK' (hence the states Ø,4,8,C(12)). Productions are recognized by subsequent input characters. The pointer to the present letter being processed is advanced only in one spot. (Labeled 'NEXT'); blanks are also automatically skipped here. The state then determines the area of code to branch to; where compare instructions (with intermediate character operands) appear for each possible delimiter. The actions then taken, along the corresponding labels in the code, are in Table 5.7.

5.5.4.2.2 Operation - The function CSECT supplies a set of keyletters; a flag for each, signifying the forms that are to be accepted; and a list of addresses, one for each active form of each keyletter-these being the location of a routine to handle the parameters entered as each form is inputed.

The parameters found are always passed in the locations 'PARM' and 'PARML' (in the QUESLIST DSECT). These are both locations of 8 bytes length; normally only the first four are used. Only a return 4 uses the full eight in each: the

last four bytes in each are the first of the pair of parameters, and vice-versa. 'PARM' is the address of the delimiter preceding the parameter in the input card; 'PARML' is the length, in hex, of the parameter, as counted between delimiters, skipping blanks. Figure 5.7 shows where PARM and PARML point in a number of examples.

(NOTE 1: No special indication is passed to distinguish the last PARM from the preceding ones, in a form 3 or 4 parameter list.)

(NOTE 2: The routines that use these values often to move the parameter to some other location--this is accomplished using the Execute operation and a dummy MVC operation, with its length field (bits 8 - 15 in SS instruction) zero'ed. The value in the register for the Execute operation must be the contents of 'PARML', minus 1. See description of execute command in IBM principle of operations.)

(NOTE 3: Many Execute operations were used and any labeled instructions in the code that seem to be inaccessable are probably used in this manner.)

Forms recognized by Syntax Parser & Parameters Returned. (Arrows point to actual locations passed, that of the preceding delimiter. The corresponding length is the difference in delimiter locations, minus blanks.)

Form	Image
1	KEY , No Parameter--just go to Return 1 ↑ CHAR
2	KEY = STR , PARML PARM CHAR Call Return 2
3	KEY = (STR1 , STR2) PARML PARML CHAR PARM CHAR PARM Passed 1st time Passed 2nd time Return 3 called
4	KEY = (STR1 - STR2 , STR3 - STR4) PARML+4 PARML PARML+4 PARML PARM+4 PARM CHAR PARM+4 PARM CHAR
3,4	KEY = (STR1 , STR2 - STR3 , STR4) PARML PARML+4 PARML PARML CHAR PARM CHAR PARM+4 PARM CHAR PARM 1st time 2nd time 3rd time Call return3 Call Return 4 Call Return 3

Figure 5.7 Syntax Parser Forms

5.5.4.2.3 Grammer Syntax (with Action Symbols) (Names before colon, in action symbols, are corresponding labels in the code.)

Routine Identifier

⟨Input Card⟩ → ⟨Command⟩ X'FF' {RD: get new input}
→ X'FF' {COMERR: no colons in input sequence}

⟨COMMAND⟩ → ⟨Routine Keyword⟩ : {COLON: call function}⟨B⟩⟨SUB'S⟩⟨Command⟩
⟨NULL⟩

Syntax Parser

State		
Ø	⟨SUB'S⟩ →	⟨KEY {Store SUB}{STATE 4}⟨SUB⟩ ⟨SUB'S⟩ X'FF' {BACK: end of input}
4	⟨SUB⟩	= {S2: Start of PARM, State 9}⟨PARMS⟩ ,{S2A: End of Subcommand, CHK RTN 1, state Ø}⟨SUB'S⟩ X'FF' {S2B: End of Subcommand}{BACK: end of input} ⟨J⟩⟨SUB⟩
8	PARMS	X'FF' {S3AA: End of Subcommand, CHK 2, State Ø} ⟨SUB'S⟩ , {S3AA}⟨SUB'S⟩ ({S3A: Start of PARM, State 12}⟨STRS⟩ ⟨J⟩{S3B: COUNT}⟨PARMS⟩
12	STRS	S'FF' {S4D: End of PARM list, State Ø} {S4AB} , {S4AB: End of Subcommand}⟨STRS⟩ = {S4B: Start of PARM 2, CHK RTN 4} ⟨STRS⟩) {S4D: End of PARM list, State Ø} {S4AB} ⟨SUB'S⟩ ⟨J⟩{S4E: COUNT} ⟨STRS⟩

Table 5.5 Grammer Syntax

Symbols -- Terminal & Non-Terminals

Terminals

⟨J⟩ = Junk; any length string of any characters, except , = Ø : ( )

⟨B⟩ = Blanks; A string of - Ø blanks

, = Comma

= = Equal sign

(= Right parenthesis

) = Left parenthesis

: = Colon

X'FF' = Hexidecimal number FF --all bits on--decimal number 255

KEY = The first letter of a valid keyword parameter for command being processed.

NULL = NULL string.

Non-terminals

⟨INPUT STRING⟩ = Input card prescanned to delimit command by X'FF''s.

⟨COMMAND⟩ = One command followed by X'FF'

⟨SUB'S⟩ = Subcommands--A string of subcommands Ø

⟨SUB⟩ = One subcommand

⟨STR⟩ = A string of characters; either after KEY or a parameter (Form 1 or 2).

⟨PARMS⟩ = Parameters; single (Form 2), List (Form 3), or pairs (Form 4).

⟨STRS⟩ = Multiples parameters; list (Form 3) or pairs (Form 4).

⟨END⟩ = End delimiters of a subcommand (Form 1 or 2).

⟨ENDA⟩ = End delimiters of a parameter list (Form 3 or 4).

Table 5.6 Grammer Symbols

Symbol	Label	Action
		(Range is up to but not including second label. Single one is closest preceding one.)
{GET NEW INPUT}	--RD--	Get a new input card--from terminal--uses BSAM--in-line (No MACRO's).
{CALL FUNCTION}	--COLON--	a) Locates colon and next one (or X'FF').
		b) Marks X'FF' before next command.
	EE1	c) Uses present commands first letter in table look-up as described.
		d) Branches to function CSECT requested or gives error message.
		e) If syntax parser required; function CSECT does set.
IN COMS		
{STORE SUB}	--LOOP--	a) Have found key letter (LOOP).
		b) See if in 'LETS' last.
	OTHRS	c) If so, store A(A(RETURN1)) in BR. (TITLF)
		d) No good, print message and terminate command. (ERROR)
{START OF PARM}	--S2--	a) Store CHAR register in PARM
		b) Zero CNT register.
		c) Do state transition.
{END OF SUBCOMMAND}	--S2A	a) Store CNT register in PARML.
	S2B	b) Check to see if Return is active.
	S3AA	c) Do state transition.
	S4AB	d) Branch to function routine indicated by adding return offset to address at location in register BR. (In state 12 (S4AB), a more complex sequence must be done: because the offset is variable, if PARML has X'FO', for RETURN 4, use offset X'C'(12)--otherwise use Offset 8 then store CNT in PARML.)
{END OF INPUT}	--BACK--	a) Branch to GO routine of function.
		b) Return to routine identifier.
{END OF PARM LIST}	--S4D--	a) Make sure RETURN 4 has two PARMS--that is, PARML is X'FO'.
		b) Branch to S4AB (End of Subcommands). NOTE: no indication is passed that a PARM list is over.
{START OF PARM}	--S4B--	a) Move PARM & PARML right by 4 bytes each.
		b) Zero CNT register.
		c) Set PARML to X'FF' to indicate Return 4.

Symbol	Label	Action
{COUNT}	--S3B-- S4E	a) Add one to CNT register. b) For S4E, 'AND' PARML with X'FO' to indicate parameter has occurred.

Table 5.7 Action Symbols (cont'd)

5.5.4.2.4 Syntax Parset Set-Up The CSECT entry to the routine performs initialization necessary to handle each form. The preface establishes addressability and presets anything necessary, in addition to setting-up for the Syntax parser.

1) Load COMBASE (Register 6) with the address of the following address constants: (1 full word each)

NUML--A(Number of key initials).

LETS--A(Letter list, NUML bytes long).

RETNS--A(Flags of active returns).

OFSETS--A(Virtual offsets in RETNS list).

ADRS--A(List of returns).

GOROUT--A(Processing routine).

COMBASE ⟶ [Points to] ⟶ NUML

NUML--A fullword binary number that is the number of letters (keywords) for the command (All the following lists are of this length.

LETS--A list of single characters, containing the key letters to be recognized, where length is number above, in btyes.

RETNS--A list (same length as 'LETS' list) where each corresponding byte, to 'LETS' list, indicates if a parrticular form is to be acceptable.

BITS				
Ø - 3	Off Always			(Binary Value)
4	Form 1			(8)
5	Form 2	Bit is on		(4)
6	Form 3	if associated		(2)
7	Form 4	return is active.		(1)

OFSETS--A list (length as above) each byte containing the offset to be added to address in 'ADRS' location to get the location of the address to be branched to for RETURN 1 for the corresponding letter in 'LETS' list. (Even if RETURN 1 is not active, the offset is calculated for RETURN 1, on the assumption that the four returns are in order; therefore 4 is added to the virtural offset for RETURN 2; 8 for RETURN 3; 12 for RETURN 4.)

ADRS--A list of addresses of actives. Return is picked by adding address in 'ADRS'; the offset for the letter and the offset for the specific return.

GOROUT--The routine to be executed when all the parameters have been processed. (Can be either an A-constant (local address) or a V-constant (another CSECT)).

EXAMPLE: (Approximately from BEGIN CSECT). WORK=XX would be interpreted as Return 2, Letter W;and would be found by adding the address in register 6 + 16 (BADRS) plus 4 (for RETURN2) plus Ø (virtual offset for letter W). The example also shows two ways of interleaving return addresses; either mesh pairs of singles and adjust the offsets (as in E's & W's); or put a filler in (as in P's).

REGISTER COMBASE contains the address of BGNCOMS.

```
BGNCOMS  DC  A(BNUML,BLETS,BRETNS,BOFSTS,BADRS,BGO)
BLETS    DC  C'ELFWP'
BNUML    DC  F'5'
BRETNS   DC  FL1'1Ø,1,8,4,5'
BADRS    DC  A(E1,W2,E3,F1,L4,P2,1,P4)
BOFSTS   DC  FL1,'Ø4,12,Ø,16'
```

Form used to describe a command:

Sequence	Letter	Returns (Hex)	Active	Offset	Name	Description
1	E	(8)	1	0	E1	
		(2)	3		E3	
2	L	(1)	4	4	L4	
3	F	(8)	1	12	F1	
4	W	(4)	2	0	W2	
5	P	(4)	2	16	P2	
		(1)	4		P4	

Table 5.8 Command Description Form

An example of Planning the Offset Structure (for the Tally Command, showing how the address list is intersperced):

		Addresses		Offsets	
0	TH1	60	TA3	H	0
4	TQ2 ← I	64	TA4	I	0
8	TQ3	68	TM1 ← M	P	12
12	TQ4 ← P	72	TM2	F	112
16	TPZ	76	TT1 ← T	0	20
20	1 ← O	80	TT2	S	24
24	TO2 ← S	84	TT3	C	36
28	TO2	88	TT4	N	44
				R	44
32	TS3	92	TG1 ← G	A	52
36	TS4 ← C	96	TD1 ← D	M	68
40	TC2	100	TX1 ← X	T	76
44	TC3 ← N,R	104	TY1 ← Y	G	92
48	TN2	108	TZ1 ← Z	D	96
52	TR3 ← A	112	TF1 ← F	X	100
56	TR4	116	TF2	Y	104
				Z	108

Table 5.9 Planning Offset Structure

5.5.5 Initializing Most initializing is done in EVQINITL. This CSECT basically initializes the 'GETMAIN'ed area referred to by register QUESBASE. The only trick is how tabs are initialized. EVQINITL sets register 14 to return to the main CSECT, and then branches to EVQTTL, which is the location of the TABREST command of the TALLY command. It issues a BR 14 and returns to EVQS.

5.5.6 I/O Operations The system allows the use of a number of standard types of data files:

a) Scan Files--(Special format files), contain the master data files and allow high-speed access based on access number,

b) Work Files--(QSAM) Standard sequential files, used to store the result of scanning a scan file, for input to the analysis routines,

c) Print files--(QSAM) Standard sequential files, used to store the output from the analysis routines. These are either output to a high-speed printer or to the terminal.

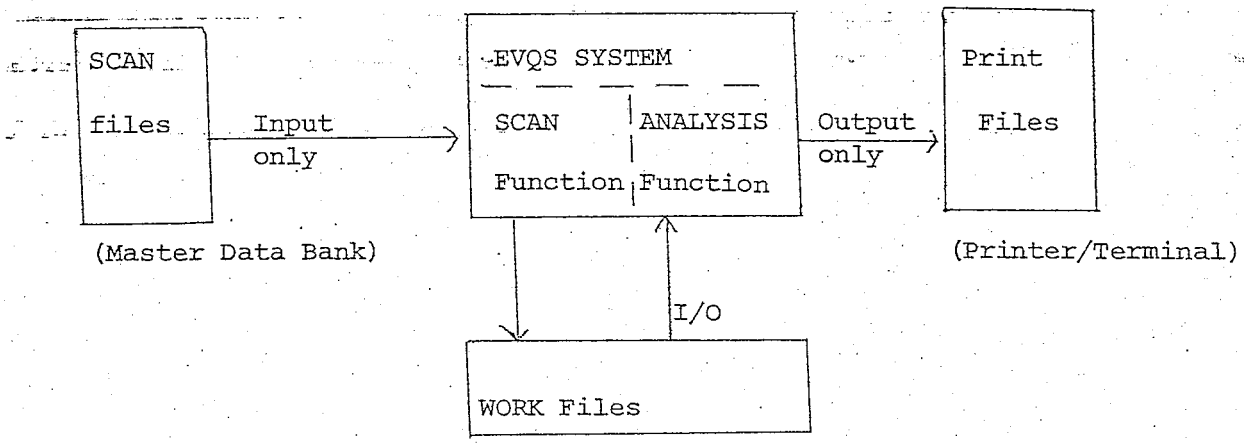


Figure 5.8 Standard Files

Various special provisions were needed to allow manipulation of the files.

5.5.6.1 Data Control Blocks The addresses of the Data Control Blocks (DCB) are stored in standard locations in the QUESLIST DSECT (as described in Figure 5.10 below).

Access Method	Name	Points to DCB for	Used by	Changeable Assign Command Name
BSAM	SYSIN	COMMAND INPUT	CONTROL CSECT	None
	SYSPRINT	TERMINAL OUTPUT	ALL ROUTINES	None
	MASPRINT	LARGE LISTINGS	LIST/TALLY	PRINT
QSAM	WORKOUT	WORK FILE FOR OUTPUT	SCAN	OUTPUT
	WORKIN	WORK FILE FOR INPUT	LIST/TALLY	INPUT
FSAM	SCANF	SCANNING FILE	SCAN	SCAN

Table 5.10 Standard DCB Address Variables

5.5.6.2 Associating DCB's with Files The standard locations that are changable are set to desired files using the 'ASSIGN' command from the set of files allowed; whose DCB's are in the EVQFILES CSECT.

File Number	DDNAME	Function	Unit/Disp.
1	EVQWKØ1	Scan file	Master data file
2	Ø2	Work file/Scan file	Temporary
3	Ø3	Work files	Temporary
4	Ø4	Work files	Temporary
5	Ø5	Work files	Permanent
P	MASPRINT	Output Files - Work/Print	Printer
T	SYSPRINT	Output Files - Work/Print	Terminal

Table 5.11 File Options

Function	Input Files Locations	Possible	Output Files Locations	Possible	Reason
SCAN	SCANF	1 (Master data) 2 (Temp. data)	WORKOUT	2 3-5 P/T	Build a subset of main data file. Build an input file for anal routine.
COPY	WORKIN	2-5	MASPRINT	{2-5 P/T}	Listing.
TALLY	WORKIN	2-5	SYSPRINT MASPRINT WORKOUT	T P/T {P/T 2-5}	Statistics (List only). Special Cases. Formated List.

Table 5.12 Detailed File Usage

5.5.6.3 I/O MACRO's Because the standard DCB names are not addresses of DCB's -- as per normal IBM conventions, the standard MACRO's, GET/PUT, cannot be used. Provision also had to be made to change the EODAD address (End of Data) in the file used for input. Arrangements had to be made to have files opened and rewound (closed) as required. (Full descriptions of the MACRO's are presented later in this manual.)

Input / Output operations were handled using the DOIO macro, which has a form similar to the standard GET/PUT MACRO's; eg., DOIO SYSPRINT,CARD. The operation the file is open for determines what happens. The EOF macro was written to specify EODAD exit addresses, EOF SYSPRINT,ENDFILE. A quick DCB macro was also written called QDCB, and it supports record formats: fixed block and unblocked. Any other RECFM's (other then FB or F)will cause the RECFM, BLKSIZE and LRECL fields to be left zero, thus allowing these to be filled from either the DD JCL card or from the file's catalogue entry.

Files are opened/closed in two ways: a) those with only one purpose are opened at system initialization times: CSECT EVQFILES is entered (it contains all the DCB's) and the normal open Macro is used, b) the work files that can be used either for input or output are opened and closed surrounding each use by duplicating the open/close macro in-line.* This requires three steps: 1) Set the first byte of the DCB variable in QUESLIST DSECT to indicate the operation to occur (See Table 5.13); 2) Load the address of the DCB variable into register 1; 3) Issue the appropriate supervisor call: Open- SVC 19; Close- SVC 20.

SVC Option Byte	Meaning	
X'8Ø'	Last file to be operated on.	
X'ØØ'	Input file.	Open only.
X'ØF'	Output file.	Open only.

Table 5.13 SVC Option Bytes

IMPORTANT NOTE: Since the WORKIN and WORKOUT files in QUESLIST DSECT are contigious, it is desirable to open both with a single SVC call. The SVC call does not check to see if both are the same DCB, as of this writing, using a 360/50 with OS/MVT release 21 under HASP 3.2. If both input and output were ASSIGN'ed the same file and opened in one SVC call, the whole 36Ø installation will crash by fouling up all the temporary file UFT's.

*See note following 'DCB Locations", Section. 5.5.6.4.